A Perfect Load Balancing Algorithm on Cube-Connected Cycles

Gene Eu Jan [†]Shao-Wei Leu [†]Cheng-Hung Li [&]Xiaoshe Dong
Institute of Electrical Engineering
National Taipei University
151 University Rd., San Shia, Taipei, 237 Taiwan
E-mail: gejan@mail.ntpu.edu.tw
[†]Department of Electrical Engineering
National Taiwan Ocean University
[&]Department of Computer Science and Technology
Xi'an Jiaotong University

Abstract: - A simple yet efficient algorithm was presented by Jan and Huang recently to distribute loads evenly on multiprocessor computers with hypercube interconnection networks. That algorithm was developed based on the well-known dimension exchange method. However, the error accumulation suffered by other algorithms based on the dimension exchange method is avoided by exploiting the notion of regular distributions, which are commonly deployed for data distributions in parallel programming. The cube-connected cycles (CCC) proposed by Preparata and Vuillemin are very similar in structure to the hypercube as an efficient general-purpose parallel system for its fixed-degree, and compact and regular layout. In this article, we propose a simple algorithm based on our previous work to distribute loads evenly on cube-connected cycles. This algorithm achieves perfect load balance over P processors with error of 1 and the worst-case time complexity of $O(M \log^2 P)$, where M is the maximum load assigned to each processor initially. More importantly, it could achieve perfect load balance over subcubes as well -- .if the cube is decomposed into two subcubes by bit 0, then the difference between the numbers of the total tasks in the subcubes is at most 1.

Key-Words: - multiprocessor, hypercube, interconnection network, cube-connected cycles, load balancing, parallel programming

1 Introduction

Distributing tasks evenly on processors is essential for multiprocessor computers, since load imbalance leads to low utilization of some processors. Load balancing on hypercubes has been studied extensively [1, 2, 6-10, 12, 14, 16]. The problem is often called the *token distribution problem* [1].

Most load balancing algorithms for hypercube and cube-connected cycles multiprocessor computers are developed based on the *dimension exchange method* [15]. The main advantage of this approach is that every processor can redistribute tasks to its neighboring processors without the information of global distributions of tasks. However, it can not always reach the perfect load balancing. Specifically, if *error* ξ is defined as the difference between the maximum number of tasks at any processors and the minimum number of tasks at any processors, then on P processors tokens, will be balanced with error $\xi = \log_2 P$ in the worst case. In order to balance loads evenly on multiprocessor systems, the information of global load distributions must somehow be available to processors. Techniques have been proposed to gather the global load distributions to achieve the perfect load balancing. However, preprocessing must be applied before tasks are redistributed based on the information [1].

Recently, we proposed an efficient algorithm to distribute loads evenly on multiprocessor computers with hypercube interconnection networks [8]. That algorithm was developed based upon the well-known dimension exchange method. However, the error accumulation suffered by other algorithms based on the dimension exchange method is avoided by exploiting the notion of *regular distributions*, which are commonly deployed for data distributions in parallel programming. That algorithm achieves perfect load balance over P processors with error of 1 and the worst-case time complexity of $O(M \log^2 P)$, where M is the maximum load assigned to each processor initially. Furthermore, perfect load balance is achieved over subcubes as well - once a cube is balanced, if the cube is decomposed into two subcubes by the lowest bit of node addresses, then the difference between the numbers of the total tasks of these subcubes is at most 1.

This article proposes a simple algorithm based on our previous work to distribute loads evenly on cube-connected cycles. The features of this algorithm are twofold:

- It does not require preprocessing to gather the global load distributions in order to achieve the prefect load balancing.
- The algorithm achieves perfect load balance over P processors with error of 1 and the worst-case time complexity of $O(M \log^2 P)$, where M is the maximum load assigned to each processor initially. In addition, perfect load balance is achieved over subcubes as well. Let C_{n-1}^0 and C_{n-1}^1 be the $(n-1)$-dimensional subcubes of an n-dimensional cube-connected cycles, C_n, i.e. the lowest bit of the node's hypercube address of every node is 0 in C_{n-1}^0, and 1 in C_{n-1}^1. Once H_n is balanced, the difference of the numbers of total tasks on C_{n-1}^0 and C_{n-1}^1 is at most 1. In addition, the same feature of perfect load balancing among subcubes holds for C_{n-1}^0 and C_{n-1}^1 as well.

The rest of this article will be organized as follows. Related work will be presented in Section 2. Section 3 briefly presents the definition of cube-connected cycles and some background information for the load balancing algorithm. Section 4 will outline the algorithm that balances the tasks on the cube-connected cycles, and the time complexity of the algorithm will be analyzed in Section 5. Section 6 concludes this article.

2 Related Work

Ranka *et al*. and Cybenko introduced the dimension exchange method for load balancing on hypercube [3, 14]. It is a simple but usually heuristic and has been adapted by many researchers to perform parallel load balancing. However, its worst-case error ξ is n on an n-dimensional hypercube. Plaxton developed a couple of load balancing algorithms based on the similar approach, and hence the error would be n as well [10]. Recently, Rim et al. adapted the dimension exchange method to distribute quantized loads with $\xi \le \lceil \log_2 P \rceil$ [15].

Raghavendra *et al*. proposed a distributed load balancing algorithm DBALANCE the concentration operation [12]. It is required that the set of overloaded processors S_o and the set of under-loaded processors S_u are known before DBALANCE can be performed to distribute tasks evenly. Furthermore, the total number of tasks N must be multiple of the number of processors P, i.e. $|S_o| = |S_u|$ where $|S_o|$ and $|S_u|$ denote the numbers of overloaded and underloaded processors, respectively.

Chlebus *et al*. improved the load balancing algorithms developed by Plaxton to avoid the error accumulation over dimensions [2, 10]. Similar to the algorithm presented in this paper, the worse-case performance of the algorithm is bounded by $O(M \log^2 P)$. However, Chlebus's algorithm must perform preprocessing to globally determine where loose tasks should be placed before tasks are redistributed by dimension exchanging.

Jan and Hwang recently presented a simple yet efficient algorithm to distribute loads evenly on multiprocessor computers with hypercube inter-connection networks [8]. That algorithm does not require preprocessing to gather global information of task distributions. Exploiting the regular distributions in parallel programming languages provides a convenient way to represent global task distributions. Besides, perfect load balancing is achieved not only on processors of the hypercube but also on subcubes.

3 Definitions and Background

3.1 Hypercubes

An n-dimension hypercube H_n consists of $P = 2^n$ processors. Each processor $p_m (0 \le m \le P)$ is identified by an n-bit binary address with the form of $m \equiv d_0 d_1 ... d_i ... d_{n-1}$, where $d_i = 0 | 1 (0 \le i \le n)$ and d_0 is the most significant bit (MSB). Processor $P_{d_0 d_1 ... d_i ... d_{n-1}}$ is connected to each processor in the set of processors $\left\{ P_{d_0 d_1 ... \overline{d_i} ... d_{n-1}} \middle| 0 \le i < n \right\}$, where $\overline{d_i}$ denotes the complement of bit d_i. As a result, processor $P_{d_0 d_1 ... d_i ... d_{n-1}}$ is connected to processor $P_{d_0 d_1 ... \overline{d_i} ... d_{n-1}}$ at dimension i.

3.2 Cube-Connected Cycles

In general, an n-dimensional cube-connected cycles, denoted by $CCC(n, 2^n)$, is constructed from an n-dimensional hypercube by replacing each node of the hypercube with a cycle of n node in the $CCC(n, 2^n)$ interconnection network [11]. A $CCC(n, 2^n)$ system is an interconnection network of computers in which processors are located at the nodes and communication channels between processors are the links. Every processor is a RAM (random access machine) with some local memory and can perform any of the basic operations such as addition, subtraction, etc., in one unit of time. Two nodes $p_{i,j}$ and $p_{i',j'}$ are linked using an edge in the $CCC(n, 2^n)$ interconnection network if and only if either: (1) $i = i'$ and $j - j' \equiv \pm 1 \bmod n$ or (2) $j = j'$ and i differs from i' in precisely the jth bit. Edges of the first type are called *cycle edges*, while edges of the second type are referred to as *hypercube edges*.

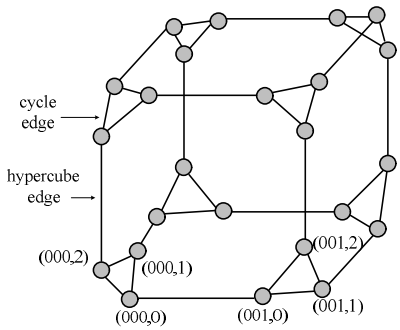


Figure 1: A 3-dimensional cube-connected cycles $CCC(3, 2^3)$ interconnection network.

For convenience, let CL_i denote the *i*th cycle and $p_{i,j}$ represent the *j*th processor in the *i*th cycles, where $0 \le i \le 2^n - 1$ and $0 \le j \le n-1$. Let $p_{i^{\bar{j}},j}$ represent the processor whose hypercube address *i* is different from that of $p_{i,j}$ in the *j*th bit. Based on this notation, processor $p_{i,j}$ is adjacent to $p_{i^{\bar{j}},j}$, $p_{i,j-1}$, and $p_{i,j+1}$. An example of the $CCC(3, 2^3)$ inter-connection network is shown in figure 1.

It is quite apparent from its construction that the cube-connected cycles is very similar in structure to the hypercube. In particular, any step of the *P*-node hypercube can be simulated in $P \log P$-node cube-connected cycles by using each $\log P$ cycle of the cube-connected cycles to simulate the action of the corresponding node of the hypercube [7].

3.3 Token Distribution Problem

The problem of distributing tasks evenly over multiprocessors is often called the token distribution problem if the following assumptions are taken

- there are no dependences between tasks. (i.e. tokens)
- each task takes a unit-time to execute.

Therefore, perfect load balancing is achieved when the error is at most 1 (i.e. $\xi \le 1$). In other words, each processor will have either $\lceil N/P \rceil$ or $\lfloor N/P \rfloor$ tasks, where *N* is the total number of tasks and *P* is the number of processors.

3.4 Perfect Load Balance

A perfect load balance means the tasks on the cube-connected cycles are distributed evenly on subcubes of all levels with error $\xi \le 1$. Specifically, let C_{n-1}^0 and C_{n-1}^1 be the (*n*-1)-dimensional sub-cubes of an *n*-dimensional cube-connected cycles, C_n, i.e. the lowest bit of the node hypercube address of every node is 0 in C_{n-1}^0, and 1 in C_{n-1}^1. Once C_n is

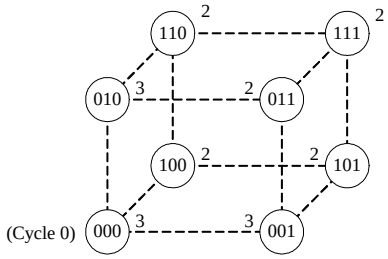


Figure 2: A 3-dimensional cube-connected cycles.

balanced, the difference between the total numbers of tasks on C_{n-1}^0 and C_{n-1}^1 is at most 1. In addition, the same load balancing feature among subcubes holds for C_{n-1}^0 and C_{n-1}^1 as well. In other words, if C_{n-2}^{00} and C_{n-2}^{10} are the subcubes of C_{n-1}^0, and C_{n-2}^{01} and C_{n-2}^{11} are the subcubes of C_{n-1}^1, then the numbers of total tasks on these four (*n*-2)-dimensional subcubes will be balanced with an error of at most 1 as well. Furthermore, the process can be iterated up to 1-dimensional subcubes and the same property will hold for all dimensions.

Consider the 3-dimensional cube-connected cycles, $CCC(3, 2^3)$ or C_3 in short, shown in Figure 2. The number beside each cycle denotes the total number of tasks on each cycle. When C_3 is divided into two subcubes, i.e. $C_2^0 \equiv \{000, 010, 100, 110\}$ and $C_2^1 \equiv \{001, 011, 101, 111\}$, the numbers of total tasks on C_2^0 and C_2^1 are 10 and 9 and hence the difference is 1. In addition, both 2-dimensional subcubes can be partitioned into 1-dimensional subcubes, i.e. $C_1^{00} \equiv \{000, 100\}$, $C_1^{10} \equiv \{010, 110\}$, $C_1^{01} \equiv \{001, 101\}$, and $C_1^{11} \equiv \{011, 111\}$. The total numbers of tasks on these 1-dimensional subcubes are 5, 5, 5, and 4, and the error is still 1. Furthermore, each of the 1-dimensional subcubes can be split into 0-dimensional subcubes, and hence there are total 8 subcubes - $C_0^{000} \equiv \{000\}$, $C_0^{100} \equiv \{100\}$, $C_0^{010} \equiv \{010\}$, $C_0^{110} \equiv \{110\}$, $C_0^{001} \equiv \{001\}$, $C_0^{101} \equiv \{101\}$, $C_0^{110} \equiv \{110\}$, and $C_0^{111} \equiv \{111\}$. Since the subcubes of all dimensions are balanced with the error of at most 1, the cube-connected cycles $CCC(3, 2^3)$ has reached a perfect load balance.

3.5 Dimension Exchange Method on Hypercubes

This method goes through all dimensions, from dimension 0 to *n*-1, and balances loads by redis-tributing the tasks across the links of each dimension.

For $i = 0$ to $n-1$ do

Balance the hypercube across dimension *i*
End do

Before load balancing is performed, each processor of the hypercube can be viewed as a 0-dimensional subcube H_0 with balanced load (i.e. tasks are distributed evenly over processors in the subcube since there is only one processor). Once iteration 0 is executed, each pair of 0-dimensional subcubes H_0^0 and H_0^1 that are connected by a link of dimension 0 will be combined into a 1-dimensional subcube H_1. Furthermore, the tasks of H_0^0 and H_0^1 will be redistributed through the link, and as a result H_1 is balanced. The same process will go on until load balancing is performed on all dimensions to form an *n*-dimensional hypercube with balanced load.

To balance 1-dimensional subcubes H_1 by redistributing tasks of 0-dimensional subcubes H_0^0 and H_0^1, results in balanced 1-dimensional subcubes with error $\xi \le 1$. In other words, if H_0^0 and H_0^1 have *x* and *y* tasks respectively, after redistribution, one of H_0^0 and H_0^1 will have $\lceil (x+y)/2 \rceil$ tasks, and the other will have $\lfloor (x+y)/2 \rfloor$ tasks. If $x+y$ is odd, the one of the two subcubes has one more task than the other. This extra task is usually called the *loose* task [2], and it can be placed at either of two subcubes and load balancing is still preserved.

The error will accumulate as a sequence of load balancing operations are performed through all dimensions, as redistributing tasks through every dimension might generate loose tasks. Specifically, after redistributing tasks across links of *n* dimensions, the hypercube will be balanced with an error $\xi \le n$.

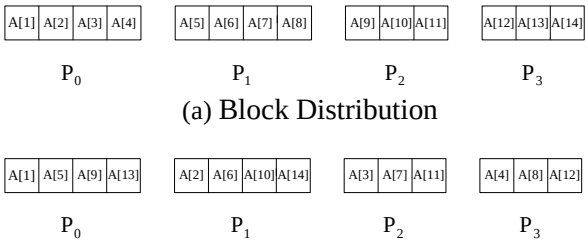
3.6 Regular Distributions

In order to perform load balancing on hypercube multiprocessors using the dimension exchange method without accumulating errors across dimensions, our previous paper exploits the notion of *regular distributions*, which are commonly used to specify the data distributions of arrays in parallel programming languages such as Fortran D [4] and HPF [5].

Definition 1 Regular Distributions

Let n_m be the number of array elements on processor $p_m (0 \le m < P)$. A distribution of *N* elements over *P* processors is called a *regular distribution* if

$$n_m = \begin{cases} \lceil N/P \rceil & \text{if } m < (N \% P) \\ \lfloor N/P \rfloor & \text{otherwise} \end{cases}$$



(a) Block Distribution

(b) Cyclic Distribution

Figure 3: Regular distributions over 4 processors

where % is the modulo function.

There are two regular data distributions that are commonly used by the parallel programming community, namely *block distributions* and *cyclic distri-butions* [12, 14]. A block distribution divides array elements into contiguous chunks of size N/P and assigns one block to each processor, whereas a cyclic distribution specifies a round-robin division of the array elements and assigns every *P*th element to the same processor. Figures 3(a) and (b) depict the layout of 14 elements of array *A* on 4 processors in a block distribution and a cyclic distribution, respectively.

The most significant feature of regular distributions that is exploited is that the number of tasks on every processor of a subcube can be obtained analytically, if the tasks are distributed over processors in the same way as array elements are partitioned in regular distributions. In other words, each processor can easily determine the numbers of tasks on all other processors according to Definition 1. Furthermore, the total number of tasks *N* and the number of processors in the subcube are the only pieces of information that every processor needs to know in order to determine how tasks are distributed over all processors.

Enforcing that tasks are partitioned over processors in regular distributions in every subcube can greatly simplify the problem of load balancing. Suppose the total numbers of tasks on *k*-dimensional subcubes H_k^0 and H_k^1 are N_k^0 and N_k^1, respectively. The addresses of nodes on H_k^0 and H_k^1 have the forms

$$\overbrace{xxx...x}^{k} 0 d_{k+1} ... d_{n-1}$$

and

$$\overbrace{xxx...x}^{k} 1 d_{k+1} ... d_{n-1}$$

The local addresses of processors in subcubes H_k^0 or H_k^1 are determined by $\overbrace{xxx...x}^{k}$ part of node addresses. When tasks of H_k^0 and H_k^1 are balanced over the edges of dimension *k*, the only piece of information

that H_k^0 and H_k^1 need to exchange before moving the tasks is the number of total tasks in its own subcube, i.e. N_k^0 and N_k^1, respectively. Every node in H_k^0 can then easily determine how the tasks are distributed in H_k^1 based on the information N_k^1: the first $(N_k^1 \% 2^k)$ processors in H_k^1 have $\lceil N_k^1/2^k \rceil$ tasks each, and the remaining processors have $\lfloor N_k^1/2^k \rfloor$ tasks each. Similarity, every node in H_k^1 knows exactly how the tasks are distributed in H_k^0. Therefore, every node in H_k^0 can use this information to decide how to move tasks between its corresponding node in H_k^1, and vice versa.

3.7 The Perfect Load Balancing Algorithm on Hypercubes

This algorithm was developed based upon the dimension exchange method. It goes through all the dimensions, from dimension 0 to *n*-1, and distributes the tasks across the edges of every dimension [8]. It first balances the load across the edges of dimension 0 to form subcubes of size 2^1 with balanced load. Similarly, the tasks of the subcubes of size 2^1 will then be distributed across the edges of dimension 1 to create balanced subcubes of size 2^2. This process will go on until the whole cube (i.e. 2^n) is balanced.

The algorithm works from dimensions 0 to $n-1$:

Algorithm 1: Perfect Load Balancing on Hypercubes
For $i = 0$ to $n-1$ do
Step 1: Balance the hypercube across dimension *i*.
Step 2: Redistribute loose tasks in each subcube to a regular distribution.
End do

At each iteration *i*, the tasks of two corresponding *i*-dimensional subcubes, say H_i^0 and H_i^1 (each has $P_i = 2^i$ processors), will be redistributed over the edges of dimension *i*. Suppose the numbers of total tasks on H_i^0 and H_i^1 are N_i^0 and N_i^1, respectively. A cube H_{i+1} with $N_i^0 + N_i^1$ tasks will be formed by merging H_i^0 and H_i^1 after the load is balanced by steps 1 and 2.

4 The Proposed Algorithm

Assume that (i, j) th processor $p_{i,j}$ has $l_{i,j}$ tasks and two buffers, $B_{i,j}(0)$ and $B_{i,j}(1)$. The initial data is stored in the buffer $B_{i,j}(1)$ and the data exchange or other computation is carried out in the buffer $B_{i,j}(0)$.

The algorithm is composed of three steps: compute the sum of tasks in each cycle, global balancing among hypercube, and local balancing among cycles. The perfect load balancing algorithm begins with step 1 by summing up all the tasks in each cycle, CL_i, to obtain the SUM_i in parallel. Each cycle in the cube-connected cycles can then be treated as a supernode in the hypercubes. Step 2 distributes the overloaded tasks from the overloaded cycles to the underloaded cycles using the Perfect Load Balancing on Hypercubes algorithm in a pipelining fashion. Local load balancing among cycles will be the final step to immigrate the overloaded tasks $(DIF(p_{i,j}) > 0)$ in each processor to its neighbor clock wisely in a pipelining fashion for *n* iterations. The load balancing in the cycle function and our proposed algorithm will be described as follows.

Function 1: Load Balancing in the Cycle.
Step 1: Obtain the load differences in each processor.
Step 1.1: Compute the SUM_i of tasks in each cycle, CL_i. Circulatory summing up all the tasks stored in the buffer $B_{i,j}(1)$ of processor $p_{i,j}$ in each cycle, CL_i, to obtain the SUM_i and then place it in the buffer $B_{i,j}(0)$.
Step 1.2: Compute the $AVG_i = \lfloor SUM_i/n \rfloor$ and then place it in the buffer $B_{i,j}(1)$ of each processor.
Step 1.3: Compute the $DIF(p_{i,j}) = l_{i,j} - AVG$ and then place it in the buffer $B_{i,j}(1)$ of each processor.
Step 2: Tasks Immigration.
Do in parallel
Simultaneously immigrate the overloaded tasks $(DIF(p_{i,j}) > 0)$ in each processor to its clockwise neighbor in a pipelining fashion for *n* iterations.

Algorithm 2: Perfect Load Balancing on CCC
Step 1: Compute the SUM_i of tasks in each cycle, CL_i.
Do in parallel
Circulatory summing up all the tasks stored in the buffer $B_{i,j}(1)$ of processor $p_{i,j}$ in each cycle, CL_i, to obtain and then place the SUM_i in the buffer $B_{i,j}(0)$.
Step 2: Global Balancing among Hypercube.

Call Algorithm 1: Perfect Load Balancing on Hypercubes (Each cycle in the cube-connected cycles can be treated as a supernode in the hypercubes and the tasks immigration in each processor of the cycles is in a pipelining fashion.)

Step 3: Local Load Balancing among Cycles.

Do in parallel CALL Function 1.

5 Performance Analysis

It is obvious that each cycle will obtain its sum of tasks simultaneously with the time complexity of $O(\log P)$ at step 1. If each processor is assigned at most M tasks before the load balancing algorithm is performed, then in each iteration i, there are at most $(M/2)\log P$ tasks for a cycle (supernode) to be immigrated across the hypercube edges of dimension i at step 2. Therefore, the time complexity contributed by step 2 will be $O(M\log^2 P + \log^3 P)$ in the worst case [8]. Regarding to the load balancing in the cycle function, step 1 is bounded by the time complexity of $O(\log P)$ as the first step in the perfect load balancing on cube-connected cycles algorithm. Furthermore, the operations that are performed at step 2 take at most $Mn < \mathrm{O}(M\log P)$ time units to immigrate overloaded tasks in any cycle simultaneously. Therefore, the worst-case time complexity of this algorithm is $O(M\log^2 P + \log^3 P)$. Furthermore, if $M > \log P$, then the time complexity could be dominated by the factor $O(M\log^2 P)$.

6 Concluding Remarks

This article proposed an algorithm that could distribute loads evenly on cube-connected cycles with the worst-case time complexity of $O(M\log^2 P)$. This algorithm could achieve a perfect load balance over P processors of the cube-connected cycles with an error of 1. More importantly, perfect load balance is achieved over subcubes as well -- once a cube is balanced, if the cube is decomposed into two subcubes by the lowest bit of node addresses, then the difference between the numbers of the total tasks of these subcubes is at most 1.

References:

[1] F. Meyer auf der Heide, B. Osterdiekhoff, and R. Wanka, "Strongly adaptive token distribution," in *Proceedings of the 20th International Colloquium on Automata, Languages, and Programming*, pp. 398–409, 1993.

[2] B. S. Chlebus, J. D. P. Rolim, and G. Slutzki. "Distrubuting tokens on a hypercube without error accumulation," in *Proceedings of the IPPS*, pp. 573–578, 1996.

[3] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *JPDC*, No. 7, pp. 279–301, 1989.

[4] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu, "Fortran D language specification," Technical report COMP TR90-141. Dept. of Computer Science, Rice Univ., Dec., 1990.

[5] High Performance Fortran Forum. High Performance Fortran Language Specification. *Scientific Programming*, 2(1-2): 1-170, 1993.

[6] J. JâJâ and K. W. Ryu, "Load balancing and routing on the hypercube and related networks," *JPDC*, No. 14, pp.431–435, 1992.

[7] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, San Mateo, California: Morgan Kaufmann, 1992.

[8] G. E. Jan and Y.-S. Hwang, "An efficient algorithm for perfect load balancing on hypercube multi- processors," *The Journal of Supercomputing*, Vol. 25, No. 1, May, 2003.

[9] G. E. Jan and M.-B. Lin, "Effective load balancing on highly parallel multicomputers based on superconcentrators," in *Proceedings of the ICPDS*, pp. 216–221, 1994.

[10] C. G. Plaxton, "Load balancing, selection and sorting on the hypercube," in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pp. 64–73, 1989.

[11] F. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *CACM*, vol. 24, No. 5, pp. 300-309, May, 1981.

[12] S. Raghavendra, C. S. Chalasani, and R. V. Boppmana, "Improved algorithms for load balancing in circuit switching hypercubes," in *Proceedings of the IPPS*, pp.537-542, 1991.

[13] Sanjay Ranka and Sartaj Sahni, *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition*, Springer Verlag, 1990.

[14] Sanjay Ranka, Youngju Won, and Sartaj Sahni, "Programming a hypercube multicomputer," *IEEE Software*, pp.69-77, 1988.

[15] Hwakyung Rim, Ju-Wook Jang, and Sungchun Kim, "An efficient dynamic load balancing using the dimension exchange method for balancing of quantized loads on hypercube multiprocessor," in *Proceedings of the IPPS*, pp. 708–712, 1999.

[16] J. Woo and S. Sahni, "Load balancing on a hypercube," in *Proceedings of the IPPS*, pp. 525–530, 1991.