JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **41,** 139–155 (1997)
ARTICLE NO. PC961295

RCC-Full: An Effective Network for Parallel Computations[1]

Mounir Hamdi*,[2] and Richard W. Hall†

*Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong; and
†Department of Electrical Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania 15261

A new interconnection network is proposed for the construction of a massively parallel computer system. The systematic construction of this interconnection network, denoted RCC-FULL, is performed by methodically connecting together a number of basic atoms where a basic atom is a set of fully interconnected nodes. Key communication characteristics are derived and evaluated for RCC-FULL and efficient routing algorithms, which need only local information to route messages between any two nodes, are also derived. An *O*(log (*N*)) sorting algorithm is shown for RCC-FULL and RCC-FULL is shown to emulate deterministically the CRCW PRAM model, with only *O*(log (*N*)) degradation in time performance. Finally, the hardware cost for the RCC-FULL is estimated as a function of its pin requirements and compared to that of the binary hypercube and most instances of RCC-FULL have substantially lower cost. Hence, RCC-FULL appears to be a particularly effective network for PRAM emulation, and might be considered as a universal network for future supercomputing systems.

1. INTRODUCTION

The theoretical RAM model closely matches real serial machines in that the observed performance of an algorithm on a serial machine can be expected to closely match the theoretical analysis. The parallel model that corresponds to the RAM is the parallel random access machine (PRAM) [4, 6, 11]. A PRAM is an idealized parallel machine which consists of a set of processors all of which have unit-time access to a shared memory. In every step of the PRAM, each of its processors may execute a private RAM instruction. In particular, the processors may all simultaneously access (read from or write into) the common memory. Various types of PRAMs have been defined, differing in the conventions used to deal with read/write conflicts, i.e., attempts by several processors to access the same variable in the same step. In the most restrictive model, exclusive read–exclusive write or EREW PRAMs, no variable may be accessed by more than one processor in a given step. In contrast, CRCW (concurrent read–concurrent write) PRAMs allow simultaneous reading as well as simultaneous writing of each variable, with some rule defining how to handle simultaneous writing of distinct variables to the same location.

From a programmer's perspective, it would be ideal to develop parallel algorithms for the PRAM. PRAMs are very convenient for expressing parallel algorithms since one may concentrate on decomposing the problem at hand, without having to worry about the communication between the tasks. For this reason there are many parallel algorithms written for the PRAM [2, 3, 11, 16, 18]. Unfortunately, the PRAM is not a very realistic model of parallel computation when the number of processors grows large. Present and foreseeable technology does not seem to make it possible to implement this model with more than a small number of processors. This has led many researchers to consider the emulation of the idealized parallel machine, the PRAM, on more realistic parallel machines using interconnection networks such as the hypercube, the mesh, and the mesh-of-trees [6, 20, 24, 32].

However, to the best of our knowledge, no practical interconnection network has been proposed to interconnect the processors of a parallel machine that can emulate *deterministically* any of the PRAM models of the same size in better than polylogarithmic degradation in time performance. We should mention here that there are some researchers who were able to show that certain networks are capable of emulating the PRAM in better than polylogarithmic degradation in time performance with high probability unlike our deterministic methods in this paper [21]. Further, there has been some research conducted that shows that the PRAM can be emulated on, for example, the reconfigurable mesh in constant time [27]. However, in this case and many other related cases the size of the emulating machine is substantially bigger than that of the emulated machine (PRAM) which reduces its practicality. Thus, in this paper, we investigate a class of modular networks, RCC-FULL, which is constructed incrementally by compounding certain primitive graphs together, and we find that this class is able to emulate PRAM models with better than polylogarithmic degradation in time performance. This also means that the RCC-FULL can emulate any interconnection network of

[1]This research was supported in part by the Hong Kong Research Council under Grant RGC/HKUST 100/92E and the Defense Advanced Research Projects Agency under Grant AFOSR-90-0310, monitored by the Air Force Office of Scientific Research.

[2]E-mail: hamdi@cs.ust.hk.

0743-7315/97 $25.00

the same size with better than polylogarithmic degradation in time performance.

This paper is organized as follows. In Section 2 we define the RCC-FULL interconnection network, we present some of its properties and key communication characteristics, and we present efficient routing strategies for the RCC-FULL. In Section 3 we demonstrate the efficient emulation of the PRAM models on the RCC-FULL. Finally, in Section 4 we analyze the hardware cost of the RCC-FULL and compare it to that of the binary hypercube as a function of their pin requirements.

2. RCC-FULL INTERCONNECTION NETWORK

In this section we present a systematic way of constructing the RCC-FULL class of interconnection networks, and we analyze some of its properties and communication characteristics. Then we present simple routing algorithms for the RCC-FULL, which need only local information to route messages between any two nodes in the network.

2.1. Construction

The proposed interconnection network, RCC-FULL, is a recursively compounded graph constructed incrementally by systematically connecting together a number of basic atoms. A basic atom is a set of fully interconnected nodes. An RCC-FULL is characterized by two parameters, (N_A, L), where N_A is the number of nodes in the basic atom and L is its level of recursion. An $(N_A, 0)$ RCC-FULL is a fully interconnected network with N_A nodes, this is simply a single atom. An $(N_A, 1)$ RCC-FULL is constructed by fully interconnecting N_A basic atoms creating a fully interconnected graph of basic atoms. Each node in an $(N_A, 1)$ RCC-FULL is specified by an n-bit binary number where $n = 2\log(N_A)$ and for convenience in exposition we assume N_A is a power of 2.[3] The most significant $\log(N_A)$ bits identify the atom that this node belongs to, and the least significant $\log(N_A)$ bits are used to distinguish among nodes within the same atom. The links between these basic atoms are formed by connecting PE (processing element) ij to PE ji for all i and j, with $i \neq j$, where i and j are binary numbers of $\log(N_A)$ bits each.[4] This is similar to the construction scheme in [13, 14]. These interatom links will be referred to as level 1 *transpose* links. In general, an (N_A, L) RCC-FULL of size N is constructed by fully interconnecting $N^{1/2}$ copies of $(N_A, L-1)$ RCC-FULLs where $N^{1/2}$ is the number of nodes in an $(N_A, L-1)$ RCC-FULL. Each node in an (N_A, L) RCC-FULL is specified by an m-bit binary number where $m = \log(N)$. The most significant (1/2)$\log(N)$ bits identify the $(N_A, L-1)$ RCC-FULL that this node belongs to, and the least significant (1/2)$\log(N)$ bits are used to distinguish among nodes within the same $(N_A, L-1)$ RCC-FULL. The links between these $(N_A, L-1)$ RCC-FULLs, referred to as level L *transpose* links, are formed by connecting PE ij to PE ji for all i and j, with $i \neq j$, where i and j are binary numbers of length (1/2)$\log(N)$ bits each. Figure 1 illustrates the construction of a (4, 2) RCC-FULL.

[3] All logarithms are taken to base 2.

[4] ij refers to concatenation of binary number i with binary number j.

2.2. Network Properties

The number of nodes, $N(N_A, L)$, of an (N_A, L) RCC-FULL is simply given by $N(N_A, L) = N(N_A, L-1) \times N(N_A, L-1)$ where $N(N_A, 0) = N_A$; thus

$$N(N_A, L) = N_A^{2^L}. \quad (1)$$

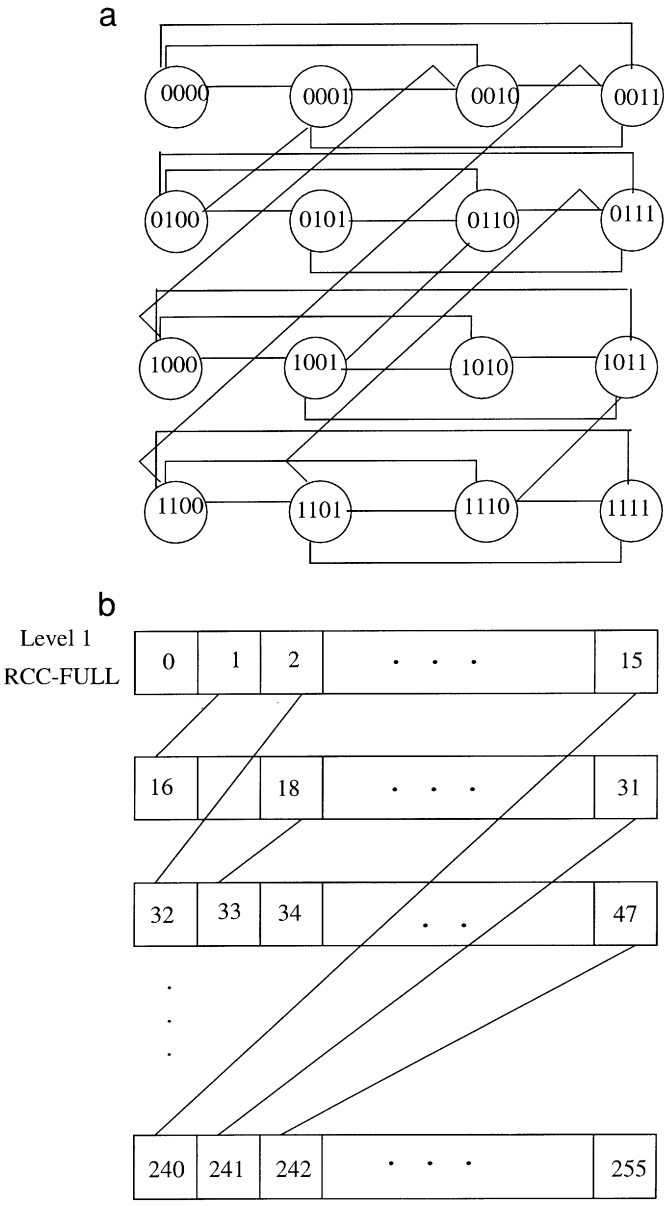

FIG. 1. Construction of a level 2 RCC-FULL where the basic atom is a 4-node fully connected network. (a) Level 1 RCC-FULL (PE indices are given in binary). (b) Level 2 RCC-FULL (PE indices are given in decimal).

TABLE I
Some Key Characteristics of the RCC-FULL and the Hypercube

Network	Number of nodes	Degree	Diameter	Network	Number of nodes	Degree	Diameter
(4,0) RCC-FULL	4	3	1	Hypercube	4	2	2
(4,1) RCC-FULL	16	4	3	Hypercube	16	4	4
(4,2) RCC-FULL	256	5	7	Hypercube	256	8	8
(4,3) RCC-FULL	65,536	6	15	Hypercube	65,536	16	16

The diameter of an (N_A, L) RCC-FULL, defined as the maximum number of links that must be traversed through the shortest path between any two nodes, is denoted $D(N_A, L)$. From the construction of the RCC-FULL we see that $D(N_A, L) = 2D(N_A, L-1) + 1$, where $D(N_A, 0) = 1$. Thus the diameter of an (N_A, L) RCC-FULL is given by

$$D(N_A, L) = 2^{L+1} - 1. \tag{2}$$

Hence if L is held constant and the network size grows by increasing the atom size N_A, the diameter of RCC-FULL is $O(1)$.

The degree of an (N_A, L) RCC-FULL, denoted $D(N_A, L)$, grows by 1 for each additional level of the construction. Thus $\Delta(N_A, L) = \Delta(N_A, L-1)+1$, where $\Delta(N_A, 0) = N_A - 1$ and

$$\Delta(N_A, L) = N_A + L - 1. \tag{3}$$

Table I compares favorably the characteristics of the RCC-FULL to that of the hypercube. Thus, even though the basic atom is a fully connected network, which is an expensive network, the way these basic atoms are interconnected together and their relatively smaller size makes RCC-FULL a fairly practical network as compared to the hypercube.

An (N_A, L) RCC-FULL of size N can be viewed as a network containing $N^{1/2}$ rows of PEs, where each row is an $(N_A, L-1)$ RCC-FULL, and $N^{1/2}$ columns of PEs where the rows are fully interconnected together (see Fig. 1). However, if we perform a parallel exchange operation of data in the PEs that are connected by a level L *transpose* link, this transposes the whole network and with this transpose operation all the columns become effectively fully connected also. This is a very useful property of RCC-FULL since the PEs of the columns are not directly interconnected; thus if we have to perform operations between the PEs within the columns, we transpose the whole network and all the columns' PEs would be able to utilize the same connection structure as the PEs within the rows of the network. Such a *transpose* characteristic makes writing algorithms for the RCC-FULL quite easy as it hides the asymmetry of the network.

2.3. Network Communication Measures

The capacity of an interconnection network to deliver a high volume of messages per unit time is another key performance measure. This factor is often used to establish lower bounds on the performance of parallel algorithms and in VLSI implementations. Algorithms, like sorting and general divide-and-conquer approaches, usually need to transfer large quantities of data from one region of the network to the other. There are many interconnection networks that have communication diameter $O(\log(N))$, but this small diameter does not guarantee logarithmic, or even polylogarithmic time complexity for algorithms that need a high transfer of data between different regions of the network. There is no one definition for measuring message capacity. We have chosen three distinct measures to consider: bisection bandwidth, message traffic density, and queuing delay.

2.3.1. Bisection Bandwidth

The bisection bandwidth, BB, of an interconnection network is the minimum number of links cut when a network is partitioned into two equal halves over all partitions. This measure gives lower bounds for certain parallel algorithms where large numbers of messages must be sent between two halves of the network during algorithm execution. When a fully connected network of size M is partitioned into two equal halves, the minimum number of links cut is equal to $M^2/4$. Thus, if in forming the partition we do not dichotomize any of the $N^{1/2}$ rows of an (N_A, L) RCC-FULL of size N nodes, the number of links cut when it is partitioned into two equal halves is $N/4$, since the $N^{1/2}$ rows are fully interconnected. Moreover, if we do not dichotomize any of the $N^{1/2}$ columns of the (N_A, L) RCC-FULL, the number of links cut when it is partitioned into two equal halves is $>N/4$ since the $N^{1/2}$ columns are fully connected (e.g., see Fig. 1) and links will also be cut across the rows. Thus, for an (N_A, L) RCC-FULL of size N, $BB = N/4$, if we do not dichotomize any of the $N^{1/2}$ rows or dichotomize any of the $N^{1/2}$ columns. If we allow the dichotomization of the rows and columns of an (N_A, L) RCC-FULL in the same partition, it is not straightforward to find the exact number of links cut when it is partitioned into two equal halves. However, the number of links cut in that case will be higher than $N/4$. This leads to the following corollary.

Corollary 1. *For an (N_A, L) RCC-FULL of size N the bisection bandwidth, $BB \geq N/4$, if we do not dichotomize any of the $N^{1/2}$ rows or any of the $N^{1/2}$ columns.*

The bisection bandwidth of a hypercube of size N is $N/2$ [13]. Hence, the bisection bandwidth of an (N_A, L) RCC-FULL is close to that of the hypercube for any value of L. Further, the bisection bandwidth of the RCC-FULL grows linearly as a function of N.

2.3.2. Message Traffic Density

The message traffic density gives an estimate on the average number of messages passing through a bidirectional link of the network, when every node of the network communicates with every other node in the network[5] [5, 10, 15]. The message traffic density is denoted by ρ and is defined by

$$\rho \equiv \frac{Average\ Message\ Distance \times Number\ of\ Nodes}{Number\ of\ Links}. \tag{4}$$

The average message distance in a network is the summation of distances between all possible pairs of nodes divided by the number of nodes in the network, where we allow a source and a destination node to be the same. The average message distance can be a better indicator of the communication efficiency of the network than its diameter. Now, let us determine the average distance of an (N_A, L) RCC-FULL, $\overline{D}_L$, under *uniform* probability. That is, the probability of a source node communicating with any destination node is the same. This is the most general form of finding average distance. If we assume that there is a *locality*, or *sphere* of communication [9], this will only favor the RCC-FULL since it is strongly connected locally through the topology of its basic atoms. We let s be the source node, d the destination node, and $Prob(event)$ the probability that an event occurs. Then, by examining the topology of the RCC-FULL, we find:

$$\begin{aligned}\overline{D}_L = {} & Prob(s \text{ and } d \text{ are identical}) \times 0 \\ & + Prob(s \text{ and } d \text{ are in the same row}) \times \overline{D}_{L-1} \\ & + Prob(s \text{ and } d \text{ are in different rows}) \\ & \times f(s, d).\end{aligned} \tag{5}$$

The value of $f(s, d)$ depends on the location of s and d, as developed in the following. Let s and d belong to distinct rows, r_s and r_d respectively, as illustrated in Fig. 2. Let p_1 (belonging to r_s) and p_2 (belonging to r_d) be the nodes connected by the transpose link connecting r_s and r_d. Then we can have the following four cases:

1. $s = p_1$ and $d = p_2$
2. $s \neq p_1$ and $d = p_2$
3. $s = p_1$ and $d \neq p_2$
4. $s \neq p_1$ and $d \neq p_2$.

The four cases cover all possible locations of s and d when they are in different rows and the probability of the occurrence of each case, with the associated average distance, is given below:

$$\begin{aligned}Prob(s = p_1 \text{ and } d = p_2) &= 1/N^{1/2} \times 1/N^{1/2} \\ &= 1/N \\ prob(s \neq p_1 \text{ and } d = p_2) &= (1 - 1/N^{1/2}) \times 1/N^{1/2} \\ &= (N^{1/2} - 1)/N \\ prob(s = p_1 \text{ and } d \neq p_2) &= 1/N^{1/2} \times (1 - 1/N^{1/2}) \\ &= (N^{1/2} - 1)/N \\ prob(s \neq p_1 \text{ and } d \neq p_2) &= (1 - 1/N^{1/2}) \times (1 - 1/N^{1/2}) \\ &= ((N^{1/2} - 1)/N^{1/2})^2.\end{aligned}$$

The above probabilities are used to determine $f(s, d)$ in the following manner:

$$\begin{aligned}f(s, d) = {} & Prob(s = p_1 \text{ and } d = p_2) \times 1 \\ & + prob(s \neq p_1 \text{ and } d = p_2) \\ & \times (1 + \overline{D}_{L-1}) + prob(s = p_1 \text{ and } d \neq p_2) \\ & \times (1 + \overline{D}_{L-1}) + prob(s \neq p_1 \text{ and } d \neq p_2) \\ & \times (1 + 2\overline{D}_{L-1}).\end{aligned} \tag{6}$$

Next, we derive the probabilities of occurrence of each of the events needed to find the average distance, $\overline{D}_L$, as given by Eq. (5) above:

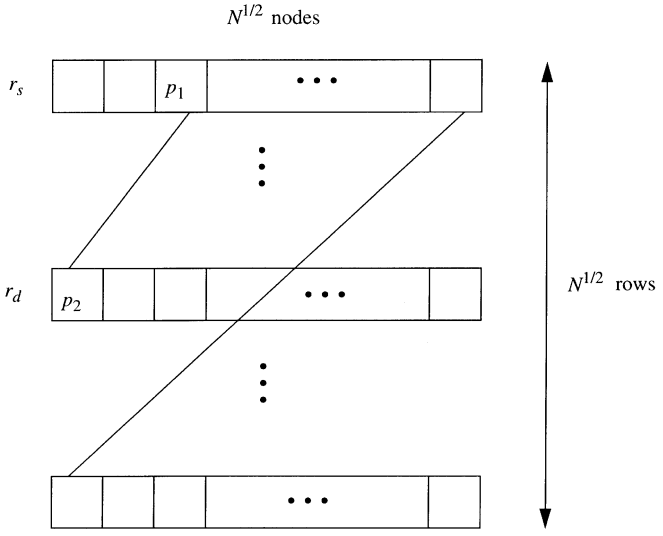

FIG. 2. Possible locations of source PE and a destination PE when they are in different rows.

[5]Some authors define the message traffic density by (Average Message Distance) (Number of Nodes) (Number of Nodes − 1)/(Number of Links). Our definition of message traffic density is given by Eq. (4) and is based on the definition given by [5, 10, 15].

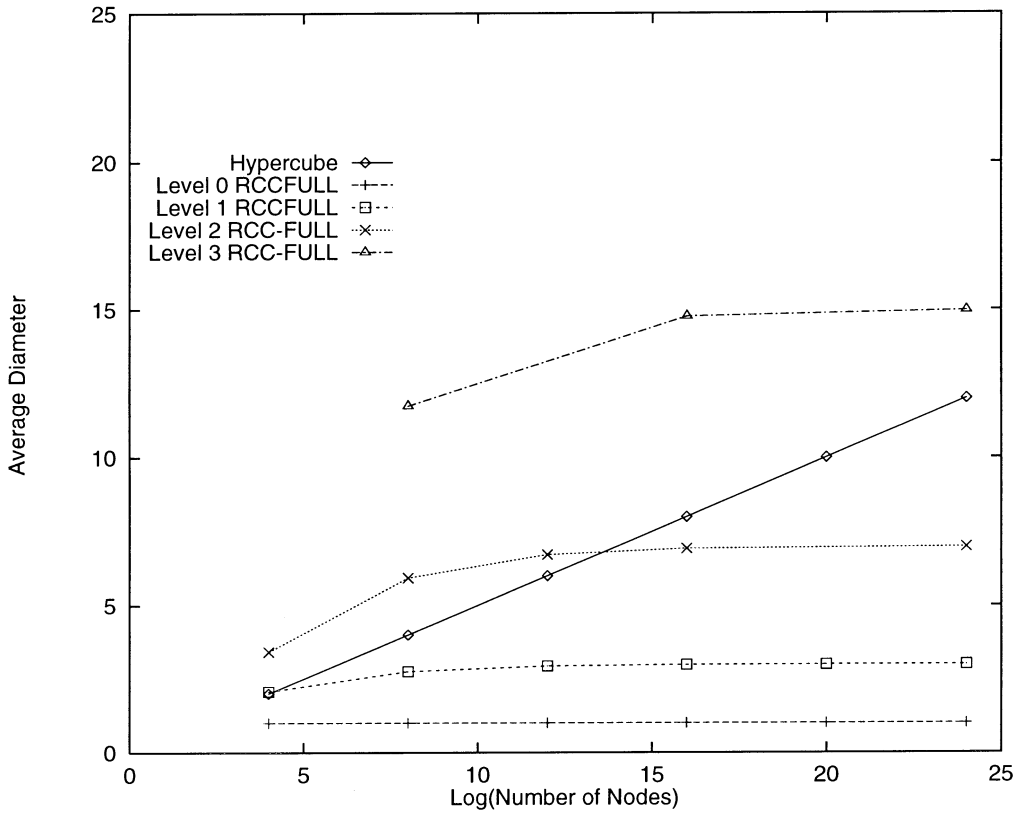

FIG. 3. Average distance of RCC-FULL and hypercube.

$$Prob(s \text{ and } d \text{ are identical}) = 1/N^{1/2} \times 1/N^{1/2} = 1/N$$

$$Prob(s \text{ and } d \text{ are in the same row but not identical}) = 1/N^{1/2} \times (1 - 1/N^{1/2}) = (N^{1/2} - 1)/N$$

$$Prob(s \text{ and } d \text{ are in different rows}) = 1 \times (1 - 1/N^{1/2}) = (N^{1/2} - 1)/N^{1/2}.$$

Thus, having identified all the terms needed to derive the average distance, $\overline{D}_L$, we substitute them into Eq. (5) to get

$$\begin{aligned}\overline{D}_L &= 1/N \times 0 + (N^{1/2} - 1)/N \\ &\quad \times \overline{D}_{L-1} + (N^{1/2} - 1)/N^{1/2} \\ &\quad \times [1/N \times 1 + (N^{1/2} - 1)/N \\ &\quad \times (1 + \overline{D}_{L-1}) + (N^{1/2} - 1)/N \times (1 + \overline{D}_{L-1}) \\ &\quad + [(N^{1/2} - 1)/N^{1/2}]^2 \times (1 + 2\overline{D}_{L-1})].\end{aligned}$$

$$\overline{D}_L = \left(2 - \frac{3}{N^{1/2}} + \frac{1}{N}\right) \overline{D}_{L-1} + 1 - \frac{1}{N^{1/2}}. \tag{7}$$

The average distance of a hypercube of size N, $\overline{D}_H$, is given by [5]

$$\overline{D}_H = \frac{\sum_{i=0}^{\log(N)} \binom{\log(N)}{i} i}{N} = \frac{1}{2}\log(N). \tag{8}$$

Figure 3 shows the average distance of the (N_A, L) RCC-FULL for L = 0, 1, 2, and 3 and compares it to that for the hypercube. The average distances of the $(N_A, 0)$ and $(N_A, 1)$ RCC-FULL are superior to those of the hypercube. The average distance of the hypercube is better than that of the $(N_A, 2)$ RCC-FULL only for small network sizes. Finally, the average distance of the hypercube is better than that of the $(N_A, 3)$ RCC-FULL, especially for small network sizes. Thus, with the RCC-FULL, we have flexibility in tuning the average distance performance of the system by choosing the appropriate network level, L. This flexibility is not present in the hypercube network. Moreover, if we assume that the probability of local message traffic is higher than that of global message traffic [9], then the average diameter of the RCC-FULL would tend to be more attractive than that of the hypercube since the RCC-FULL is strongly connected locally, as basic atoms are fully connected.

The next variable needed to evaluate the message traffic density is the total number of links for each network. First, let us determine the number of links for an (N_A, L) RCC-FULL of size N. When L = 0, RCC-FULL is a fully connected network, and the number of links is $N(N-1)/2$. In general, the number of links, $NL(N_A, L)$, of (N_A, L) RCC-FULL of size N is given by the total number of links within all rows, $N^{1/2} \times NL(N_A, L-1)$ plus the total number of level L *transpose* links. This leads to

$$\begin{aligned}NL(N_A, L) &= N^{1/2} \times NL(N_A, L-1) \\ &\quad + \tfrac{1}{2}N^{1/2}(N^{1/2} - 1).\end{aligned} \tag{9}$$

This would give us

$$NL(N_A, 1) = 1/2(N^{3/2} - N) + 1/2N^{1/2}(N^{1/2} - 1)$$
$$NL(N_A, 2) = 1/2(N^{5/4} - N^{3/4}) + 1/2N^{1/2}(N^{1/2} - 1)$$
$$NL(N_A, 3) = 1/2(N^{9/8} + N - N^{7/8} - N^{3/4}) + 1/2N^{1/2}(N^{1/2} - 1).$$

The number of links in a hypercube of size N, $NL_{\text{Hypercube}}$, is given by

$$NL_{\text{Hypercube}} = \tfrac{1}{2}N \log N. \tag{10}$$

Figure 4 compares the total number of links for an (N_A, L) RCC-FULL, where L = 0, 1, 2, and 3 to that for the hypercube as a function of the network sizes. The $(N_A, 0)$ and $(N_A, 1)$ RCC-FULLs have more links than the hypercube. The $(N_A, 2)$ RCC-FULL has more links than the hypercube only for small network sizes. Finally, the $(N_A, 3)$ RCC-FULL uses fewer links than the hypercube. We have to note that even though the total number of links gives a rough measure of the hardware cost of a network, it is not a complete measure. In Section 4 we will suggest that the hardware cost can be more accurately represented using packaging pin requirements analysis.

Having calculated the average distance and the number of links of the RCC-FULL and the hypercube, we can easily compute and compare their message traffic densities. The message traffic density of an (N_A, L) RCC-FULL of size N is given by

$$\rho_{\text{RCC-FULL}} = \frac{((2 - 3/N^{1/2} + 1/N)\overline{D}_{L-1} + 1/N^{1/2}) \times N}{N^{1/2} \times NL_{L-1} + \frac{1}{2}N^{1/2}(N^{1/2} - 1)}. \tag{11}$$

The message density of a hypercube of size N is given by

$$\rho_{\text{Hypercube}} = \frac{\frac{1}{2}\log(N) \times N}{\frac{1}{2}\log(N) \times N} = 1. \tag{12}$$

Figure 5 depicts the variation of ρ as a function of the network size N for both the RCC-FULL and the hypercube. The traffic density of an $(N_A, 0)$ and $(N_A, 1)$ RCC-FULL is better than that of the hypercube for all sizes of N. The traffic density of an $(N_A, 2)$ RCC-FULL is better than that of the hypercube for large network sizes. Finally, the message traffic density of the hypercube is better than that of the $(N_A, 3)$ RCC-FULL for all cases considered. Again, the RCC-FULL topology gives us more flexibility in choosing an appropriate message traffic density depending on the level of recursion, L. Moreover, the message traffic density of the RCC-FULL decreases as the size of the network increases for all levels of

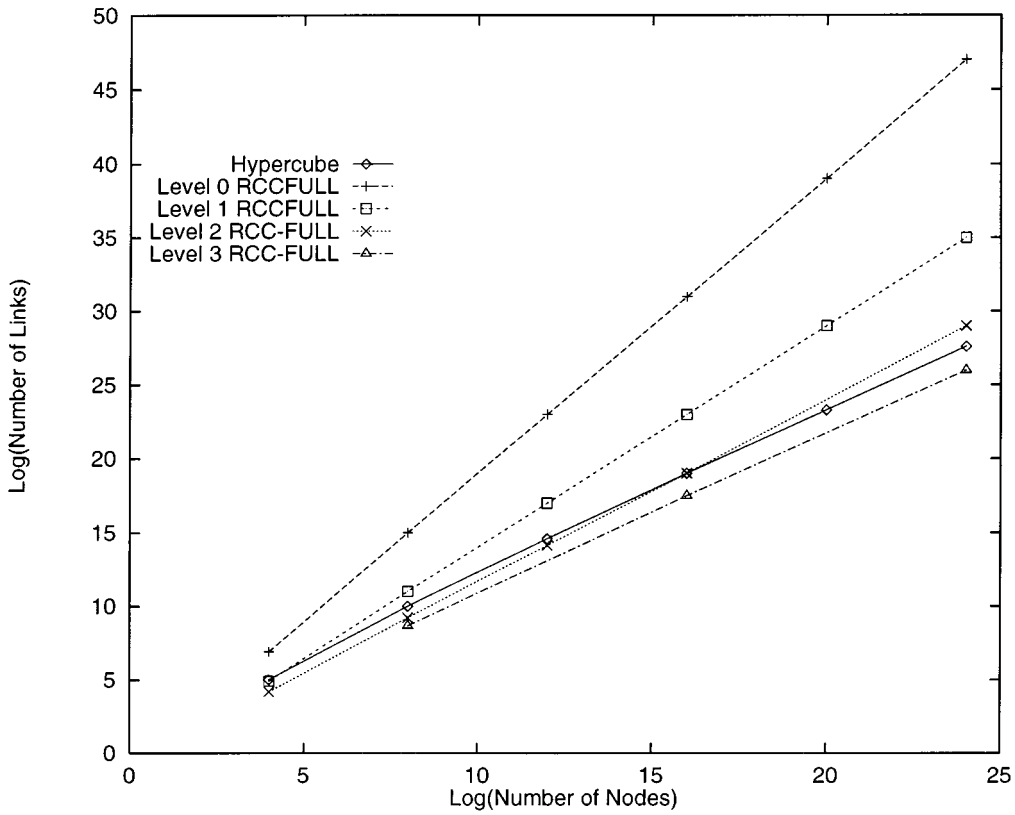

FIG. 4. Number of links of RCC-FULL and hypercube.

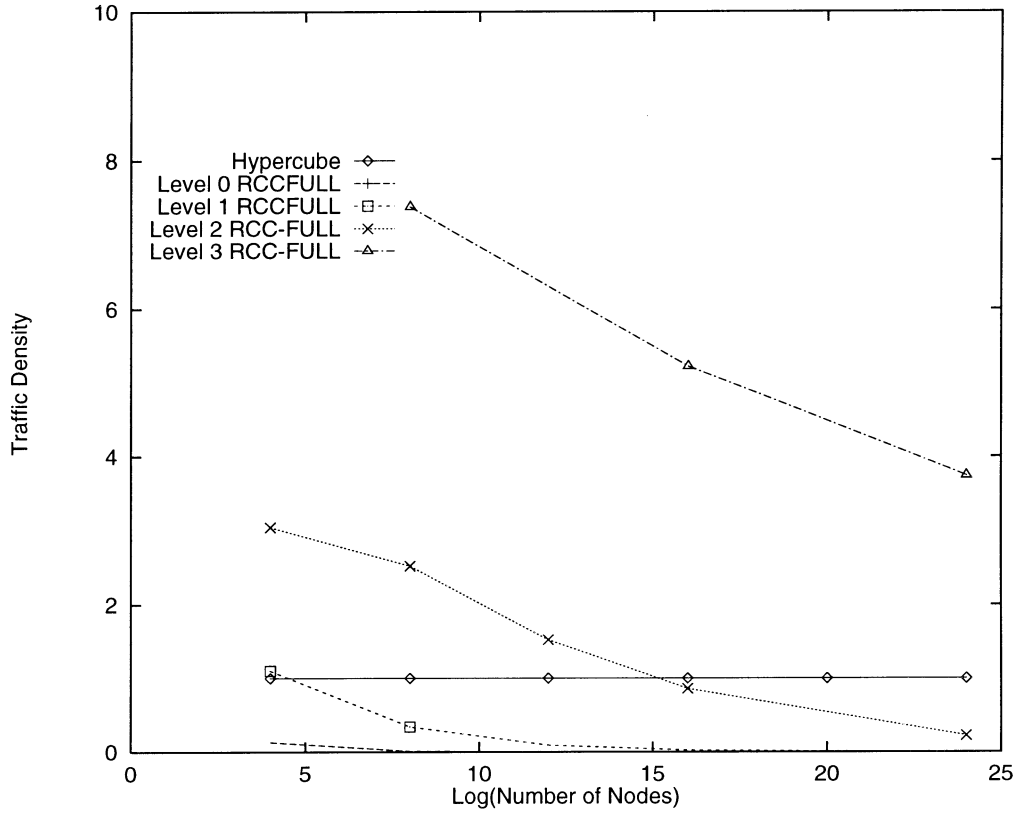

FIG. 5. Traffic density of RCC-FULL and hypercube.

recursion, L. This property is desirable for constructing massively parallel computers.

2.3.3. Queuing Time Delay

The average distances of the RCC-FULL and the hypercube give us a measure of their performance under the assumption that there is no message contention. In order to take message contention into consideration, we use a simple model for queuing analysis which has been adopted by many researchers [5, 9, 10]. An RCC-FULL can be modeled as a communication network with the ith channel represented as an M/M/1 system with Poisson arrivals at a rate λ_i and exponential service time of mean $1/\mu c_i$. The variable μ is the average service rate and c_i is the capacity of the ith channel. The assumptions made in [5, 9, 10] are repeated here for clarity:

1. Each node is equally likely to send a message to every other node in a fixed time period.
2. The routing is fixed.
3. The load is evenly distributed; i.e., λ_i is the same for all i.
4. The capacity of each link has been optimally assigned.
5. The cost per capacity per link is unity.

Under the above conditions, the delay of the network is given by

$$T = \overline{D}\left(\sum_{i=1}^{M}\left(\frac{\lambda_i}{\lambda}\right)^{1/2}\right)^2 \Big/ \mu C(1-\overline{D}\delta), \qquad (13)$$

where

$$M = \text{total number of directed links}$$
$$\lambda = \sum_{i=1}^{M}\lambda_i = M\lambda_i$$
$$\delta = \text{Utilization factor.}$$
$$C = \sum_{i=1}^{M} c_i = \text{total capacity of the network.}$$

Thus, the above equation can be simplified, and the queuing delay is given by

$$T = \frac{\overline{D}M}{\mu C(1-\overline{D}\delta)}. \qquad (14)$$

For an (N_A, L) RCC-FULL, $\overline{D} = \overline{D}_L$ and $M = 2NL(N_A, L)$ are given by Eq. (7) and Eq. (9), respectively. Figure 6 illustrates the variation of the normalized queuing

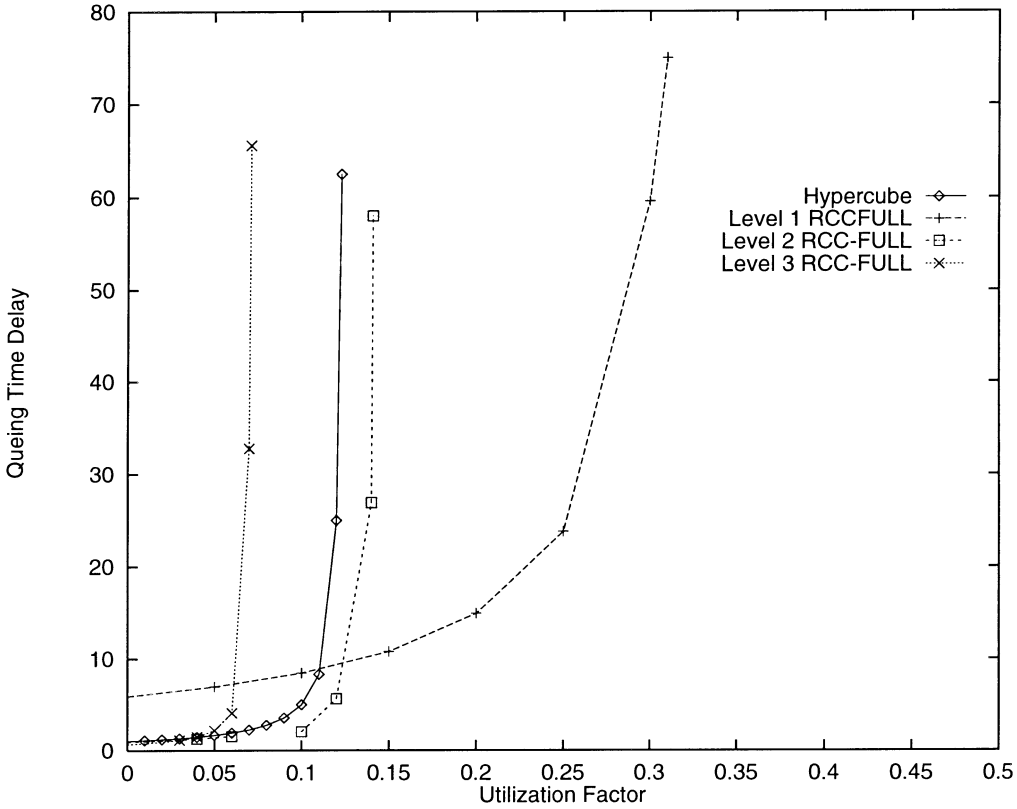

FIG. 6. Queuing delay of RCC-FULL and hypercube.

delay versus δ for the RCC-FULL and the hypercube. In this figure, a typical massively parallel network is assumed with size N = 64K processors. The queuing delay increases exponentially with increasing utilization and saturates at a particular load. As shown in the figure, the $(N_A, 1)$ and the $(N_A, 2)$ RCC-FULLs saturate for higher values of network utilization as compared to the hypercube. However, the $(N_A, 3)$ RCC-FULL saturates for lower values of network utilization than the hypercube.

Thus, by analyzing the communication measures of the RCC-FULL and the hypercube, we can conclude that RCC-FULL appears to be superior to the hypercube when $L \leq 2$. Moreover, with the RCC-FULL, we have flexibility in tuning the performance of the system by choosing the appropriate network level, L.

2.4. Routing

One of the desirable characteristics of a large network of processors is the ability of the processors to route messages without total knowledge of all the details in the network [1, 30]. In this section, we propose three routing algorithms for the RCC-FULL which can be easily implemented at each processor, and which require only the source address and the destination address to perform the local routing of messages at any node in the network. An (N_A, L) RCC-FULL with N nodes can be thought of as a network containing $N^{1/2}$ rows of PEs, each row being an $(N_A, L-1)$ RCC-FULL, and $N^{1/2}$ rows of PEs are fully interconnected together. Thus, in all the routing algorithms we describe below, the source node for the message is PE $i_1 j_1$, and the destination node for the message is PE $i_2 j_2$ where i_1, j_1, i_2, j_2 are binary numbers of (1/2)$\log(N)$ bits each, where i_1 and i_2 indicate the row addresses and j_1 and j_2 indicate the column addresses. Further, we assume that each PE can simultaneously use all its links for sending and receiving messages. This is denoted in the literature as a multiaccepting PE.

ALGORITHM 1. To send a message, m, from a source node to a destination node, Algorithm 1 performs the following steps:

1. PE $i_1 j_1$ sends m to PE $i_1 i_2$.
2. PE $i_1 i_2$ sends m to PE $i_2 i_1$.
3. PE $i_2 i_1$ sends m to PE $i_2 j_2$.

Figure 7 illustrates this movement. For an $(N_A, 1)$ RCC-FULL, steps 1 and 3 are routing within a fully interconnected network. Step 2 is one routing step along a transpose link for all levels of the RCC-FULL. In general, for an (N_A, L) RCC-FULL, steps 1 and 3 are $(N_A, L-1)$ RCC-FULL routing. Via this recursion the routing algorithm is fully defined. This is similar to the MIAM routing in [13]. Algorithm 1 can have a congestion problem when there is a high transfer of data

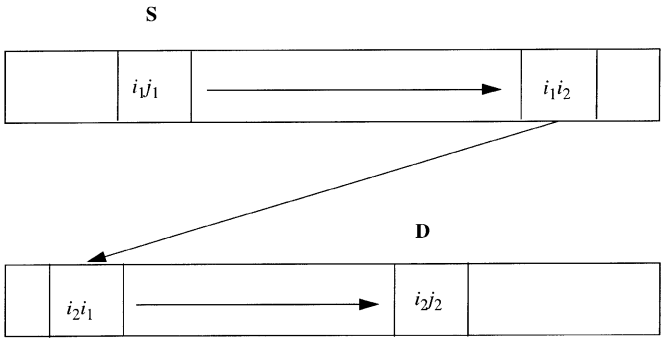

FIG. 7. Algorithm 1 routing strategy from source PE, **S**, to destination PE, **D**.

between two rows where all the messages have to use the same transpose link. This results in an $O(N^{1/2})$ worst case routing time. Further, since we are using multiaccepting PEs, we would not have any congestion problem in the basic atoms since they are fully connected. Thus, the only congestion problem that occurs in Algorithm 1 is in its second step, that is, while using the transpose links.

ALGORITHM 2. The second routing algorithm has been identified to provide alternate paths to solve the congestion problem that could occur by using Algorithm 1. Algorithm 2 routes a message, m, from a source node to a destination node by performing the following steps:

1. PE i_1j_1 sends m to PE j_1i_1.
2. PE j_1i_1 sends m to PE j_1i_2.
3. PE j_1i_2 sends m to PE i_2j_1.
4. PE i_2j_1 sends m to PE i_2j_2.

Figure 8 illustrates this routing movement. For an $(N_A, 1)$ RCC-FULL, steps 2 and 4 are routing within a fully connected network. Steps 1 and 3 are one routing step each along a transpose link for all levels of RCC-FULL. In general, for an (N_A, L) RCC-FULL, steps 2 and 4 are routing within an $(N_A, L-1)$ RCC-FULL. This is similar to the SS routing in [13].

However, Algorithm 2 can have a congestion problem on its own. Specifically, when there is a high transfer of messages between a column and a row, as all the messages have to traverse the same transpose link resulting in a worst case routing time of $O(N^{1/2})$.

Both Algorithm 1 and Algorithm 2 are examples of oblivious routing algorithms where the path followed by a message depends on the source address and the destination address, not the message distribution or congestion. Leighton has shown that for the 1-to-1 message routing problem posed in our analysis, the required number of routing steps $\geq N^{1/2}/2d$ for a network of size N and degree d [17]. For an (N_A, L) RCC-FULL this produces a lower bound of $N^{1/2}/2(N_A + L - 1)$. Both Algorithm 1 and Algorithm 2 have a worse case routing time of $O(N^{1/2})$. Thus, they fail to reach the optimal oblivious performance. Next we will demonstrate a significantly better routing performance with a combination of Algorithm 1 and Algorithm 2 which may no longer be considered oblivious.

Both routing algorithms, Algorithm 1 and Algorithm 2, complement each other; i.e., each algorithm could solve its congestion problem which results in the worst case routing time of $O(N^{1/2})$ if it has the chance to use the other algorithm for that specific situation. This leads us to propose a third routing algorithm, Algorithm 3, which has $O(N^{1/4})$ worse case routing time. It is a three-phase routing algorithm, where

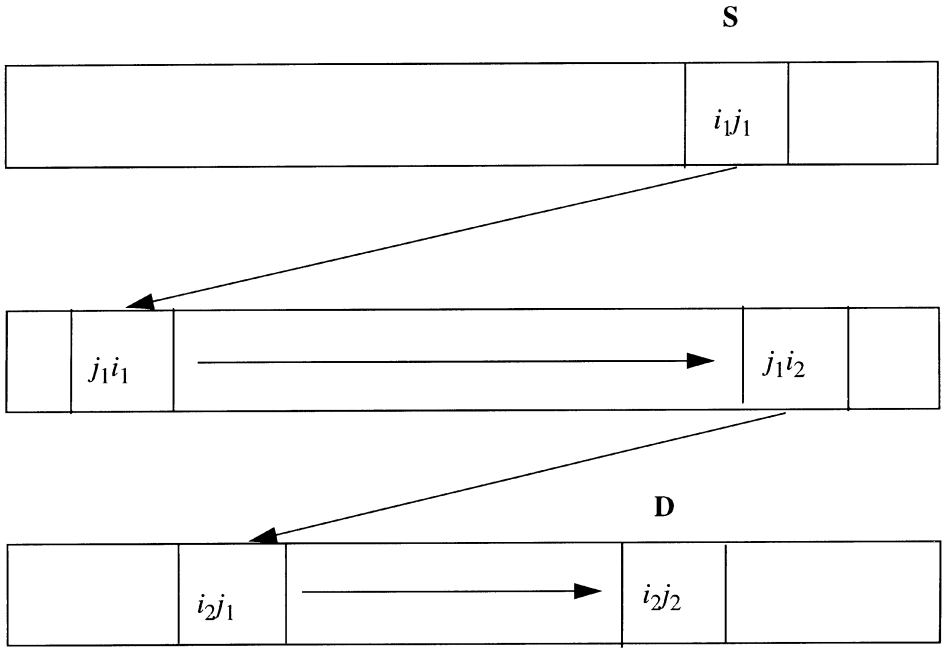

FIG. 8. Algorithm 2 routing strategy from source PE, **S**, to destination PE, **D**.

the first phase implements the routing using Algorithm 1, and the third phase implements the routing using Algorithm 2. An intermediate second phase is used to acknowledge receipt of a subset of all messages transmitted in the first phase. Before any permutation is routed, each message being routed is duplicated. Then we use Algorithm 1 to route one set of the messages. At the second step of Algorithm 1, where we have to perform a transpose operation which is the source of the congestion problem, some PEs would have a number of messages less than or equal to $N^{1/4}$ to transpose, and others would have a number of messages greater than $N^{1/4}$ to transpose. We allow these PEs to transmit at most $N^{1/4}$ messages, dropping all subsequent messages from the network. At the end of phase 1, some messages would have reached their destinations. Each PE receiving a message at the end of phase 1, would acknowledge that to the source PE. The acknowledgment is needed to prevent the source PEs, whose messages have successfully reached their final destination, from sending the duplicates in the final phase of Algorithm 3. Then, in the third phase, we use Algorithm 2 to route the second set of messages, which have not been acknowledged. Formally, Algorithm 3 is presented below. Given a message, m, and source PE i_1j_1 and destination PE i_2j_2, Algorithm 3 performs the following steps:

ALGORITHM 3.

Phase 1.

1. PE i_1j_1 sends m to PE i_1i_2.
2. PE i_1i_2 sends m to PE i_2i_1 if the number of received messages is $\leq N^{1/4}$; otherwise, it drops all subsequent messages from the network.
3. PE i_2i_1 sends m to PE i_2j_2.

Phase 2.

4. All destination PEs acknowledge the reception of the message to the source PEs.

Phase 3.

5. PE i_1j_1 sends m to PE j_1i_1.
6. PE j_1i_1 sends m to PE j_1i_2.
7. PE j_1i_2 sends m to PE i_2j_1.
8. PE i_2j_1 sends m to PE i_2j_2.

Each phase of Algorithm 3 is completely finished before progressing to the next phase. Note that PE i_1i_2 of step 2 can receive $>N^{1/4}$ messages for transpose link transmission only at level L; thus, recursive calls to Phase 1 of Algorithm 3 for lower levels are the same as recursive calls to Algorithm 1. The acknowledgments of Phase 2 imply the inverse use of level L transpose links guaranteeing that at most $N^{1/4}$ messages are routed over any level L transpose link when using Algorithm 1 for Phase 2, so we will use Algorithm 1 for Phase 2. Recursive calls in Phase 3 will use Algorithm 1.

We will now argue that Algorithm 3 needs $O(N^{1/4})$ routing steps in the worse case. The following lemma establishes the key result.

LEMMA 2. *Given $R(N, L)$ defined as the number of routing steps taken by Algorithm* 3 *for a level L RCC-FULL of size N; then $R(N, L) \leq 3N^{1/4} + 1 + 6 \times R(N^{1/2}, L-1)$ where $R(N, 0) = 1$.*

Proof. We want to find out what is the worst case routing time using Algorithm 3 to route a message m stored in PE i_1j_1 and destined to PE i_2j_2 under any message distribution. Using the first phase of Algorithm 3 the message m follows the path

$$i_1j_1 \rightarrow i_1i_2 \Rightarrow i_2i_1 \rightarrow i_2j_2.$$

The symbol $\rightarrow$ denotes routing within an $(N_A, L-1)$ RCC-FULL, and the symbol $\Rightarrow$ denotes routing along a level L *transpose* link. The only congestion problem for the message m in this routing phase occurs when PE i_1i_2 has x messages ($x > 1$) to transmit to PE i_2i_1 along the single transpose link between them. This means that there are x messages, including m, which originated in the same row, i_1, and are destined to the same row, i_2. If $x \leq N^{1/4}$, the x messages would be routed along that single *transpose* link and thus the message m originating at PE i_1j_1 would reach its destination PE i_2j_2 in at most $2 \times R(N^{1/2}, L-1) + N^{1/4}$ routing steps. Further, the source PE, i_1j_1, would be acknowledged of its message delivery so that it will not be transmitted again in Phase 3 of Algorithm 3. The acknowledgment process of Phase 2 takes at most $2 \times R(N^{1/2}, L-1) + N^{1/4}$ routing steps since the level L *transpose* links are used with the same distribution as in Phase 1 but in the opposite direction. Thus, the whole process of routing message m in this case takes at most $4 \times R(N^{1/2}, L-1) + 2 \times N^{1/4}$ routing steps.

In the case where $x > N^{1/4}$, some messages including say m have been dropped out of the network. Thus, PE i_1i_2 sends m again to PE i_2i_2 using Phase 3 of Algorithm 3. In this case, m follows the path

$$i_1j_1 \Rightarrow j_1i_1 \rightarrow j_1i_2 \Rightarrow i_2j_1 \rightarrow i_2j_2.$$

The only possible congestion at level L for message m is along the second *transpose* link. In this case, PE j_1i_2 has y messages to transmit to PE i_2j_1 along the single *transpose* link between them. This means that there are y messages, including m, which originated in the same column, j_1 (but different rows), and are destined to the same row, i_2. Denote these messages $m_1, m_2, m_3, \ldots, m_y$. Each congested message, m_i, must have had a congestion with $z_i - 1$ other messages during Phase 1 of Algorithm 3, where $z_i > N^{1/4}$. Otherwise, they would have reached their destination using Phase 1, and would not be routed using Phase 3. Note each of the $z_i - 1$ messages was destined to the same row as m_i and $m_1, m_2, m_3, \ldots, m_y$ are destined to the same row. Thus,

$$\sum_{i=1}^{y} z_i \leq N^{1/2}.$$

Since each $z_i > N^{1/4}$, $y < N^{1/4}$ (e.g., in the worse case $y = N^{1/4} - 1$). Therefore, our original message, m, would

have a maximum congestion of $y < N^{1/4}$ in Phase 3 of Algorithm 3. Hence, we can route our original message, m, in at most $2 \times R(N^{1/2}, L-1) + N^{1/4} + 1$ routing steps. Hence, Algorithm 3 takes at most a total of $6 \times R(N^{1/2}, L-1) + 3 \times N^{1/4} + 1$ routing steps. Q.E.D.

The worse case routing time for Algorithm 1 on an $(N^{1/2}, L-1)$ RCC-FULL, $W(N^{1/2}, L-1)$, satisfies

$$W(N^{1/2}, L-1) \leq N^{1/4} + 2 \times W(N^{1/4}, L-2),$$

since at most $N^{1/4}$ messages could be routed along any level $L-1$ *transpose* link. Thus

$$\begin{aligned} W(N^{1/2}, L-1) &\leq N^{1/4} + N^{1/8} + N^{1/16} + \ldots \\ &= O(N^{1/4}). \end{aligned}$$

Since in Algorithm 3 we revert to Algorithm 1 for recursive calls on level $L-1$ or lower, $R(N^{1/2}, L-1) = W(N^{1/2}, L-1)$ and the main theorem follows.

THEOREM 3. *Algorithm* 3 *requires at most* $O(N^{1/4})$ *routing steps on an* (N_A, L) *RCC-FULL.*

3. EMULATION OF THE PRAM ON RCC-FULL

Many parallel algorithms in the literature are designed to run on a PRAM. PRAMs are very convenient for expressing parallel algorithms since one may concentrate on decomposing the problem at hand into simultaneously executable tasks, without having to worry about the communication between these tasks. Further, it is quite easy to design parallel programming languages for such a model [11]. For this reason, the implementation of certain data movement operations that will enable realistic parallel architectures to emulate a PRAM has been considered by many researchers in order to take advantage of all the parallel algorithms that have been written for the PRAM and to enable them to write PRAM algorithms directly on their networks. These data movement operations are random access read (RAR) and random access write (RAW), also known as concurrent read and concurrent write, respectively [25, 26, 29]. They are used to allow a given parallel architecture to emulate the concurrent read and the concurrent write capabilities of a CRCW PRAM. These two data movement operations are implemented using well-defined routines. We will analyze the time complexity of each of these routines on the RCC-FULL to find the time complexity of RAR and RAW operations when performed on the RCC-FULL. These routines are sorting, compression, ranking, distribution, and generalization [26, 29]. In asymptotic analysis we assume that the RCC-FULL level, L, is fixed and the network size N grows large by increasing the atom size, N_A.

3.1. Sorting on RCC-FULL

Now we will identify the time complexity of sorting on the RCC-FULL. Again, an RCC-FULL of size N can be thought of as a network containing $N^{1/2}$ rows of PEs and $N^{1/2}$ columns of PEs and the rows are fully interconnected together. The sorting algorithm is defined as follows: a collection of N elements are distributed in the RCC-FULL, one element per processor; then viewing the input as an $N^{1/2} \times N^{1/2}$ array, the array is sorted into row-major order. The following sorting algorithm is based on the sorting algorithm given by Marberg and Gafini [22], and works by alternately transforming the rows and columns of the RCC-FULL a constant number of times. The details of the sorting algorithm on the RCC-FULL, denoted *RCC-FULL SORT*, are given below:

ALGORITHM RCC-FULL SORT.

1. Sort all the columns downward.
2. Sort all the rows to the right.
3. Rotate each row, i, $i \times N^{1/4} \pmod{N^{1/2}}$ positions to the right.
4. Sort all the columns downward.
5. Rotate each row, i, $i \pmod{N^{1/2}}$ positions to the right.
6. Sort all the columns downward.
7. Rotate each row, i, $i \times N^{1/4} \pmod{N^{1/2}}$ positions to the right.
8. Sort all the columns downward.
9. Perform the following two steps three times:
   a. Sort all the even-numbered rows to the right and all odd-numbered rows to the left.
   b. Sort all the columns downward.
10. Sort all the rows to the right.

A step-by-step application of *RCC-FULL SORT* is shown in Fig. 9.

Since rotation of elements within a row can be emulated by sorting along that row, all the steps of *RCC-FULL SORT* can be implemented by using sorting in a row or column in an RCC-FULL. For a $(N_A, 1)$ RCC-FULL, each row is a fully connected network; thus sorting the rows of a $(N_A, 1)$ RCC-FULL takes $O(\log(N))$ time, since sorting N elements on a fully connected network of size N takes $O(\log(N))$ time [4, 7]. Sorting on the columns of a $(N_A, 1)$ RCC-FULL can be performed on the RCC-FULL rows after performing a network transposition, with one parallel exchange operation. One final transposition returns all data to their desired destinations. Hence, sorting the columns of a $(N_A, 1)$ RCC-FULL takes $O(\log(N))$ time, and the whole sorting algorithm can be performed on a $(N_A, 1)$ RCC-FULL in $O(\log(N))$ time. For an $(N_A, 2)$ RCC-FULL, each sorting step of *RCC-FULL SORT* would be sorting on a $(N_A, 1)$ RCC-FULL, which each takes $O(\log(N))$ time as shown above. Consequently, sorting on a $(N_A, 2)$ RCC-FULL takes $O(\log(N))$ time. In general, for a (N_A, L) RCC-FULL, the sorting time, $ST(L)$, is given by

$$ST(L) = K_1 ST(L-1) + K_2, \tag{15}$$

where K_1 = 15 and K_2 = 14, as found from *RCC-FULL SORT*. Since $ST(0) = K_3 \log(N)$ with K_3 constant [4, 7], then by

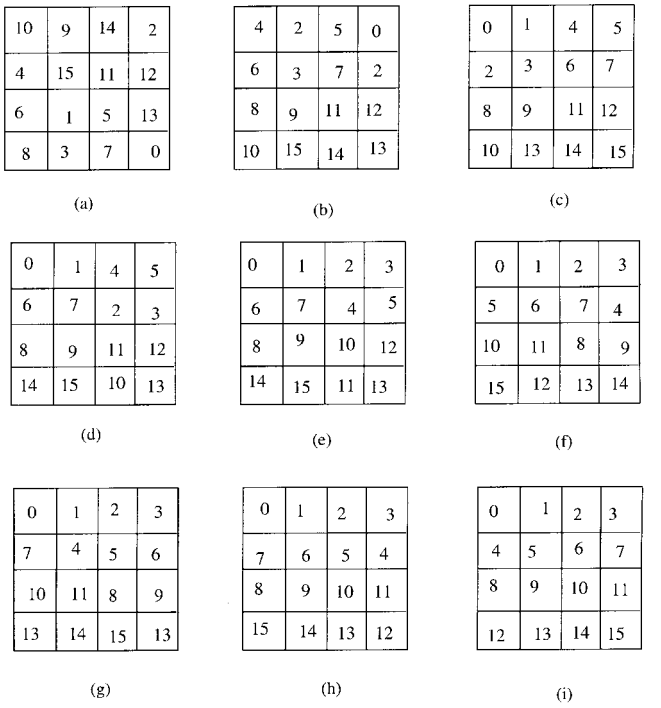

FIG. 9. RCC-FULL SORT on a 4×4 array. (a) Initial data configuration. (b) After step 1 of RCC-FULL SORT. (c) After step 2 of RCC-FULL SORT. (d) After step 3 of RCC-FULL SORT. (e) After step 4 of RCC-FULL SORT. (f) After steps 5 and 6 of RCC-FULL SORT. (g) After steps 7 and 8 of RCC-FULL SORT. (h) After step 9 of RCC-FULL SORT. (i) After step 10 of RCC-FULL SORT.

solving the recursion we get

$$ST(L) = K_1^L K_3 \log\ N + \frac{1 - K_1^L}{(1 - K_1)} k_2. \qquad (16)$$

Thus if L is held constant, the time complexity of *RCC-FULL SORT* is $O(\log(N))$ when implemented on the RCC-FULL.

3.2. RAR and RAW Time Complexity

Here we develop the time complexity of compression, ranking, distribution, and generalization [26, 29] which when added to the time complexity of *RCC-FULL SORT* would give us the time complexity of RAR and RAW on the RCC-FULL. The compression, ranking, distribution, and generalization routines are all instances of the *ascend* class of algorithms [28]. An algorithm is said to be in the *ascend* class if it performs a sequence of operations on pairs of data that are successively $2^0, 2^1, \ldots, 2^{k-1}$ locations apart on a problem of size 2^k [28].

An algorithm of size $N = 2^k$ which is in the *ascend* class can be performed on an RCC-FULL in the following manner:

1. Perform operations on pairs of data that are successively $2^0, 2^1, \ldots, 2^{k/2-1}$ locations apart.
2. Exchange all the data in the PEs which are directly connected through a transpose link.
3. Perform operations on pairs of data that are successively $2^0, 2^1, \ldots, 2^{k/2-1}$ locations apart.
4. Exchange all the data in the PEs which are directly connected through a transpose link.

When $L = 0$, it takes $\log(N)$ steps to perform the above algorithm on an RCC-FULL since all the PEs are directly connected together [28]. For a $(N_A, 1)$ RCC-FULL steps 1 and 3 are algorithms of size $2^{k/2}$ in the *ascend* class performed on a fully connected network of size $2^{k/2}$. In the general case, to perform the above algorithm on a (N_A, L) RCC-FULL, steps 1 and 3 would be the execution of an algorithm of size $2^{k/2}$ in the *ascend* class on a $(N_A, L-1)$ RCC-FULL. Thus if we denote $AS(L)$ to be the number of communication steps taken on a (N_A, L) RCC-FULL to execute an algorithm in the *ascend* class, we get

$$AS(L) = 2AS(L-1) + 2, \qquad (17)$$

where $AS(0) = \log(N)$. Solving this recursion we get

$$AS(L) = 2^L(2 + \log\ N) - 2. \qquad (18)$$

Thus if L is held constant, the time complexity of an algorithm in the *ascend* class is $O(\log(N))$ when implemented on an RCC-FULL.

Now, we present a brief introduction of the following sub-algorithms which are used in the implementation of RAR and RAW:

3.2.1. Ranking

When some PEs are selected in a network, the rank of a PE is equal to the number of selected PEs with a smaller index. The number of communication steps needed by this algorithm is equal to the number of communication steps needed by an *ascend* algorithm which is equal to $\log(N)$ for an RCC-FULL. The number of computation steps is $2\log(N)$ on the RCC-FULL.

3.2.2. Compression

When the number of active PEs in a network is s, a proper subset of all the PEs in the network applying the compression algorithm on these active elements will move these active elements to the PEs indexed 0, 1, 2, ..., $s-1$. The number of communication steps needed by this algorithm is exactly equal to the number of communication steps needed by the *ascend* algorithm. It has no computation steps. Thus, when implemented on an RCC-FULL, it needs $\log(N)$ communication steps.

3.2.3. Distribution

Assume that some PEs m of a network each have a datum d_m and a PE destination h_m such that if $i < j$ then $h_i < h_j$. Executing the distribution algorithm on the network consists of routing, for each PE m, the datum d_m to the PE h_m.

The distribution algorithm is the inverse of the compression algorithm and requires the same number of communication steps.

3.2.4. Generalization

Given a set of PEs m of a network each has a datum d_m and a PE destination h_m such that if $i < j$ then $h_i < h_j$. The generalization algorithm consists of routing multiple copies of the datum d_m to the destination $h_{m-1} + 1$ through h_m. The number of communication steps is equal to that needed by the *ascend* algorithm which is $\log(N)$ steps for the RCC-FULL. The number of computation steps needed is $3\log(N)$.

Hence, compression, ranking, distribution, and generalization can each be executed on the RCC-FULL in $O(\log(N))$ time. A RAR is performed by executing the sorting subalgorithm twice, the ranking subalgorithm once, the compression subalgorithm twice, the distribution subalgorithm once, and the generalization subalgorithm once [26, 29]. Thus, to perform a RAR operation, an RCC-FULL requires $O(\log(N))$ time. A RAW operation is performed by executing the sorting subalgorithm once, the ranking subalgorithm once, the compression subalgorithm once, and the distribution subalgorithm once [26, 29]. Thus, to perform a RAW operation, an RCC-FULL requires $O(\log(N))$. Hence, an RCC-FULL of size N can emulate a CRCW PRAM of the same size with at most $O(\log(N))$ degradation in time performance when L is held constant. This also means that $O(\log(N))$ is an upper bound on the time needed for the RCC-FULL to emulate arbitrary interconnection networks of the same size. Thus, in some sense the RCC-FULL can be considered to be a universal network [19].

4. HARDWARE COST

In this section we investigate the amount of hardware needed by the RCC-FULL under certain packaging constraints for different network sizes. Then we compare this hardware complexity to that for the binary hypercube in order to assess the potential of the RCC-FULL as a practical parallel machine. Many parallel machines have been constructed using hypercube interconnections and are commercially available [31]. One useful measure of hardware cost is the area required when the entire parallel computer is laid out on a single sheet of silicon. This measure has been well-studied, and the very-large-scale integration (VLSI) area requirements of many interconnection networks are also known. However, actual parallel machines, especially for large network sizes, are typically laid out on a number of separate chips, each of which has a limited number of pins through which connections can be made to other chips. In most cases the number of pins available per chip is a more serious limitation than the amount of area available per chip. This is particularly true for networks that have a relatively large number of links per processor such as the hypercube and the RCC-FULL unlike systolic arrays or networks with very small degrees such as the binary tree and the 2D mesh [8, 12, 17, 23]. As a result, the pin requirements of a highly connected parallel computer are a very important measure of its hardware cost. This has motivated the analysis presented in this paper regarding the pin requirements of the RCC-FULL as compared to that of the hypercube.

A package is the entity that houses the implementation of a computer system. It comes with a variety of forms such as the integrated circuit chip, a printed circuit board that houses many of these chips, and a chassis that houses multiple circuit boards. These packages have a hierarchical relationship; the low-level packages are housed by a higher level package. All levels of packages, however, share a common characteristic. Each package contains PEs and communication links. Some of these communication links connect PEs within the packages, and others connect PEs in different packages. The latter communication links constitute the pin requirements of the package. As argued before, in our analysis of the hardware cost of the RCC-FULL and the hypercube, we will concentrate on the pin requirements of the packages that are needed to build them. In other words, the VLSI area requirements of the packages are assumed to be less severe than their pin requirements.

Let N be the number of network processors, and MCB be the number of chips to which the network is partitioned. The goal is to find the minimum number of pins per chip, $MIOC$, over all partitions. Hence, MCB should be greater than or equal to 2 to avoid the trivial solution of having all processors contained in the same chip, and therefore, $MIOC$ = 0. For a hypercube network, we assume that it is partitioned into smaller dimensional hypercubes with MPC processors each, that is, $MPC = N/MCB$. Each processor will then be connected to $O(\log(N) - \log(N/MCB)) = O(\log(MCB))$ processors in different chips. Thus, the total number of pins per chip is: $MIOC = O(N/MCB(\log(MCB)))$. This result has been discovered by several researchers [8, 23].

THEOREM 4. *For a hypercube network with N processors partitioned over MCB chips, the pin requirement per chip, $MIOC$, is given by $MIOC = O(N/MCB(\log(MCB)))$.*

The same analysis carried on the chip package level can be carried on the next higher level, the printed circuit board level package. In this case, we partition the hypercube into smaller dimensional hypercubes of size MPB processors each, and we let MBC denote the number of partitions (e.g., boards). Then, each board will have N/MBC processors. Each processor will be connected to $O((\log(N) - \log(N/MBC))) = O(\log(MBC))$ processors in different boards. Thus, the total number of I/O ports per board is $MIOB = O(N/MBC(\log(MBC)))$. However, we have to note that the number of processors per board, MPB, is directly dependent on the number of processors per chip, MPC. Hence, if for a specific network, the number of processors per chip is small because of the pin limitations, then that will directly affect the number of processors per board. Therefore,

that would increase the number of boards needed to build the network, which in turn would increase its hardware cost. Further, since a board will contain a hypercube of smaller dimension where the processors would be connected to other processors in other boards, the pin limitations at the board level packaging would be even more severe than that at the chip level packaging. Thus, if the partitioning of two given networks, N_1 and N_2, at the chip level and the circuit board level is made under the same assumptions (e.g., smaller dimensional hypercube), then if N_1 has more pin requirements at the chip level than N_2, N_1 will definitely have more pin requirements at the board level than N_2. Consequently, just analyzing the pin requirements of two networks at the chip level would give us a clear indication of their hardware cost. For this reason, we will compare the pin requirements of the hypercube and the RCC-FULL at the chip level only.

Now, let us determine the pin requirements of an RCC-FULL of size N under the chip level packaging. We carry our analysis under the following two situations:

Case 1. When the number of RCC-FULL partitions (chips), MCB, is less than or equal to $N^{1/2}$. In this case, we assume that the partitioning is being performed horizontally as shown in Fig. 10. That is, each chip will contain an integral number of $(N_A, L-1)$ RCC-FULLs. This partition scheme would minimize the number of pins per chip, $MIOC$. Otherwise, $MIOC$ would include additional links through the partitioning of $(N_A, L-1)$ RCC-FULLs. Similar assumptions have been made for the same analysis of several interconnection networks [8, 12, 17, 23].

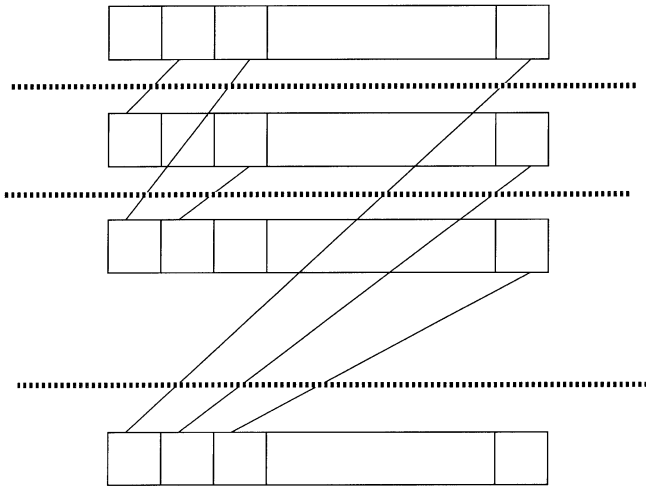

FIG. 10. Partitioning of an RCC-FULL where $MCB = N^{1/2}$.

In this case only *transpose* links would be needed to connect processors in different chips. Thus, the total number of pins needed would be exactly equivalent to partitioning a fully connected network, FCN, of size $N^{1/2}$ since we have $N^{1/2}$copies of $(N_A, L-1)$ RCC-FULLs. The number of links in each processor of an $N^{1/2}$ FCN is $N^{1/2}-1$. Again, we let MCB denote the total number of chips. Each processor will be connected to $O(N^{1/2}-1-(N^{1/2}/MCB-1)) = O(N^{1/2}-N^{1/2}/MCB)$ processors on different chips. Thus, the pin requirement per chip is $MIOC = O(N^{1/2}/MCB(N^{1/2}-N^{1/2}/MCB)) = O(N/MCB - N/MCB^2)$.

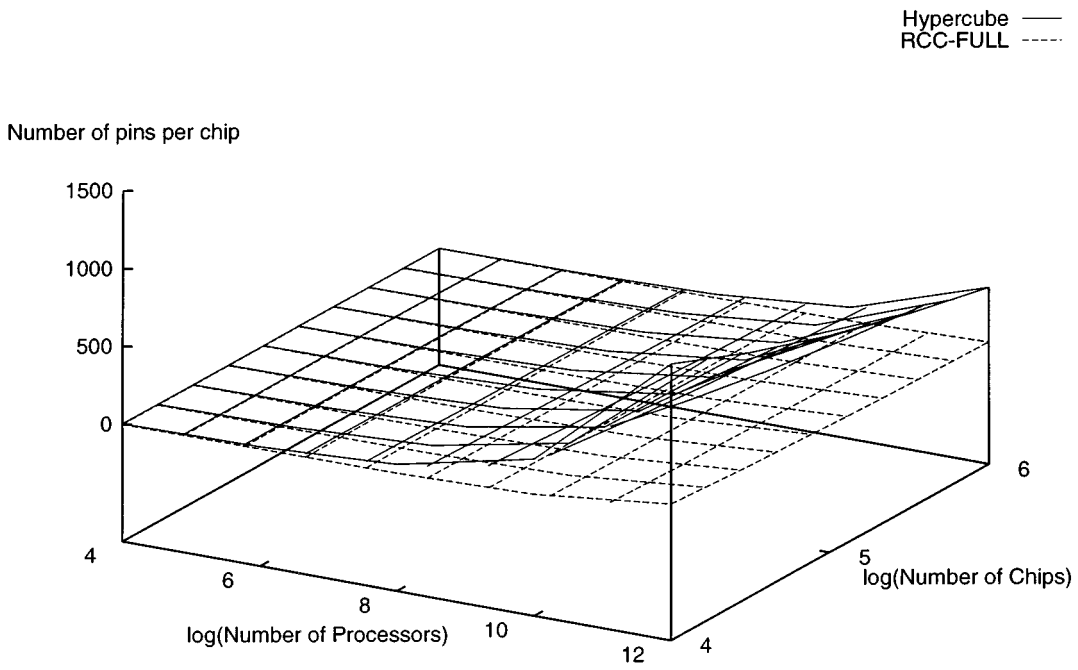

FIG. 11. Pin requirements at the chip level for an RCC-FULL and a hypercube in Case 1.

THEOREM 5. *For an (N_A, L) RCC-FULL network with N processors partitioned over MCB chips, the pin requirement per chip, $MIOC$, is given by $MIOC = O(N/MCB - N/MCB2)$, where MCB is less than or equal to $N^{1/2}$.*

Thus, when MCB is less than or equal to $N^{1/2}$, the pin requirement of the RCC-FULL is asymptotically less than that of the hypercube for the same network sizes. Moreover, the pin requirement of the RCC-FULL, in this case, is independent of the level of recursion, L; they are all equal. Figure 11 compares the pin requirements at the chip level, $MIOC$, of a hypercube network and an RCC-FULL as a function of the number of processors, N, and the number of chips, MCB. As depicted in the figure, the chip pin requirement of a hypercube is higher than that of an RCC-FULL. Thus, the hardware cost of a hypercube would be higher than that of an RCC-FULL of the same size. Moreover, since the number of pins per chip cannot be arbitrarily high (e.g., 1000), many of the hypercube chip requirements are technologically infeasible (e.g., when N is high and MCB is low).

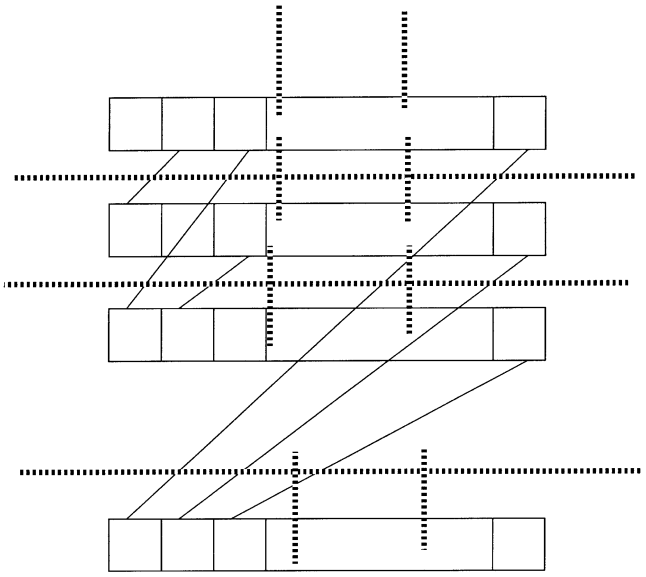

FIG. 12. Partitioning of an RCC-FULL where $MCB = I \times N^{1/2}$.

Case 2. We assume that the number of partitions (chips) of an RCC-FULL is $MCB = I \times N^{1/2}$, where I is an integer greater or equal to 2. That is, each $(N_A, L-1)$ RCC-FULL is partitioned over exactly I chips. An example of this partitioning is illustrated in Fig. 12.

Thus in this case the number of pins per chips, $MIOC$, would correspond to *transpose* links *and* to *internal* links of an $(N_A, L-1)$ RCC-FULL. The number of pins that correspond to *transpose* links are $O(N/MCB_1 - N/MCB_1^2)$, where $MCB_1 = N^{1/2}$. Therefore, the pins that correspond to *transpose* links, $MIOC_T$, is $O(N^{1/2}-1)$ for each $(N_A, L-1)$ RCC-FULL (e.g., each processor except one has a single *transpose* link). Moreover, the total number of pins that correspond to *transpose* links is independent of the level of recursion, L, of an RCC-FULL.

For the number of pins that correspond to *internal* links, we have two cases:

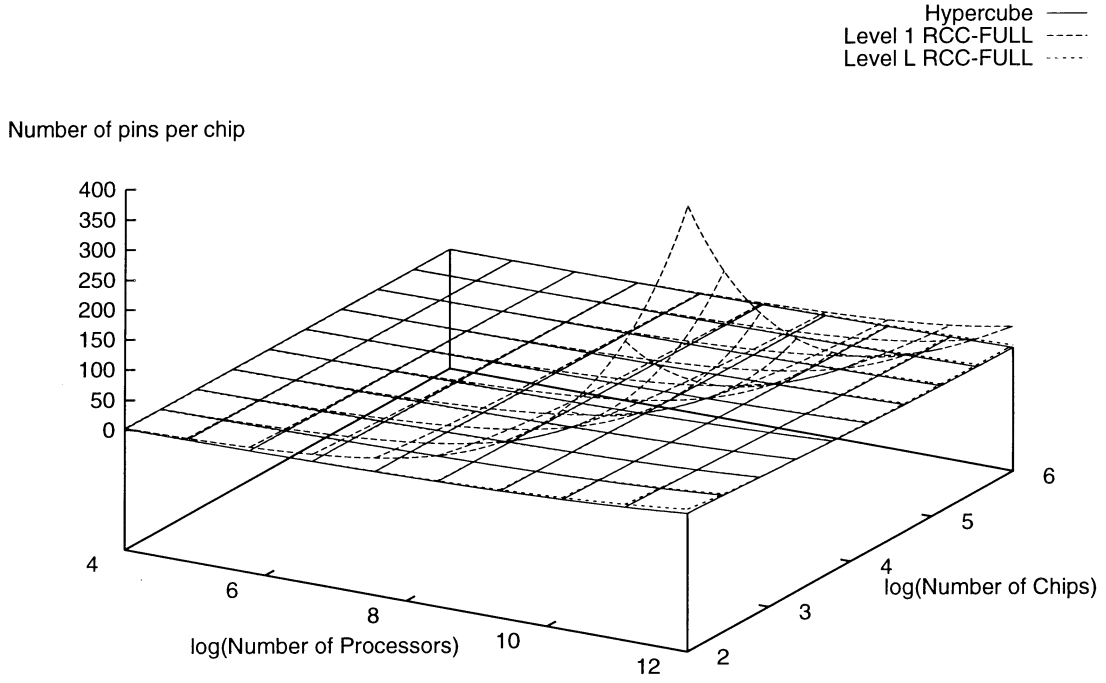

FIG. 13. Pin requirements at the chip level for an RCC-FULL and a hypercube in Case 2.

A. When $L = 1$, meaning that the rows of an RCC-FULL are a fully connected network, then the number of pins per chip that correspond to *internal* links is $O(N/I - N/I^2)$.

B. When $L > 1$, then the number of pins per chip that correspond to internal links is $O(N^{1/2}/I - N^{1/2}/I^2)$. In this case, since I is less than or equal to $N^{1/2}$, it would correspond to Case 1 above of the RCC-FULL. This leads to the following theorem.

THEOREM 6. *For an (N_A, L) RCC-FULL network with N processors partitioned over MCB chips, and $MCB = I \times N^{1/2}$, the pin requirement per chip, $MIOC$, is given by $MIOC = O(N/I - N/I^2 + N^{1/2}/I - 1)$ when $L = 1$, is given by $MIOC = O(N^{1/2}/I - N^{1/2}/I^2 + N^{1/2}/I - 1)$ when $L > 1$. The first 2 terms correspond to pins due to internal links, and the last 2 terms correspond to pins due to transpose links.*

Thus, when the number of chips used in the partitioning of the network is greater than $N^{1/2}$, the pin requirement at the chip level for a $(N_A, 1)$ RCC-FULL is asymptotically equivalent to that of a hypercube of the same size. On the other hand, when the level of recursion of the RCC-FULL, L, is greater than 1, the pin requirement at the chip level of the RCC-FULL is asymptotically less than those of a hypercube of the same size. Figure 13 compares the pin requirements at the chip level, $MIOC$, of a hypercube network and an RCC-FULL as a function of the number of processors, N, and the number of chips.

5. CONCLUSION

We have presented a new interconnection network, RCC-FULL, for the construction of large scale parallel supercomputing systems. Its systematic construction and some of its key communication properties have been shown and compared favorably to those of the hypercube. Further, the RCC-FULL has more flexibility in choosing the desired performance level through its level of recursion, L, unlike the hypercube. Three efficient routing algorithms have been derived for the RCC-FULL which need only local information to route messages between any two nodes in the network. A sorting algorithm is shown for RCC-FULL which has $O(\log(N))$ complexity and the RCC-FULL has been shown to emulate deterministically the CRCW PRAM model with only $O(\log(N))$ degradation in time performance. The hardware cost of the RCC-FULL as a function of its pin limitations has been estimated and compared to that of the hypercube and most instances of RCC-FULL have substantially lower cost. RCC-FULL appears to offer particularly good potential as an interconnection network for systems which emulate the PRAM models and which can be considered as *universal* networks for their ability to emulate any other interconnection network of the same size with small degradation in time performance.

ACKNOWLEDGMENT

The authors thank the anonymous referees, whose valuable comments improved the quality of the paper.

REFERENCES

1. Agrawal, D. P., Janakiram, V. K., and Pathak, G. C. Evaluating the performance of multicomputer configurations. *IEEE Comput.* **9,** 5 (May 1986), 23–37.
2. Akl, S. G. *The Design and Analysis of Parallel Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1989.
3. Akl, S. G., and Lyons, K. A. *Parallel Computational Geometry.* Prentice Hall, Englewood Cliffs, NJ, 1993.
4. Alt, H., Hagerup, T., Mehlhorn, K., and Preparata, F. Deterministic simulation of idealized parallel computers on more realistic ones. *SIAM J. Comput.* **16** (Oct. 1987), 808–835.
5. Bhuyan, L. M., and Agrawal, D. P. Generalized hypercube and hyperbus structures for a computer network. *IEEE Trans. Comput.* **33,** 4 (April 1984), 323–333.
6. Borodin, A., and Hopcroft, J. Routing, merging, and sorting in parallel models of computation. *Proc. ACM Symposium Theory Comput.* 1982, pp. 338–344.
7. Cole, R. Parallel merge sort. *SIAM J. Comput.* **17** (1988), 770–785.
8. Cypher, R. Theoretical aspects of VLSI pin limitations. *SIAM J. Comput.* **22** (1993), 356–378.
9. Dandamudi, S. P., and Eager, D. L. Hierarchical interconnection networks for multicomputer systems. *IEEE Trans. Comput.* **39,** 6 (June 1990), 786–797.
10. El-Amawy, A., and Latifi, S. Properties and performance of folded hypercubes. *IEEE Trans. Parallel Distrib. Systems* **2,** 1 (Jan. 1991), 31–42.
11. Fortune, S., and Wyllie, J. Parallelism in random access machines. *Proc. ACM Symposium Theory Comput.* (1978), pp. 114–118.
12. Ghosh, J., and Hwang, K. Mapping neural networks onto message-passing multicomputers. *J. Parallel Distrib. Comput.* **6,** 2 (April 1989), 291–330.
13. Hamdi, M., and Hall, R. W. An efficient class of interconnection networks for parallel computations. *Comput. J.* **37,** 3 (1994), 206–218.
14. Hamdi, M., and Hall, R. W. Compound graph networks for parallel image processing. *Proc. of the 1991 Workshop on Comput. Arch. for Machine Perception.* 1991, pp. 365–377.
15. Kumar, J. M., and Patnaik, L. M. Extended hypercube: A hierarchical interconnection network of hypercube. *IEEE Trans. Parallel Distrib. Systems* **3,** 1 (Jan. 1992), 31–42.
16. Kruskal, C. P., Rudolph, L., and Snir, M. A complexity theory of efficient parallel algorithms. *Theoret. Comput. Sci.* **71,** 1 (Mar. 1990), 95–132.
17. Leighton, T. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, CA, (1993).
18. Leiserson, C. E., and Maggs, B. M. Communication-efficient parallel algorithms for distributed random-access machines. *Algorithmica* (1988), 53–77.
19. Leiserson, C. E. Fat-Trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.* **34** (1985), 892–900.
20. Luccio, F., Pietracaprina, A., and Pucci, G. A new scheme for the deterministic simulation of PRAMs in VLSI. *Algorithmica* **5** 4 (1990), 529–544.
21. Luccio, F., Pietracaprina, A., and Pucci, G. An efficient probabilistic simulation of PRAMs in VLSI. *Inform. Process. Lett.* (1988), 141–146.

22. Marberg, J. M., and Gafni, E. Sorting in constant number of row and column phases in a mesh. *Algorithmica* **3,** 4 (1988), 561–572.
23. Maresca, M., and Li, H. Polymorphic-torus networks. *IEEE Trans. Comput.* **38,** 9 (Sep. 1989), 1345–1351.
24. Mehlhorn, K., and Vishkin, U. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informat.* (1984), 339–374.
25. Miller, R., and Stout, Q. F. Parallel Algorithms for Regular Architectures. MIT Press, Cambridge, MA, 1994.
26. Nassimi, D., and Sahni, S. Data broadcasting in SIMD computers. *IEEE Trans. Comput.* **30,** (1981), 101–107.
27. Olariu, S., Schwing, J. L., and Zhang, J. On the power of two-dimensional processor arrays with reconfigurable bus systems. *Parallel Process. Lett.* **1,** 1 (Sep. 1991), 29–34.
28. Preparata, F. P., Vuillemin, J. The cube-connected cycles: A versatile network for parallel computations. *Comm. ACM* **24** (1981), 300–309.
29. Ranka, S., and Sahni, S. *Hypercube Algorithms With Applications to Image Processing and Pattern Recognition.* Springer-Verlag, New York, 1990.
30. Reed, D. A., and Fujimoto, R. M. *Multicomputer Networks: Message-Based Parallel Processing.* MIT Press, Cambridge, MA, 1987.
31. Trew, A. and Wilson, G. Eds. *Past, Present, Parallel: A Survey of Available Parallel Computer Systems.* Springer-Verlag, Berlin/New York, 1991.
32. Upfal, E., and Wigderson, A. How to share memory in a distributed system. *J. Assoc. Comput. Mach.* **34** (1987), 116–127.

Received April 30, 1991; revised July 1, 1994; accepted August 23, 1996

MOUNIR HAMDI received the B.Sc. with distinction in electrical engineering from the University of Southwestern Louisiana in 1985, and the M.Sc. and Ph.D. in electrical engineering from the University of Pittsburgh in 1987 and 1991, respectively. While at the University of Pittsburgh, he was a research fellow involved with various research projects on interconnection networks, high-speed communication, parallel algorithms, switching theory, and computer vision. In 1991 he joined the Computer Science Department at Hong Kong University of Science and Technology as an assistant professor. His main areas of research are parallel computing, high-speed networks, and ATM packet switching architectures. Dr. Hamdi has published over 50 papers in these areas in various journals and conference proceedings. He co-founded and co-chairs the International Workshop on High-Speed Network Computing, and has been on the program committees of various international conferences. Dr. Hamdi is a member of IEEE and ACM.

RICHARD HALL has been a member of the faculty of the Department of Electrical Engineering at the University of Pittsburgh since 1975. His M.S. and Ph.D. were awarded from Northwestern University in 1971 and 1975, both in electrical engineering. He did his undergraduate work in electrical engineering at Johns Hopkins University. His early research was concerned with the modeling of mammalian vision. His current research is concerned with the study of parallel algorithms and architectures for computer vision, including the development of fast parallel connectivity preserving reduction and reduction–augmentation algorithms in 2D and 3D image spaces and applications of 3D mesh architectures.