EFFICIENT CRCW PRAM EMULATION ON PRACTICAL NETWORKS

Mounir Hamdi
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Email: hamdi@cs.ust.hk

Abstract

A new interconnection network is proposed for the construction of massively parallel computers. The systematic construction of this network, denoted RCN-FULL, is performed by methodically connecting together a number of basic atoms where a basic atom is a set of fully connected nodes. Key communication characteristics and efficient routing algorithms are derived for RCN-FULL. An O(log(N)) sorting algorithm is shown for RCN-FULL and RCN-FULL is proven to deterministically emulate the CRCW PRAM model, with only O(log(N)) degradation in time performance. Finally, the hardware cost for the RCN-FULL is estimated as a function of its pin limitations and compared favorably to that of the hypercube.

1) INTRODUCTION

One of the main challenges involved in designing a parallel algorithm for a parallel processing network follows from the fact that the routing of messages from one processing element (PE) to another is the responsibility of the algorithm designer. This challenge is removed completely by using a parallel random access machine (PRAM) [1, 3]. In PRAM, the PEs no longer communicate directly through a network. Instead, a common memory is used as a bulletin board and all data exchanges are executed through it. Any pair of PEs can communicate through this shared memory in constant time. Various types of PRAMs have been defined, differing in the conventions used to deal with read/write conflicts. In the most restrictive model, EREW PRAMs, no variable may be accessed by more than one PE in a given step. In contrast, CRCW PRAMs allow simultaneous reading and writing of each variable. Unfortunately, the PRAM is not a very realistic model of parallel computation when the number of PEs grows large.

How can we reconcile the convenience of CRCW PRAMs with the limitations of a real parallel computer? The only alternative is to emulate a CRCW PRAM on a real network. Indeed, this school of thought has led many researchers to consider the emulation of the PRAM on more realistic networks such as the hypercube, the mesh-of-trees, and the 2-D mesh [1, 5, 7]. However, to the best of our knowledge, no practical network has been shown to *deterministically* emulate any of the PRAM models of the *same* size in better than polylogarithmic degradation in time performance. Thus, in this paper, we investigate a class of Recursively Connected Networks (RCN), which is constructed by compounding FULLy connected graphs together, termed RCN-FULL; and we find that this class is able to emulate any PRAM model with better than polylogarithmic degradation in time performance. Hence, the design of the RCN-FULL appears to be a step closer towards the realization of a practical PRAM.

The paper is organized as follows. In Section 2 we define the RCN-FULL and we present some of its key characteristics. In Section 3, we demonstrate the efficient emulation of the PRAM models on the RCN-FULL. Finally, in Section 4 we analyze the hardware cost of the RCN-FULL and compare it to that of the hypercube.

2) THE RCN-FULL NETWORK

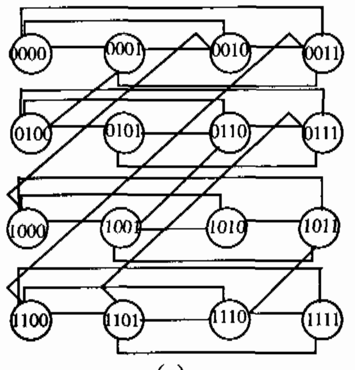
In this section we define the construction of the RCN-FULL, and we analyze some of its communication characteristics. Then we present simple routing algorithms.

2.1) Construction

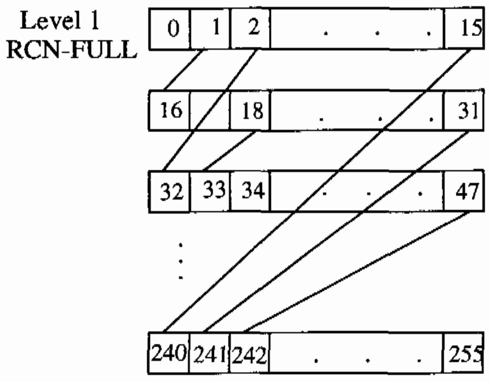
The proposed interconnection network, RCN-FULL, is a recursive network constructed by connecting together a number of basic atoms. A basic atom is a set of fully connected nodes. An RCN-FULL is characterized by two parameters, (N_A, L), where N_A is the number of nodes in the basic atom and L is its level of recursion. An $(N_A, 0)$ RCN-FULL is a fully connected network with N_A nodes. An $(N_A, 1)$ RCN-FULL is constructed by fully connecting N_A basic atoms creating a fully network of basic atoms. In general, an (N_A, L) RCN-FULL of size N is constructed by fully connecting $N^{1/2}$ $(N_A, L\text{-}1)$ RCN-FULLs where $N^{1/2}$ is the number of nodes in an $(N_A, L\text{-}1)$ RCN-FULL. Each node in an (N_A, L) RCN-FULL is specified by an m-bit binary number where $m = \log(N)$. The most significant $(1/2)\log(N)$ bits identify the $(N_A, L\text{-}1)$ RCN-FULL that this node belongs to, and the least significant $(1/2)\log(N)$ bits are used to distinguish among nodes within the same $(N_A, L\text{-}1)$ RCN-FULL. The links between these $(N_A, L\text{-}1)$ RCN-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 $(1/2)\log(N)$ bits each. Fig. 1 illustrates a (4, 2) RCN-FULL. .

2.2) Properties

The number of nodes, $N(N_A, L)$, of an (N_A, L) RCN-FULL is given by $N(N_A, L) = N^2(N_A, L\text{-}1)$ where $N(N_A, 0)$



(a)



(b)

Fig. 1. Construction of an RCN-FULL (a) Level 1 RCN-FULL. (b) Level 2 RCN-FULL.

$= N_A$; thus

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

Denoting the diameter of an (N_A, L) RCN-FULL by $D(N_A, L)$, from the construction of the RCN-FULL we see that $D(N_A, L) = 2D(N_A, L\text{-}1) + 1$, where $D(N_A, 0) = 1$; thus

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

Hence if the level of recursion, L, is held constant, the diameter of RCN-FULL is $O(1)$. In this case the network grows large by increasing N_A.

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

$$\Delta(N_A, L) = N_A + L - 1. \qquad (3)$$

The diameter and the degree of an RCN-FULL compare favorably to those of the hypercube for most practical size networks. Further, some characteristics of the static and dynamic behavior of RCN-FULL (e.g. message delay, throughput, fault-tolerance, etc.) has been studied and compared favorably to those of the hypercube [4, 5].

2.3) Routing

We propose two routing algorithms for the RCN-FULL which require only the source and the destination addresses to perform the routing of messages at any node in the network. An (N_A, L) RCN-FULL with N nodes can be thought of as containing $N^{1/2}$ rows of PEs, each being an $(N_A, L\text{-}1)$ RCN-FULL, and $N^{1/2}$ columns of PEs and the rows are fully connected together. In both routing algorithms we describe below, the source node is PE i_1j_1, and the destination node is PE i_2j_2 where i_1 and i_2 indicate the row addresses and j_1 and j_2 indicate column addresses.

***Algorithm 1*:**

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

1) PE i_1j_1 sends m to PE i_1i_2.
2) PE i_1i_2 sends m to PE i_2i_1.
3) PE i_2i_1 sends m to PE i_2j_2.

Step 2 is one routing step along a transpose link for all levels of the RCN-FULL. In general, for an (N_A, L) RCN-FULL, steps 1 and 3 are $(N_A, L\text{-}1)$ RCN-FULL routing.

***Algorithm 2*:**

The second routing algorithm has been identified to solve the congestion problem that could occur using *Algorithm 1* when there is a high transfer of data between two rows. *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.

For an (N_A, L) RCN-FULL, steps 2 and 4 are routing within an $(N_A, L\text{-}1)$ RCN-FULL, and steps 1 and 3 are routing along level L transpose links.

3) EMULATION OF PRAM ON RCN-FULL

The implementation of data movement operations that enable realistic networks to emulate a CRCW PRAM has been considered by many researchers. These data movement operations are random access read (RAR) and random access write (RAW), also known as concurrent read and concurrent write, respectively [7, 8]. They are implemented using well-defined routines. We will analyze the time complexity of each of these routines on the RCN-FULL to find the time complexity of RAR and RAW when performed on the RCN-FULL. These routines are sorting, compression, ranking, distribution, and generalization.

3.1) Sorting on RCN-FULL

The sorting algorithm is defined as follows: a collection of N elements are distributed in the RCN-FULL, one element per PE; 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 [6], and works by alternately transforming the rows and columns of the RCN-FULL a constant number of times. It perfectly suits the structure and the *transpose* capability of the RCN-FULL topology, and is given below:

Algorithm RCN-FULL SORT

1. Sort all the columns downward.
2. Sort all the rows to the right.

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

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

$$ST(L) = K_1 ST(L-1) + k_2 \qquad (4)$$

where $K_1 = 15$ and $K_2 = 14$ as found from *RCN-FULL SORT*. Since $ST(0) = K_3\log(N)$ with K_3 constant [1, 2], then by solving the recursion we get

$$ST(L) = K_1^L K_3 (\log N) + \frac{1-K_1^L}{1-K_1} K_2 \qquad (5)$$

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

3.2) RAR and RAW time complexity

Here we develop the time complexity of compression, ranking, distribution, and generalization [8] which when added to the time complexity of *RCN-FULL SORT* would give us the time complexity of RAR and RAW on the RCN-FULL. These routines are all instances of the *ascend* class of algorithms [5]. 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, ..., 2^{k-1} locations apart on a problem of size 2^k [5].

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

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

When $L = 0$, it takes $\log(N)$ time steps to perform the above algorithm on an RCN-FULL since all the PEs are directly connected together [5]. To perform the above algorithm on an (N_A, L) RCN-FULL, steps 1 and 3 would be the execution of an algorithm of size $2^{k/2}$ in the *ascend* class on an (N_A, L-1) RCN-FULL. Thus if we denote $AS(L)$ to be the number of communication steps taken on an (N_A, L) RCN-FULL to execute an algorithm in the *ascend* class, we get

$$AS(L) = 2AS(L-1) + 2, \qquad (6)$$

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

$$AS(L) = 2^L(\log(N) + 2) - 2. \qquad (7)$$

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

Hence, compression, ranking, distribution, and generalization can each be executed on the RCN-FULL in $O(\log(N))$ time [4, 5]. A RAR is performed by executing the sorting twice, the ranking once, the compression twice, the distribution once, and the generalization once [5]. Thus, to perform a RAR operation, an RCN-FULL requires $O(\log(N))$ time. A RAW operation is performed by executing the sorting once, the ranking once, the compression once, and the distribution once [5]. Thus, to perform a RAW operation, an RCN-FULL requires $O(\log(N))$ time. Hence, an RCN-FULL of size N can emulate a CRCW PRAM of the same size with at most $O(\log(N))$ degradation in time performance. This also means that $O(\log(N))$ is an upper bound on the time needed for the RCN-FULL to emulate arbitrary interconnection networks of the same size. Thus, in some sense the RCN-FULL can be considered as a *universal* network [7].

4) HARDWARE COST

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 VLSI area requirement of many networks are also known. However, actual parallel machines 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 VLSI area available per chip. This is particularly true for networks that have a relatively large number of links per PE such as the hypercube and the RCN-FULL. This has motivated the analysis presented in this paper about the chip pin requirements of the RCN-FULL and its comparison to that of the hypercube.

Let N be the number of network PEs, 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. For a hypercube, we assume that it is partitioned into smaller dimensional hypercubes with *MPC* PEs each, that is, $MPC = N / MCB$. Each PE will then be connected to $O(\log(N) - \log(N/MCB)) = O(\log(MCB))$ PEs in different chips. Thus, the total number of pins per chip is $MIOC = O(N/MCB(\log(MCB)))$.

Theorem 1: For a hypercube network with N processors partitioned over *MCB* chips, the pin requirement per chip, *MIOC*, is given by $MIOC = O(N/MCB(\log(MCB)))$.

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

Case 1: When the number of partitions (chips), *MCB*, is $\leq N^{1/2}$. In this case, we assume that the partitioning is being performed horizontally. That is, each chip will contain an integral number of $(N_A, L\text{-}1)$ RCN-FULLs.

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

Theorem 2: For an RCN-FULL of size N partitioned over *MCB* chips, the pin requirement per chip is given by $MIOC = O(N/MCB - N/MCB^2)$, where $MCB \leq N^{1/2}$.

Thus, when $MCB \leq N^{1/2}$, the pin requirements of the RCN-FULL is asymptotically less than that of the hypercube. Fig. 2 compares favorably the pin requirements of an RCC-FULL to that of the hypercube. Moreover, since the number of pins per chip cannot be arbitrarily high, many of the hypercube chip requirements are unrealistic.

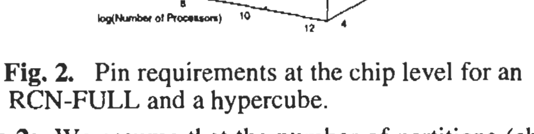


Fig. 2. Pin requirements at the chip level for an RCN-FULL and a hypercube.

Case 2: We assume that the number of partitions (chips) of an RCN-FULL is $MCB = I \times N^{1/2}$, where I is an integer ≥ 2. That is, each $(N_A, L\text{-}1)$ RCN-FULL is partitioned over exactly I chips.

In this case the number of pins per chips, *MIOC*, would correspond to *transpose* links *and* to *internal* links of an $(N_A, L\text{-}1)$ RCN-FULL. The number of pins that correspond to *transpose* links is $O(N/MCB_1 - N/MCB_1^2)$, where $MCB_1 = N^{1/2}$. Therefore, the pins that correspond to *transpose* links, $MIOC_T$, is $O(N^{1/2} - 1)$ for each $(N_A, L\text{-}1)$ RCN-FULL. For the number of pins that correspond to *internal* links, we have two cases:

A) When $L = 1$, 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 $I \leq N^{1/2}$, it would correspond to **Case 1**.

Theorem 3: For an RCN-FULL of size N partitioned over *MCB* chips, and $MCB = I \times N^{1/2}$, the pin requirement per chip is $MIOC = O(N/I - N/I^2 + N^{1/2}/I - 1)$ when $L = 1$, and is $MIOC = O(N^{1/2}/I - N^{1/2}/I^2 + N^{1/2}/I - 1)$ when $L > 1$.

In this case, the pin requirements for an $(N_A, 1)$ RCN-FULL is asymptotically equivalent to that of a hypercube. However, when $L > 1$, the pin requirements of the RCN-FULL is asymptotically less than those of a hypercube.

5) CONCLUSION

We have presented a new interconnection network, RCN-FULL, for the construction of massively parallel computers. Its construction and some of its key properties have been shown. A sorting algorithm is shown which has $O(\log(N))$ time and the RCN-FULL has been shown to emulate the CRCW PRAM model in $O(\log(N))$ time. The hardware cost of the RCN-FULL under pin limitations has been compared favorably to that of the hypercube. RCN-FULL offers good potential as a network for systems which emulate the PRAM models and which can be considered as *universal* networks.

References

1. Alt H., Hagerup T., Mehlhorn K.., "Deterministic simulation of idealized parallel computers on more realistic ones," *SIAM J. Comput.*, pp. 808-835, 1987.
2. R. Cole, "Parallel merge sort," *SIAM J. Comput.*, Vol. 17, pp. 770-785, 1988.
3. Fortune S. and Wyllie J., "Parallelism in random access machines," *Proc. ACM Symp. Theory Comput.*, 1978, pp. 114-118.
4. Hamdi M. and Hall R. W., "An efficient class of interconnection networks of parallel computations," *The Computer Journal*, 1994.
5. Hamdi M., *Communication-Efficient Interconnection Networks for Parallel Computations.* Ph.D. Thesis, Department of Electrical Engineering, University of Pittsburgh, 1991.
6. Maberg J. M., and Gafni E., "Sorting in constant number of row and column phases in a mesh," *Algorithmica*, pp. 561-572, 1988.
7. Miller R., and Stout Q. F. *Parallel Algorithms for Regular Architectures.* MIT Press, 1990.
8. Nassimi D., and Sahni S., "Data broadcasting in SIMD computers," *IEEE Trans. Comput.* Vol. 30, pp. 101-107, 1981.