

# Emulation of a PRAM on Leveled Networks

Michael Palis<sup>1</sup>, Sanguthevar Rajasekaran<sup>1</sup>, David S.L. Wei<sup>2</sup>

## ABSTRACT

There is an interesting class of ICNs, which includes the *star graph* and the *n*-way shuffle, for which the network diameter is *sub-logarithmic* in the network size. This paper presents, for the first time, optimal emulations of the CRCW PRAM on the star graph and the *n*-way shuffle. These results are special cases of a more general result that gives an optimal emulation of the CRCW PRAM on a large class of non-constant degree *leveled networks*. We also present an efficient emulation of the CRCW PRAM on a two-dimensional mesh. Although Ranade's emulation technique can be applied to the mesh to obtain an asymptotically optimal algorithm for emulation of the PRAM, the underlying constant in the time bound will be very large, say 100, making it uninteresting from a practical point of view. In this paper we provide a better emulation algorithm whose time bound is only  $4n + o(n)$ . This algorithm also has some nice 'locality' properties (e.g. if each request for memory access originates within a distance  $d$  of the location of the memory, then the algorithm terminates in  $6d + o(d)$  steps). The queue size of this algorithm is  $O(1)$ .

## 1 Emulating a PRAM on any ICN

### 1.1 Summary

In this section we give a brief summary of the emulation algorithm. Even though the following discussion assumes the EREW PRAM, it can be extended to other PRAMs (using tricks like 'message combining'). A single instruction of an EREW PRAM with  $N$  processors can be thought of as the following task. Each processor has a packet of information and also each processor wants to access the information some other processor has. The requests are such that any processor asks for exactly one packet and each processor's packet is asked by exactly one processor.

One way of handling these requests on an ICN will be as follows. Distribute the packets randomly among the processors (such that each packet is equally likely to end up in any processor). Now each processor accesses the packet it wants using any of the randomized packet routing algorithms. (Notice that the initial distribution of packets can also be accomplished with a routing algorithm). The analysis used in the randomized algorithms (like [13]'s and [4]'s) will imply that w.h.p. the above emulation procedure will terminate within time that is a constant factor of the diameter. The only problem with the above procedure is that if the packets are initially distributed randomly, there is no way of the processors knowing the address of the packets they want to access. Karlin and Upfal [2] avoided this problem by using a hash function (chosen randomly out of a 'small' class of hash functions) to perform the distribution. This hash function is picked by one processor (say processor 1) and is broadcast to every other processor.

---

<sup>1</sup>Department of Computer and Information Science, University of Pennsylvania

<sup>2</sup>Computer Science Department, Radford University

In summary, Karlin and Upfal's algorithm had four phases after processor 1 picks a hash function and the packets have been distributed according to this function. If processor  $i$  wants to access the packet that processor  $j$  has, then the following happens: 1) processor  $i$ 's request is sent to a random processor, say  $k$ ; 2) from  $k$ , the request is sent to processor  $h(j)$  where  $h$  was the hash function chosen; 3) if the request was 'read',  $h(j)$  sends the necessary packet to a random processor; and 4) finally the packet is sent to processor  $i$ .

Two of the four phases involve sending packets to random nodes. These are there only to simplify the analysis, and can indeed be eliminated (as in Ranade's [9] algorithm). Our algorithm consists of only two phases: 1) processor  $i$  sends a request to processor  $h(j)$  and 2) if the request was 'read',  $h(j)$  sends back the packet requested to processor  $i$ . We make use of the same class of hash functions used by Karlin and Upfal. Each one of the two phases of our algorithm corresponds to a routing task. Realize that each routing need not be a permutation. But each phase is a permutation request 'more or less' w.h.p.

We will be employing the above techniques for emulating an EREW PRAM on the leveled network as well as the Mesh.

## 1.2 Address Mapping

To emulate a PRAM of  $M$  address space on a network of  $N$  processors, the  $M$  shared memory locations are mapped onto the  $N$  memory modules of the network according to a randomly chosen hash function  $h$  from the following class of hash functions [2]:

$$H = \{h|h(x) = ((\sum_{0 \leq i < \delta} a_i x^i) \bmod P) \bmod N\}$$

where  $P$  is a prime,  $P \geq M$ ,  $a_i \in Z_P$ .  $\delta$  is chosen to be a constant multiple of the diameter of the ICN.

The above class of hash functions has the following interesting property:

**Fact 1.1** ([2]) If  $N$  items are mapped into  $N/2^i$  buckets using a random hash function (from the class defined before), the maximum number (call it  $Y_i$ ) of items mapped into a single bucket satisfies:

$$\text{Prob.}[Y_i \geq j] \leq N \left( \frac{2^i}{j - \delta} \right)^\delta.$$

## 2 Packet Routing

### 2.1 Various Types of Routing

The routing problem is defined as follows: Given a specific network and a set of packets of information in which a packet is a  $\langle \text{source}, \text{destination} \rangle$  pair. To start with, the packets are placed on their sources. These packets must be routed in parallel to their own destinations such that at most one packet passes through any link of the network at any time and all packets arrive at their destinations as quickly as possible. A paradigmatic case of general routing is **permutation routing** in which initially there is exactly one packet at each node and the destinations form some permutation of the sources. A more general case of routing is **partial  $h$ -relations routing** in

which initially there are at most  $h$  packets at any node and there are no more than  $h$  packets with the same destination. Though we will mainly focus on permutation routing, it is not hard to show that the proposed algorithms can be easily modified (in most cases) to other general routing problems.

### 3 PRAM Emulation on a Leveled Network

In this section we present details on the emulation of an EREW PRAM on a leveled network with sub-logarithmic diameter. Let  $l$  stand for the number of levels in the network. We make use of the address mapping discussed in the previous section to distribute messages. The only details of the emulation that are missing are a description of the packet routing algorithm to be used and an analysis of its run time. If we make use of the address mapping of Karlin and Upfal for a leveled network with  $l$  levels, Fact 3.1 implies that no processor will get more than  $O(l)$  packets in the initial distribution. (Here we assume  $l = O(\log N / (\log \log N))$ ).

#### 3.1 The Routing Algorithm

The algorithm used for routing packets is an adoption of the two phase routing algorithm invented by Valiant [13]. More details follow.

**Algorithm 3.1** {A universal Routing Algorithm}

*Phase 1*

**for** each packet  $x$  **do in parallel** starting from the first level traverse to a random node in the last level (i.e. level  $\ell$ ) by choosing a random link at each level.  
{The queuing discipline is first-in first-out (FIFO).}

*Phase 2*

Send each packet  $x$  from its intermediate node to its correct destination along the unique path.

**Theorem 3.1** For a leveled network of  $\ell N$  nodes with  $\ell$  levels, any permutation routing of  $N$  packets<sup>3</sup> (from the first column to the last column) can be completed in  $\tilde{O}(\ell)$  steps provided that  $d \geq 2$ , where  $d$  is the degree of the network. The queue needed for each link is FIFO of size  $\tilde{O}(\ell)$ .

**Proof Sketch:** Let  $\pi$  be an arbitrary packet in the network. We compute an upper bound on the delay this packet suffers as follows. Let  $d_i$  stand for the number of packets that meet  $\pi$ 's path for the first time at level  $i$  (for  $1 \leq i \leq l$ ). Since the algorithm we use for routing is **non repeating**, we can use the *queue line lemma* [13] to infer that an upper bound for the delay  $\pi$  suffers is  $\sum_{i=1}^l d_i$ .

The generating function for  $\text{Prob.}[d_i = k]$  is calculated as  $G_i(x) = e^{x/d^2}$ ,  $d$  being the degree of the network. The generating function (call it  $G(x)$ ) for  $\sum_{i=1}^l d_i$  is the product of the  $G_i$ 's.  $G(x)$  simplifies to  $e^{xl/d^2}$ . This immediately implies that

$$\text{Prob.}[\sum_{i=1}^l d_i \geq q] \leq \sum_{x=q}^{\infty} \left(\frac{l}{d^2}\right)^x \frac{1}{x!}$$

---

<sup>3</sup>The result can easily be extended for permutations of  $\ell N$  packets.

If  $q$  is chosen to be  $c\alpha$  (for some appropriate constant  $c$ ), the above probability can be shown to be no more than  $N^{-\alpha}$ □.

**Note:** This proof assumes that there is only one packet to start with at any node in the ICN. A similar proof can be given to show that even if there are  $l$  packets to start with in every node the above algorithm terminates in  $\tilde{O}(l)$  steps. (For details see [14]).

These results yield the following

**Theorem 3.2** Each step of the EREW PRAM can be emulated by a leveled network of  $\ell$  levels with degree  $d$ ,  $\ell = O(d)$ , in  $\tilde{O}(\ell)$  steps.

**Corollary 3.1** Each step of the EREW PRAM can be emulated optimally on the  $n$ -star graph and the  $n$ -way shuffle since these are leveled networks.

**Note:** The above emulation algorithms can be extended to CRCW PRAMs also using some additional tricks like message combining. Details can be found in [14].

## 4 Emulating a PRAM on the Mesh Connected Computers

Mesh Connected Computers (MCCs) are increasingly being accepted as a feasible model for building machines for many reasons including their simple interconnection, linear scalability etc. Several machines have already been built based on this model (e.g., ILLIAC IV, Massively Parallel Processor (MPP), Blitzen etc.) Thus it is interesting to study the emulation of PRAMs on the MCCs. Even though an asymptotically optimal algorithm for emulating a PRAM on the MCC is implied by Ranade's algorithm, the underlying constant in the time bound is very high ( $> 100$ ). Since a MCC has a large diameter, any algorithm on it will have to have a time bound within a small constant factor of its diameter in order to be practical. In this section such an algorithm for emulation of a PRAM is given. On an  $n \times n$  Mesh our algorithm takes  $4n + o(n)$  steps for emulating a single step of an EREW PRAM. If each request for memory access originates within a distance  $d$  of the location of the memory, the same algorithm terminates in time  $6d + o(d)$  steps with high probability (abbreviated as w.h.p. here after). Our algorithm needs a queue size of only  $O(1)$  with overwhelming probability. By high (or overwhelming) probability we mean a probability of  $\geq (1 - N^{-\alpha})$ , for any constant  $\alpha \geq 1$ ,  $N$  being the network size.

### 4.1 Model Definition

A MCC is nothing but an  $n \times n$  square grid in which each grid point corresponds to a processing element and each edge corresponds to a communication link. Thus each processor has 4 or less neighbors. We assume that all the links are bi-directional. Several variations of this topology can be found in the literature. The model we use (called the MIMD) has been assumed in previous works (see e.g., [13], [4], [5], [6], [8]). In a single step each processor can perform a local computation (like a comparison) and also communicate with all its (4 or less) neighbors.

### 4.2 Preliminaries

In this section we state some facts that will prove useful in our algorithm.

Figure 1: Partitioning of the Mesh

One of the subroutines to be used in our algorithm is for permutation routing. A vast amount of literature exists on this topic. Valiant and Brebner [13] started the research on routing on a Mesh giving a  $3n + o(n)$  time and  $O(\log n)$  queue randomized algorithm. This work was followed by Krizanc, Rajasekaran, and Tsantilas [4] who presented a  $2n + O(\log n)$  time  $O(1)$  queue randomized algorithm and Kunde [5] who gave a  $2n + O(n/q)$  time algorithm with a queue size of  $q$  (for any  $1 \leq q \leq n$ ). Finally Leighton, Makedon, and Tollis [6] displayed a  $2n - 2$  time algorithm with a queue size of roughly 672. Recently, Rajasekaran and Overholt [8] have presented a  $2n - 2$  step algorithm with a queue size of only 58.

The emulation algorithm consists of two phases as explained in Summary. We use the same address mapping. The following facts about the address mapping will be useful in our analysis. The following three corollaries easily follow and will be used in our analysis.

**Corollary 4.1** If  $N$  items are mapped into  $N$  buckets, no single bucket will get more than  $\tilde{O}\left(\frac{\log N}{\log \log N}\right)$  items. If  $N = n^2$  items are mapped into  $\beta n$  buckets ( $\beta$  being a constant), the maximum number of items mapped into any bucket will not be more than  $\frac{n}{\beta} + \tilde{O}(n^{3/4})$ . If  $N$  items are mapped into  $N$  buckets, and if  $S$  is a collection of  $\log N$  buckets, the number of items mapped into  $S$  will not exceed  $\tilde{O}(\log N)$ .

Next we describe the routing algorithm to be used in each phase.

### 4.3 Our Routing Algorithms

In both the phases we make use of the same routing algorithm. We make use of the same algorithm as the one given in [4], with a different analysis. Each phase will be finished in  $2n + o(n)$  steps w.h.p. First we present a routing algorithm with the promised time bound but which needs a queue size of  $O(\log N)$ . Later we will describe how to bring down the queue size to  $O(1)$ .

Partition the  $n \times n$  Mesh into horizontal slices with  $\epsilon n$  (for some  $\epsilon$  to be fixed) rows in each slice (see Figure 5). There are three stages in the algorithm. Contention for edges are resolved by furthest destination first queueing discipline. Let  $\pi$  be a packet that originates in node  $(i, j)$  (i.e., in row  $i$  and column  $j$ ) and whose destination is  $(k, l)$ .

#### stage1

$\pi$  chooses a random node (call it  $(i', j)$ ) in the column of its origin in the same slice and traverses to that node along column  $j$ .

#### stage2

$\pi$  traverses along row  $i'$  to the node  $(i', l)$ .

#### stage3

Finally the packet  $\pi$  traverses along column  $l$  to the node  $(k, l)$ .

### 4.3.1 Analysis of the Algorithm

Consider the following routing problem on a linear array of size  $n$ . There are  $k_i$  packets to start with at node  $i$  (for  $1 \leq i \leq n$ ) such that  $\sum_{i=1}^n k_i = n'$ . Each node chooses a random node in the linear array as its destination. How fast can this routing be performed (assuming the furthest destination first priority scheme)?

The answer is  $n' + o(n)$  for the following reason. Let  $i$  be the origin of a packet  $\pi$  and let  $j$  be its destination. W.l.o.g. assume  $j$  is to the right of  $i$ . Since all the links are bidirectional, for the worst case analysis we can assume all the packets are traversing from left to right. The number of packets that will have a higher priority than  $\pi$  is given by the binomial  $B((n-j)n', 1/n)$ . Using Chernoff bounds, this number is no more than  $\frac{(n-j)n'}{n} + o(n)$  w.h.p. Applying the queueing lemma [13], the time needed for  $\pi$  to reach its destination is no more than  $(j-i) + \frac{(n-j)n'}{n} + o(n)$  w.h.p. In the worst case this time bound is  $n' + o(n)$ .

If we apply the above fact to the first stage of our routing algorithm we see that the time bound for stage 1 is no more than  $\epsilon n + o(n)$ . (Realizing that the number of packets originating from any column slice is no more than  $\epsilon n + o(n)$  w.h.p. (see Corollary 4.1). If we fix  $\epsilon$  to be  $1/(\log n)$ , the time needed for stage 1 is  $o(n)$  w.h.p.

In the second stage of the routing algorithm, consider any row  $j$ . How many packets will there be in row  $j$  at the beginning of stage 2? Using Corollary 4.1 and Chernoff bounds one can readily see that this number is no more than  $n + o(n)$  w.h.p. Given this fact, we can use arguments similar to the one given for linear array to prove that both the second and the third phases will be completed in  $n + o(n)$  each w.h.p.

Using Corollary 4.1, we can also prove a queue size of  $O(\log n)$ .

This proves the following

**Theorem 4.1** *The routing algorithm described terminates in  $2n + o(n)$  steps w.h.p. The queue size is  $O(\log n)$ .*

The above theorem together with the emulation algorithm described before will yield the following

**Theorem 4.2** *Each instruction of an EREW PRAM can be emulated on the MCC in  $4n + o(n)$  steps w.h.p. The queue size of the processors is  $O(\log n)$ .*

We can reduce the queue size of the above algorithm to  $O(1)$  making use of Corollary 4.1. The improvement will parallel the  $2n + O(\log n)$  time routing time algorithm presented in [4], with a slightly different analysis. In similar lines we can also prove the following

**Theorem 4.3** *If each memory request originates within a distance of  $d$  of the location of the memory, the above emulation algorithm terminates in  $6d + o(d)$  steps w.h.p.*

## 5 Conclusions

In this paper we have presented optimal algorithms for emulating a PRAM on more realistic machine models. The model we considered was a leveled network with sub-logarithmic diameter. We also presented a  $4n + o(n)$  steps emulation algorithm for an  $n \times n$  mesh. Even though Ranade's algorithm will imply an asymptotically optimal algorithm for emulation on the MCC, the underlying constant

will be impractically large. For a mesh, a large constant is particularly intolerable owing to its large diameter.

## References

- [1] Alt, H., Hagerup, T., Mehlhorn, K., and Preparata, F., 'Deterministic Simulation of Idealized Parallel Computers on More Realistic Ones,' *SIAM Journal on Computing*, vol. 16(5), 1987, pp. 808-835.
- [2] Karlin, A., and Upfal, E., 'Parallel Hashing—An Efficient Implementation of Shared Memory,' *Proc. ACM Symposium on Theory Of Computing*, 1986, pp. 160-168.
- [3] Karp, R., and Ramachandran, V., 'Parallel Algorithms for Shared-Memory Machines,' in *Handbook of Theoretical Computer Science*, North-Holland, 1990.
- [4] Krizanc, D., Rajasekaran, S., and Tsantilas, T., 'Optimal Routing Algorithms for Mesh-Connected Processor Arrays,' *Proc. AWOC 1988*, Springer-Verlag Lecture Notes in Computer Science 319, pp. 411-422. Also submitted to *Algorithmica*, 1989.
- [5] Kunde, M., 'Routing and Sorting on Mesh-Connected Arrays,' *VLSI Algorithms and Architectures: Proc. AWOC 1988*, Springer-Verlag Lecture Notes in Computer Science 319, pp. 423-433.
- [6] Leighton, T., Makedon, F., and Tollis, I.G., 'A  $2n - 2$  Step Algorithm for Routing in an  $n \times n$  Array With Constant Size Queues,' *Proc. 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 328-335.
- [7] Palis, M., Rajasekaran, S., and Wei, D., 'General Routing Algorithms for Star graphs,' in *Proc. Fourth International Parallel Processing Symposium*, April, 1990, pp. 597-611.
- [8] Rajasekaran, S., and Overholt, R., 'Constant Queue Routing on a Mesh,' *Proc. Symposium on Theoretical Aspects of Computer Science*, Hamburg, Germany, Feb. 1991.
- [9] Ranade, A.G., 'How to Emulate Shared Memory,' *Proc. IEEE Symposium on Foundations Of Computer Science*, 1987, pp. 185-194.
- [10] Snyder, L., 'Type architectures, shared memory, and the corollary of modest potential,' *Annu. Rev. Comput. Sci.* 1, 1986, pp. 289-317.
- [11] Upfal, E., and Wigderson, A., 'How to Share Memory in a Distributed System,' *IEEE Symposium on Foundations Of Computer Science*, 1984, pp. 171-180.
- [12] Valiant, L.G., 'A bridging model for Parallel Computation,' *CACM*, August 1990, Vol.33, No 8, pp. 103-111.
- [13] Valiant, L.G., and Brebner, G.J., 'Universal Schemes for Parallel Communication,' *Proc. ACM Symposium on Theory Of Computing*, 1981, pp. 263-277.
- [14] Wei, D.S.L., 'Fast Parallel Routing and Computation on Interconnection Networks,' Ph.D. Thesis, Univ. of Pennsylvania, Jan. 1991.