Emulation of a PRAM on Leveled Networks

Michael Palis
University of Pennsylvania

Sanguthevar Ranjasekaran
University of Pennsylvania

David S. L. Wei
Radford University

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
http://repository.upenn.edu/cis_reports/402


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/402
For more information, please contact repository@pobox.upenn.edu.
Emulation of a PRAM on Leveled Networks

Abstract
We present efficient emulations of the CRCW PRAM on a large class of processor interconnection networks called leveled networks. This class includes the star graph and the n-way shuffle, which have the interesting property that the network diameter is sub-logarithmic in the network size. We show that a CRCW PRAM can be emulated optimally on these networks (i.e., each emulation step takes time linear in the network diameter). This is the first result that demonstrates PRAM emulation in less than logarithmic time. We also present an efficient emulation of the CRCW PRAM on an n x n mesh. Although an O(n)-time emulation algorithm for the mesh is known, the underlying constant in the run-time is large, making it impractical. We give an improved emulation algorithm whose time bound is only 4n + o(n).

Comments

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/402
Emulation of a PRAM on Leveled Networks

Michael Palis and Sanguthevar Rajasekaran
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

David S. L. Wei
Department of Computer Science
Radford University
Radford, VA 24142

ABSTRACT

We present efficient emulations of the CRCW PRAM on a large class of processor interconnection networks called leveled networks. This class includes the star graph and the n-way shuffle, which have the interesting property that the network diameter is sub-logarithmic in the network size. We show that a CRCW PRAM can be emulated optimally on these networks (i.e., each emulation step takes time linear in the network diameter). This is the first result that demonstrates PRAM emulation in less than logarithmic time.

We also present an efficient emulation of the CRCW PRAM on an \( n \times n \) mesh. Although an \( O(n) \)-time emulation algorithm for the mesh is known, the underlying constant in the run-time is large, making it impractical. We give an improved emulation algorithm whose time bound is only \( 4n + o(n) \).
1 Introduction

The parallel random-access machine (PRAM) model has become a popular vehicle for investigating parallel algorithms for a wide variety of problems such as sorting, graph and matrix problems, computational geometry, etc. [5]. It is an abstract parallel computer model consisting of an arbitrary number of processors that communicate via a shared global memory. Each memory access to the shared memory is assumed to take unit time. This unit-time memory access property simplifies programming because it permits parallel algorithms to be designed and analyzed solely on the basis of their computational requirements, divorced from issues of interprocessor communication.

Considerable research has been done on efficient emulations of the PRAM model on physically realizable models such as processor interconnection networks (ICNs). In an ICN, each processor has its own private memory and can communicate with other processors only by explicit packet routing. In [13], Ranade showed that one step of a concurrent-read, concurrent-write (CRCW) N-processor PRAM can be emulated by a butterfly network with the same number of processors in time $O(\log N)$ with high probability. This result also implies an $O(\log N)$ emulation of the CRCW PRAM on the binary n-cube network with $N = 2^n$ processors. Furthermore, the emulations are optimal because both butterfly and n-cube networks have $O(\log N)$ diameter.

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. There are no known efficient emulations of the CRCW PRAM on these networks; indeed, it was not known until [10] whether packet routing can be performed on these networks in optimal time (i.e., linear in the network diameter). 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. Ranade's emulation technique can be applied to the mesh to obtain an asymptotically optimal algorithm for emulation of the PRAM. However the underlying constant in the time bound will be very large making it uninteresting from a practical point of view. To be more precise, Ranade’s algorithm runs in time $O(n)$ on an $n \times n$ mesh. The underlying constant is roughly 100. Since a mesh has already a large diameter (i.e., $2n - 2$), any algorithm on it will be practical only if the time bound is within a small constant factor of its diameter. In this paper we provide such an algorithm whose time bound is only $4n + o(n)$. This algorithm also has some nice ‘locality’ properties. In particular, 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 $O(1)$. 

2
2 Emulation on Leveled Networks

2.1 Address Mapping

In previous works, Valiant's two phase randomized routing strategy has been used to effectively emulate the PRAM. If items stored in the common memory of a PRAM are uniformly randomly distributed among the local memories of the emulating network, and each processor knows where the items are, without any modification, Valiant's two phase randomized routing strategy can be used alone to emulate the PRAM. Unfortunately, in order to emulate a read/write request, each processor has to know the address of the data it wants. However, [4]'s technique of parallel hashing can be employed together with Valiant's two phase routing algorithm to solve this problem.

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:

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

where \( P \) is a prime, \( P \geq M \), \( a_i \in Z_P \), and \( \delta = c\ell \) for some constant \( c \) chosen to guarantee the desired performance. \( \ell \) is the diameter of emulating leveled network. Each memory location \( x \) in the PRAM's memory is mapped to a memory module whose identification number is \( h(x) \), where \( h(x) \) is randomly chosen from \( H \). Each PRAM instruction can be emulated by sending read/write request packets from each processor to the memory module holding the item it wishes to access and back in case of a read instruction. The source of each packet is the label of the sending processor, and the destination of the packet is the label of the memory module which holds the item the packet wishes to access. The communication is supposed to be finished in \( \tilde{O}(\ell) \) time. If within the allotted time the communication has not been completed, a designated processor chooses a new hash function, and all the \( M \) memory locations are remapped to their new locations in distributed memory modules. Although rehashing is very expensive, rehashings hardly happen. It is also worth pointing out that each hash function in \( H \) needs only \( O(\ell \log M) \) bits to describe. This makes our scheme practical.

2.2 Packet Routing

2.2.1 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. Another case of routing is partial routing in which initially there is
at most one packet at each node and all packets have distinct destinations. A more general case of routing is\textbf{ 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. Also,\textbf{ many-one routing} is defined as one where initially there is at most one packet at every node and some of the packets may have 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.

A routing algorithm is said to be\textbf{ oblivious} if the path taken by each packet depends only on its own source and destination. A routing algorithm is\textbf{ non-oblivious} otherwise. Batcher's \textit{sorting} algorithms are examples of non-oblivious routing algorithms. They require $\Theta(\log^2 N)$ routing time for the cube class networks or $7n$ routing time for the $n \times n$ mesh-connected arrays and hence are not optimal and only work for permutation routing although they possess the advantage that they need not have queues. An oblivious routing strategy is preferable since it will lead to a simple control structure for the individual processing elements. Also oblivious routing algorithms can be used in a distributed environment. We will be only concerned with oblivious routing strategies.

We use $\tilde{O}$ to represent the complexity bounds of randomized algorithms. We say a randomized algorithm has resource (like time, space etc.) bound of $\tilde{O}(g(n))$ if there exists a constant $c$ such that the amount of resource used by the algorithm (on any input of size $n$) is no more than $cg(n)$ with probability $\geq 1 - \frac{1}{n^c}$. Under this notation Valiant's algorithm runs in time $\tilde{O}(\log N)$ on an $n$-cube of $N = 2^n$ nodes.

We first define the \textbf{path} (route) of a packet $x$ as the sequence of nodes and links that the packet $x$ ever travels. Also we define the \textbf{delay} of a packet $x$ in a run of a routing scheme as the total number of time units during which $x$ waits unserved in queues of nodes along its path. The performance of any routing scheme is usually assessed in terms of its\textbf{ routing time, queue size,} and \textbf{queueing discipline}. The routing time is the number of steps taken by the last packet to reach its destination. The number of steps taken by a packet $x$ is simply the sum of the \textbf{delay} of $x$ and the length of the \textbf{path} of $x$. It determines how fast a routing can be finished. The maximum number of packets residing at any node at any time step during the entire course of routing determines the queue size of a routing scheme and consequently determines the amount of additional hardware needed per node. The queueing discipline is a strategy of the processors in the network to assign priority to the packets queued. A first-in first-out (FIFO) is a simpler queueing strategy than that of a priority queue and is thus preferable.

\textbf{Definition 2.1} A routing scheme is\textbf{ nonrepeating} provided the following is true: if the paths taken by any two distinct packets share some links and then diverge, then the remainder of these two paths will never share any link again.

\textbf{Definition 2.2} A \textbf{queue line} is a directed path taken by a packet $x$, together with the packets that overlap with $x$. (Two packets are said to overlap if there are $\geq 1$ common links in their paths.)

\textbf{Fact 2.1 (Queue line lemma)} The number of steps a packet $x$ is delayed is less than or equal to the number of packets that overlap with $x$ provided the routing scheme is nonrepeating.
Proof: Refer to [19]. □

2.2.2 Some Facts from Probability Theory

The following facts are needed for the analysis required to prove the performance of randomized routing schemes.

Consider a sequence of N independent trials of a two outcome experiment. If every trial has the same success (or failure) probability, then they are called Bernoulli trials. Otherwise they are Poisson trials. Let the probability that at least m of the trials succeed in a sequence of N independent Bernoulli trials (each with probability P of success) be denoted by \( B(m, N, P) \).

Fact 2.2 (Hoeffding) If we have N independent Poisson trials with probability of success \( p_1, p_2, \ldots, p_N \) and if
\[
NP = \sum_{i=1}^{N} p_i
\]
and \( m \geq NP + 1 \) is an integer, then the probability of at least m success out of N Poisson trials is \( \leq B(m, N, P) \).

Fact 2.3 (Chernoff) If \( m \geq NP \) is an integer, then
\[
B(m, N, P) \leq \left( \frac{NP}{m} \right)^m e^{m-NP}.
\]

Definition 2.3 Let X be a discrete random variable taking on only nonnegative integer values. The function \( G(z) \), called the generating function of X, is defined by
\[
G(z) = \sum_{k=0}^{\infty} p_k z^k, \text{ where } p_k = p(X = k).
\]

Fact 2.4 Let \( X_i, 1 \leq i \leq s \) be independent random variables such that \( G_i(z) \) represents generating function of \( X_i \). The generating function for \( \sum_{i=1}^{s} X_i \) is given by \( \prod_{i=1}^{s} G_i \).

2.3 A Universal Optimal Randomized Routing Algorithm on the Leveled Networks

In this subsection, we provide a universal routing algorithm and network-independent analysis which works for both constant degree and non-constant degree leveled networks (although the algorithm doesn’t guarantee a constant queue size). We will make use of this algorithm for communication in the emulation of PRAM.

2.3.1 Definition of a Leveled Network

A leveled network of \( \ell N \) nodes is a network with \( \ell \) groups of nodes such that each group has \( N \) nodes and these groups form a sequence of \( \ell \) columns (one group per column), say \( c_1, c_2, \ldots, c_\ell \).
The only links in the network are between nodes in $c_i$ and nodes in either $c_{i+1}$ or $c_{i-1}$, $1 < i < \ell$. Every node in each column has at most $d$ incoming and outgoing links where $d$ is the degree of the network. Each node in the columns except the first and the last ones can be either a processor or a switch. For each node in the first column, there exists a unique path of length $\ell$ connecting it to any node in the last column. See Figure 1. Many classical networks, like hypercube, butterfly, etc., fall into this class of networks. A new interconnection network called star graph ([2, 1]) which has better properties than the hypercube is also in this class. The routing on these networks will be discussed in subsequent sections. The reader can easily see why these networks can be expressed in the form of leveled networks, and why the algorithm together with its analysis discussed in this section is also applicable to these networks.

2.3.2 The Universal Routing Algorithm

**Fact 2.5** For any routing scheme which runs in $O(T(n))$ or $\tilde{O}(T(n))$ time, the queue needed for each link is of size $O(T(n))$.

**Proof:** If there exists one link whose queue size $> O(T(n))$, then the processor to which the link is incident will take $> O(T(n))$ steps to process those packets waiting in the queue, and the routing time of the scheme will be $> O(T(n))$ rather than $O(T(n))$.\[\square\]

**Theorem 2.1** For a leveled network of $\ell N$ nodes with $\ell$ levels, any permutation routing of $N$ packets\(^1\) (from the first column to the last column) can be completed in $\tilde{O}(\ell)$ steps provided that

\(^1\)The result can easily be extended for permutation of $\ell N$ packets
\[ d \geq 2, \text{ where } d \text{ is the degree of the network}^{2}. \text{ The queue needed for each link is FIFO of size } O(\ell). \]

To prove this theorem, we first present the routing algorithm.

**Algorithm 2.1** {A universal Routing Algorithm}

**Phase 1**

\[ \text{for each packet } x \text{ do in parallel select a random link as a bridge to go to the} \]

next level by flipping a \( d \) sided coin where \( d \) is the number of outgoing links of

the node at which the packet is residing.

\{Each packet will reach a node in the last column

which is a random intermediate node.\}

\{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.

**Proof of Theorem 2.1:** See [20]

Next we show how Theorem 2.1 can be applied to some frequently used networks.

### 2.3.3 Routing on the \( n \)-Star Graph

In [1, 2], an algorithm was presented for routing a **single** packet from a source to an arbitrary
destination on the star graph. The more general problem of permutation routing was not considered.
In this section, we present efficient deterministic and randomized algorithms for permutation routing
on the star graph. Both algorithms are oblivious.

### 2.3.4 The Star Graph

**Definition 2.4** Let \( d_1d_2\ldots d_n \) be a permutation of \( n \) symbols, \( e.g., 1\ldots n \). For \( 1 < j \leq n \), we
define \( SWAP_j(d_1d_2\ldots d_n) = d_jd_2\ldots d_{j-1}d_{j+1}\ldots d_n \).

**Definition 2.5** An \( n \)-star graph is a graph \( G=(V,E) \) with \( |V| = n! \) nodes, where \( V = \{d_1d_2\ldots d_n \mid d_1d_2\ldots d_n \text{ is a permutation of } 1\ldots n\} \), and \( E = \{(u,v) \mid u,v \in V \text{ and } v = SWAP_j(u) \text{ for some } j, 1 < j \leq n\} \).

The 3-star and 4-star graphs are depicted in Figure 2. In [2], Akers, Harel and Krishnamurthy have
shown that the star graph is superior to the \( n \)-cube with respect to the degree and diameter. An
\( n \)-star graph has \( n! \) nodes, degree \( n - 1 \), and diameter \( [\frac{3}{2}(n - 1)] \). On the other hand, an \( n \)-cube
has \( 2^n \) nodes, degree \( n \), and diameter \( n \). Thus, the degree and diameter of the star graph grows
more slowly as a function of the network size than does the \( n \)-cube. Moreover, the star graph is
both vertex symmetric and edge symmetric (just like the \( n \)-cube.) Oftentimes, these properties
lead to a simpler analysis of the routing algorithm.

\[ ^2\text{We will prove Theorem 2.1 only for Phase 1 and it will be clear how the proof can be modified to apply to the} \]

second phase as a mirror image of the first phase.
Definition 2.6 A subgraph of an $n$-star graph $G$ is said to be an $i$-th stage subgraph, denoted $G^i$, iff $G^i$ is itself an $(n-i)$-star graph, $0 \leq i < n$, and the last $i$ symbols of the labels of all nodes in it are identical.

The $G^i's$ of any $G^{i-1}$ partition $G^{i-1}$ into $n-i+1$ identical subgraphs. Let’s define the stage of the network during a run of the routing algorithm to be simply the collection of the nodes together with the packets each node holds in its queue. Hence the routing algorithm can be thought of as a sequence of stage transitions $S_1, ..., S_f$, where in $S_1$ each node has a single packet that originated in that node, and in $S_f$ each node has a single packet that is destined for it.

Look at all the $G^i's$ that constitute any $G^{i-1}$. It is easy to see that for any node $u$ in any one of these $G^i's$, there is exactly one other node $v$ adjacent to $u$ such that $v$ is contained in some other $G^i$. We call $v$ the critical point to $u$ and vice-versa, at stage $i$. For example, in Figure 2(b), node $BACD$ is a critical point to node $DACB$ at stage 1.

Definition 2.7 A stage $S_i$ is said to be $i$-th stage stable, denoted $S_i^{stable}$, iff for every $i$-th stage subgraph $G^i$, the destination of each packet in the subgraph is in the subgraph itself.

An interesting question is: For those non-constant degree networks with sublogarithmic diameter, like $n$-star graph, can we route (using randomization) a permutation request in sublogarithmic steps with high probability? Valiant [19] has shown that permutation routing can be done on the $d$-way shuffle graph (which has $N = d^n$ nodes and diameter $n$) in $\tilde{O}(n \log d / \log \log d)$ steps. For the $n$-way shuffle graph, Valiant’s algorithm runs in time $\tilde{O}(n \log n / \log \log n)$ and hence is not optimal. In this subsection, we present a randomized routing algorithm [10] for the $n$-star graph that runs in time of the order of the diameter with high probability.
Algorithm 2.2

Phase 1

Step 1: for each packet $x$ do in parallel select a random intermediate node.

Step 2: Send the packets to their intermediate random destinations.

{The queuing discipline is FIFO.}

Phase 2

Send each packet $x$ from its intermediate node to its correct destination.

Theorem 2.2 For the $n$-star graph (parallel model) of $N = n!$ nodes, any permutation routing can be completed by a randomized routing algorithm in $\tilde{O}(n)$ steps. The queue needed for each link is FIFO of size $O(n)$.

Proof: Our proof will be simplified if it is given using the logical network. A logical network for the 3-star graph is shown in Figure 3 which is a leveled network of $O(n)$ levels and with degree $O(n)$.

Replace $d$ and $\ell$ by $cn$ in the proof of Theorem 2.1. □

2.3.5 Routing on the $d$-Way Shuffle

The same proof technique can also be used to analyze the behavior of a simple but efficient randomized routing algorithm for the $d$-way shuffle. Our routing algorithm for the $n$-way shuffle achieves a better (in fact, optimal) time bound than that of [19].

A $d$-way shuffle network has $N = d^n$ nodes. Each node can be labelled as $d_n d_{n-1} \ldots d_1$ where each $d_i$ is a $d$-ary digit. A node labelled $d_n d_{n-1} \ldots d_1$ is connected to the nodes labelled $ld_n d_{n-1} \ldots d_2$ where $l$ is an arbitrary $d$-ary digit. Therefore, the network has diameter $n$ and a unique path of
exactly $n$ links between any pair of nodes. If we choose $d = n$, then the network is an $n$-way shuffle. Figure 4 shows an $n$-way shuffle network with $n = 2$. We can view this network as a leveled network of 2 levels with degree 2. The following algorithm can be used to perform permutation routing on the $n$-way shuffle.

**Algorithm 2.3**

**Phase 1**

*Step 1:* for each packet $x$ do in parallel select a random intermediate node.

*Step 2:* Send the packets along the unique path to their intermediate random destinations.

{The queuing discipline is FIFO}

**Phase 2**

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

**Theorem 2.3** For the $n$-way shuffle network (parallel model) of $N = n^n$ nodes, any permutation routing can be performed by a randomized routing algorithm (using Algorithm 2.3) in $\tilde{O}(n)$ steps. The queue needed for each link is FIFO of size $O(n)$.

**Proof:** The same as that of Theorem 2.1 (replace $\ell$ and $d$ by $cn$).\(\square\)

The emulating network we will use is a leveled network in which $\ell = O(d)$. $n$-star graph and $n$-way shuffle, in which $\ell = O(d) = O(n)$, are examples of this sort of networks. Figure 3 depicts the logical network of the $n$-star graph which is a leveled network of degree $n = 3$, and Figure 4 shows an $n$-way shuffle which is a leveled network of degree $n = 2$. Both of these two networks have $O(n)$ levels. For discussion convenience and without loss of correctness, we assume that nodes in the first column are processors and nodes in the last column are memory modules which are numbered.
as 0,⋯,N - 1. The routing algorithm we use for communication is the one introduced in section 3.2. Suppose S is the set of items being requested by processors for executing a PRAM instruction, |S| ≤ N. If we could prove that with very high probability (say 1 - \(\frac{1}{N^{c'}}\), c' being a constant > 0), no more than \(O(1)\) items from S will be mapped onto the same memory module, then the routing algorithm in section 3.2 together with its analysis can be directly used to prove the desired performance of the emulation. Unfortunately, with \(N^{-\beta}\) (for some \(\beta > 0\)) probability, at least one node will get \(\ell\) items. However, even if we allow \(\ell\) items to be mapped into each memory module, the desired performance can be obtained. In order to obtain the desired performance, same routing algorithm will be used but the analysis is different. We will first prove that the algorithm in section 3.2 can perform a partial \(\ell - relation\) routing in \(\tilde{O}(\ell)\) time, and then, in the next section, we will prove that with extremely high probability no more than \(\ell\) items from S will be mapped into the same memory module.

**Theorem 2.4** For the leveled network of \(\ell\) levels with degree \(d, \ell = O(d)\), any partial \(\ell - relation\) routing can be completed by a randomized routing algorithm (using Algorithm 2.1) in \(\tilde{O}(\ell)\) steps.

We need the following lemma in the proof of Theorem 2.4.

**Lemma 2.1** If a routing algorithm X can realize any permutation in \(c_1 f(N)\) steps with probability \(\geq (1 - \frac{1}{N^{c'}}), c' > 0\), then we can make use of this algorithm to perform any permutation routing in \(c_1c_2 f(N)\) steps with probability \(\geq (1 - \frac{1}{N^{2c'}})\).

**Proof:** To prove this lemma, we simply repeat algorithm X for a constant number of times, say \(c_2\). In each run of algorithm X, those packets that have not reached their destinations in \(c_1 f(N)\) steps will trace back their paths and reach their sources in \(c_1 f(N)\) steps or less and these packets will repeat algorithm X. Clearly, the probability of \(\geq 1\) unsuccessful packets in one trial is \(\leq \frac{1}{N^{c'}}\), and the probability of failure in all the \(c_2\) trials is thus \(\leq \frac{1}{N^{2c'}}\). Then the probability of success in \(c_2\) trials is \(\geq (1 - \frac{1}{N^{2c'}})\). Therefore, the total run time of the algorithm is \(c_1c_2 f(N)\) with probability \(\geq (1 - \frac{1}{N^{2c'}})\).

**Proof of Theorem 2.4:** (The proof is similar to that in Theorem 2.1, but has different parameters.)

Based on Fact 2.1, to determine the expected delay of a packet \(x\), we only need to determine how many packets \(x'\) are expected to overlap with \(x\). We first determine the probability that \(\rho\) packets overlap \(x\)'s path for the first time in level \(i\). Consider a link, say \(L\), in level \(i\). We know that these \(\rho\) packets can possibly originate from \(d^{i-1}\) number of nodes having \(\ell d^{i-1}\) packets. (Because, initially we have at most \(\ell\) packets in each processor.) Thus, there are \(\binom{\ell d^{i-1}}{\rho}\) number of ways to choose the origins of these \(\rho\) packets. For each packet, there are \(d^{i+1}\) possible paths for the packet to take before it reaches level \(i + 1\). Thus, the probability that each of these \(\rho\) packets pass through link \(L\) is \(\left(\frac{1}{d^{i+1}}\right)\). Besides, the likelihood for the remaining \(\ell d^{i-1} - \rho\) packets not to pass through link \(L\) is \(\left(1 - \frac{1}{d^{i+1}}\right)^{(\ell d^{i-1}) - \rho}\). Hence, we have an upper bound for the probability that the number of packets, whose paths overlap a given path through link \(L\) for the first time at level \(i\),
equals \( \rho \). Let \( d_i \) be number of packets that delay a given packet for the first time in level \( i \). Then,

\[
Prob(d_i = \rho) \leq \left( \frac{\ell d_i - 1}{\rho} \right) \left( \frac{1}{d_i + 1} \right)^\rho \left( 1 - \frac{1}{d_i + 1} \right)^{(\ell d_i - 1) - \rho}
\]

\[
\leq \left( \frac{\ell d_i - 1}{\rho} \right) \left( \frac{1}{d_i + 1} \right)^\rho
\]

\[
\leq \frac{(\ell d_i - 1)^\rho}{\rho!} \frac{1}{(d_i + 1)^\rho}
\]

\[
\leq \frac{1}{\rho!} \left( \frac{\ell}{d^2} \right)^\rho.
\]

But we are interested in the probability of a total delay \( d \) rather than the delay due to packets that meet the given packet for the first time in level \( i \). The total delay for the given packet is \( \sum_i d_i \).

This can be computed using generating functions.

The generating function for \( Prob(d_i = \rho) \) is

\[
G_i(x) = \sum_{\rho=0}^{\infty} \left( \frac{x^\rho}{\rho!} \right) x^\rho = e^{x\frac{d_i - 1}{\rho}}
\]

Based on Fact 2.4, the generating function for \( Prob(\sum_i d_i = \rho) \) is thus given by

\[
G(x) = \prod_{i=1}^{\ell} G_i(x) = e^{\frac{d_i - 1}{\rho!}} = \sum_{\rho=0}^{\infty} \left( \frac{\ell^2}{d^2} \right)^\rho \frac{1}{\rho!}
\]

where \( \ell \) is the number of levels of the network.

Then the probability that the total delay is greater than a given amount, say \( \zeta \), is:

\[
Prob(\sum_i d_i \geq \zeta) \leq \sum_{\rho=\zeta}^{\infty} \left( \frac{\ell^2}{d^2} \right)^\rho \frac{1}{\rho!}
\]

\[
\leq 2 \left( \frac{\ell^2}{d^2} \right) \frac{1}{\zeta!}
\]

\[
\leq 2c_1^2 \frac{1}{\zeta!}, \text{ since } \ell = c_1 d
\]

\[
\leq 2c_1^2 c_2 \frac{1}{c_2! \ell!}, \text{ let } \zeta = c_2 \ell
\]

\[
\leq c_3 c_4 \frac{1}{\ell!}, \text{ where } c_3 = \frac{2}{c_2!} \text{ and } c_4 = c_1^{2 c_2}
\]

\[
\leq c_6 \frac{1}{d^2}, \text{ since } \ell! = (c_1 d!) \geq c_1! c_5 d^\ell, \text{ and let } c_6 = \frac{c_3}{c_1! c_5} \text{ and } c_7 = c_4^{c_1}
\]

\[
\leq c_8 \frac{1}{(d^2)^{c'}}, \text{ where } 0 < c' < 1.
\]

Then it follows from Lemma 2.1 that any \( \ell \)-relation routing can be finished on the leveled networks of \( \ell \) levels in \( c'' c_0 \ell \) steps with probability at least \( 1 - \frac{1}{(d^2)^{c''}} \), \( c'' > 1.\)
Corollary 2.1 For the n-star graph (parallel model) of \(N=n!\) nodes, any partial n-relation routing can be performed by a randomized routing algorithm (using Algorithm 2.2) in \(\tilde{O}(n)\) steps.

Proof: The proof is similar to that of Theorem 2.4. (Replace \(\ell\) and \(d\) by \(c'n\).)

Corollary 2.2 For the n-way shuffle (parallel model) of \(N = n^n\) nodes, any partial n-relation routing can be performed by a randomized routing algorithm (using Algorithm 2.3) in \(\tilde{O}(n)\) steps.

Proof: The same as that of Corollary 2.1.

2.4 Performance Analysis of Emulation

We know that each item of the PRAM has been mapped to a location in distributed memory modules of the emulating network according to a hash function \(h\) randomly chosen from \(H\). To prove that each step of the PRAM can be emulated in desired time, say \(\tilde{O}(\ell)\), we need to prove that each read/write instruction of the PRAM can be performed by the emulating network in \(\tilde{O}(\ell)\) time. First, on the way to access the items, read/write request packets are sent from processors to destinations defined by \(h\). Then, on the way back (in case of a read instruction), each item (a return packet) is sent from its location (destination of the request packet and source of the return packet) to the processor that sent the request packet. The communication algorithm has been analyzed in the previous section. We have proven that if initially there is at most \(c\ell\) packets at any node and no more \(c\ell\) packets have the same destination, the communication could be completed in time \(\tilde{O}(\ell)\). Hence, if we could prove that with extremely high probability no more than \(c\ell\) items in \(S\) will be mapped into any memory module, then together with the result of Theorem 2.4 the desired emulation performance will immediately follow.

Let \(X^S\) be the number of items in \(S\) assigned by the hash function \(h\) to a memory module, then we have:

Lemma 2.2 For \(\gamma > \delta\)

\[
\text{Prob}(\max_S X^S \geq \gamma) \leq N \left(\frac{1}{\gamma - \delta}\right)^\delta.
\]

Proof: (This proof is due to Karlin and Upfal [4]. A summary follows.) Let \(H^S\) be the set of bad functions \(h_b \in H\) mapping at least \(\gamma\) elements in \(S\) to memory module \(\iota\). Also, let \(p^S\) be the probability that a random function \(h \in H\) maps at least \(\gamma\) elements in \(S\) to memory module \(\iota\). We know that for \(1 \leq i \leq \delta\) and \(0 \leq x_i, y_i < P\), there is at most one polynomial \(p\) of degree \(\delta - 1\) over the field \(Z_P\) such that \(p(x_i) = y_i\), for \(i, 1 \leq i \leq \delta\). Thus for each set of \(\delta\) elements \(x_1, x_2, \ldots, x_\delta \in S\) and \(y_1, y_2, \ldots, y_\delta\) with \(y_i = \iota\), there is at most one polynomial \(p\) of degree \(\delta - 1\) over the field \(Z_P\) determined by the set of \(\delta\) \((x_i, y_i)\) pairs. There are \(\binom{N}{\delta}\) choices for \(x_1, x_2, \ldots, x_\delta\) and \(\binom{P}{\delta}\) choices for \(y_1, y_2, \ldots, y_\delta\). Hence, the maximum possible number of different polynomials which allow this is bounded by \(\binom{N}{\delta} \left(\frac{P}{N}\right)^\delta\). For each \(h_b \in H\), it has a set of \(\gamma > \delta\) \((x_i, y_i)\) pairs such that
\[ y_i = \iota. \] Each set of \( \gamma \) points has \( \binom{\gamma}{d} \) different subsets of \( \delta (x_i, y_i) \) pairs and each subset uniquely determines the same function \( h_b \). Hence, we have

\[
\binom{\gamma}{d} |H_S| \leq \binom{N}{\delta} \left( \frac{P}{N} \right)^\delta.
\]

Since \(|H| = P^\delta \) and \( p^b_S = \frac{|H_S|}{|H|} \), we have \( p^b_S \leq \frac{\binom{N}{\delta}}{\binom{\gamma}{\delta}} \left( \frac{1}{N} \right)^\delta \leq \left( \frac{1}{\gamma^\delta} \right)^\delta \)

for all \( S \). The lemma immediately follows from the fact that

\[
\text{Prob}(\max_S X^S \geq \gamma) \leq \sum_{i=1}^{N} \max_S p^b_S. \quad \Box
\]

**Theorem 2.5** 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.

**Proof:** Using Lemma 2.2, and fixing \( \delta \) to be \( c\ell \), the probability that more than \( c\ell \) elements from any \( S \) are assigned to a memory module is bounded by \( \frac{1}{N^c} \). Together with Theorem 2.4, the theorem is proven. \( \Box \)

**Corollary 2.3** Each step of the EREW PRAM can be emulated by the \( n \)-star graph (parallel model) of \( N = n! \) nodes in \( \tilde{O}(n) \) steps.

**Proof:** Using Lemma 2.2, and fixing \( \delta \) to be \( cn \), the probability that more than \( cn \) elements from any \( S \) are assigned to a memory module is bounded by \( \frac{1}{N^c} \). Together with Corollary 2.1, the theorem is proven. \( \Box \)

**Corollary 2.4** Each step of the EREW PRAM can be emulated by the \( n \)-way shuffle (parallel model) of \( N = n^n \) nodes in \( \tilde{O}(n) \) steps.

**Proof:** The same as that of Corollary 2.3. \( \Box \)

**Theorem 2.6** Each step of the CRCW PRAM can be emulated by a leveled network of \( \ell \) levels with degree \( d \), \( \ell = O(d) \), in \( \tilde{O}(\ell) \) steps.

**Proof:** Combining all incoming packets having the same destination into one packet\(^3\) and storing \( \log d \) direction bits before sending this packet out, together with the proof of Theorem 2.5, the theorem is proven. The purpose of storing \( \log d \) direction bits is to make sure each requesting processor receives a reply. \( \Box \)

**Corollary 2.5** Each step of the CRCW PRAM can be emulated by the \( n \)-star graph (parallel model) of \( N = n! \) nodes in \( \tilde{O}(n) \) steps.

\(^3\)It is assumed that any number of incoming packets, which have the same destination, from different links can be combined into one packet in one unit time.
Proof: Immediately follows from Theorem 2.6 by replacing \( \ell \) and \( d \) by \( cn. \)

**Corollary 2.6** Each step of the CRCW PRAM can be emulated by the \( n \)-way shuffle (parallel model) of \( N = n^n \) nodes in \( \tilde{O}(n) \) steps.

**Proof:** The same as that of Corollary 2.5.\( \Box \)

### 3 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. hereafter). 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.

#### 3.1 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., [19], [6], [8], [9], [12]). In a single step each processor can perform a local computation (like a comparison) and also communicate with all its (4 or less) neighbors.

#### 3.2 Preliminaries

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

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 [19] 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 [6] who presented a \( 2n + O(\log n) \) time \( O(1) \) queue randomized algorithm and Kunde [8] 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 [9] displayed a \( 2n - 2 \) time algorithm with a
queue size of roughly 672. Recently, Rajasekaran and Overholt [12] have presented a $2n - 2$ step algorithm with a queue size of only 58.

### 3.3 The Algorithm

We restrict our attention to emulating an EREW PRAM. As has been pointed out before such an algorithm can also be used to emulate a CRCW PRAM with an additional trick of 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 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 above stated 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 [19]'s and [6]'s) will imply that w.h.p. the above emulation procedure will terminate within 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 [4] 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 and will be broadcast to every other processor.

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 [13] 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 (defined in section 2.1). 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. This can be proven using the following fact.

**Fact [4].** 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:

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

The following three corollaries easily follow and will be used in our analysis.
Corollary 3.1 If $N$ items are mapped into $N$ buckets, no single bucket will get more than $O\left(\frac{\log N}{\log \log N}\right)$ items with probability $\geq (1 - N^{-\alpha})$.

Corollary 3.2 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 $\frac{n^3}{\beta} + O(n^{3/4})$ w.h.p. (i.e., $\geq (1 - n^{-\alpha})$, for any constant $\alpha \geq 2$).

Corollary 3.3 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 $O(\log N)$ w.h.p.

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

3.4 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 [6], 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 $(k, l)$.

stage 1

$\pi$ 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 $j$.

stage 2
\(\pi\) traverses along row row \(i'\) to the node \(i', l\).

**stage3**

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

### 3.4.1 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 queueline lemma [19], 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 \(cn + o(n)\). (Realizing that the number of packets originating from any column slice is no more than \(cn + o(n)\) w.h.p. (see corollary 3.2)). 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 3.2 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 3.1, we can also prove a queue size of \(O(\log n)\).

This proves the following

**Theorem 3.1** 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 3.2** 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 3.3. The improvement will parallel the \(2n + O(\log n)\) time routing time algorithm presented in [6], with a slightly different analysis. In similar lines we can also prove the following

**Theorem 3.3** 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.
4 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


