July 1992

A Comparison of Meshes With Static Buses and Unidirectional Wrap-Arounds

Danny Krizanc
University of Rochester

Sanguthevar Rajasekaran
University of Pennsylvania

Sunil M. Shende
University of Nebraska

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

Recommended Citation


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/505
For more information, please contact repository@pobox.upenn.edu.
A Comparison of Meshes With Static Buses and Unidirectional Wrap-Arounds

Abstract
We investigate the relative computational powers of a mesh with static buses and a mesh with unidirectional wraparounds. A mesh with unidirectional wraparounds is a torus with the restriction that any wraparoundlink of the architecture can only transmit data in one of the two directions at any clock tick. We show that the problem of packet routing can be solved as efficiently on a linear array with unidirectional wraparound link as on a linear array with a broadcast bus. We also present a routing algorithm for a twodimensional torus with unidirectional wraparound links whose run time is close to that of the best known algorithm for routing on a mesh with broadcast buses in each dimension. In addition, we show that on a mesh with broadcast buses, sorting can be done in time that is essentially the same as the time needed for packet routing.

Keywords
mesh connected computer, packet routing, sorting, broadcast bus, randomized algorithms

Comments
A Comparison Of Meshes With Static Buses and Undirectional Wrap-Arounds

MS-CIS-92-58
GRASP LAB 325

Danny Krizanc
(University of Rochester)

Sanguthevar Rajasekaran
(University of Pennsylvania)

Sunil M. Shende
(University of Nebraska)

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

July 1992
A COMPARISON OF MESHES WITH STATIC BUSES AND UNIDIRECTIONAL WRAP-AROUNDS

DANNY KRIZANC
Department of Computer Science, University of Rochester
Rochester, NY 14627, U.S.A.

and

SANGUTHEVAR RAJASEKARAN
Department of Computer and Information Sciences, University of Pennsylvania
Philadelphia, PA 19104, U.S.A.

and

SUNIL M. SHENDE
Department of Computer Science and Engineering, University of Nebraska
Lincoln, NE 68588, U.S.A

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT
We investigate the relative computational powers of a mesh with static buses and a mesh with unidirectional wrap-arounds. A mesh with unidirectional wrap-arounds is a torus with the restriction that any wrap-around link of the architecture can only transmit data in one of the two directions at any clock tick. We show that the problem of packet routing can be solved as efficiently on a linear array with unidirectional wrap-around link as on a linear array with a broadcast bus. We also present a routing algorithm for a two-dimensional torus with unidirectional wrap-around links whose run time is close to that of the best known algorithm for routing on a mesh with broadcast buses in each dimension. In addition, we show that on a mesh with broadcast buses, sorting can be done in time that is essentially the same as the time needed for packet routing.

Keywords: Mesh connected computer, Packet Routing, Sorting, Broadcast Bus, Randomized Algorithms

1. Introduction

1.1. Routing and Sorting
There is an extensive body of literature pertaining to sorting and routing prob-
lems on general r-dimensional meshes; see, for example, [11, 6, 4]. Extensions of the basic architecture including those with added broadcast buses have also been studied; see, for example, [3, 13].

In the packet routing problem, a packet of information consists of some data (irrelevant to the problem) and the destination index (or indices) of some processor(s) in the architecture. In this paper, we shall mainly consider permutation routing, viz. there is at most one packet at each processor initially, and each processor may be the destination of no more than one packet. A more general case of permutation routing is one-to-one routing, where each packet is destined for only one node and each node is the destination of no more than one packet and the origins of the packets could be arbitrary. Regardless of the specific kind of routing problem, we shall always assume that the total number of packets is at most the number of processors in the mesh.

The problem of sorting on a two dimensional mesh can be described as follows. There is a key at each node in the mesh, and the task is to rearrange the keys in ascending order according to some indexing scheme. The indexing scheme assumed in this paper is the blockwise snakelike row major indexing (the same as the one assumed in previous works such as [4, 1, 9]).

1.2. The parallel models

Our basic model is a classical mesh, denoted \( M \), both one-dimensional (a linear array with \( n \) processors) and two-dimensional (an \( n \times n \) square grid with no wrap-around connections). A two-dimensional mesh contains a processor at each grid point, and every processor is connected to its four (or less) neighbors in the grid via bidirectional links. In every time step, it is assumed that each processor can communicate with any number of its neighbors (this model is referred to as the MIMD model in the literature) by sending single packets across the corresponding (directed) links. Further, each processor is capable of storing a certain number of packets. For any algorithm on the mesh, the maximum such storage required during any time step is called the queue size of the algorithm.

The mesh with static buses, denoted as \( M_b \), is obtained from the classical mesh by augmenting the latter with exclusive-write, concurrent-read broadcast buses along each dimension. Thus, the linear array with a bus contains a single bus that connects every processor in the array. The two-dimensional mesh with buses contains \( 2n \) buses, one for each row and column of the mesh. At any time instant, at most one processor attached to some given bus may write an information packet on the bus; it can be read by every other processor attached to that bus after one time step, which is the latency or delay of the bus.

Our final model is the classical mesh with unidirectional, wrap-around links, denoted as \( M_w \). This model is identical to the torus, except that the wrap-around links are constrained to be unidirectional, i.e., at any given clock tick, at most one of the endpoints of any such link is able to send a packet along the link. The wrapped distance from a processor \( i \) to processor \( j \) in the linear array \( M_w \) is given
by \(|n - j - i|\) for \(1 \leq i, j \leq n\), and denotes the distance along a path from \(i\) to \(j\) that traverses the wrap-around link.

2. Routing on the linear array \(M_w\)

The following tight bound on routing on the linear array \(M_b\) can be established.

**Lemma 1 ([7])** On the linear array \(M_b\) with \(n\) processors, one-to-one routing can be performed in \(2n/3\) steps.

We first derive a lower bound for one-to-one routing on the linear array \(M_w\). Consider the (partial) permutation routing problem in which every processor \(i\) \((1 \leq i \leq n/3)\) initially contains a single packet destined for the processor \((2n/3 + i)\) in the array, and vice versa. In particular, a total of \(2n/3\) processors contain packets, with each packet at a distance of \(2n/3\) from its destination.

If some packet \textit{never} uses the wrap-around link during the course of routing, it will need at least \(2n/3\) time steps to reach its destination. On the other hand, if all the \(2n/3\) packets use the wrap-around link, the unidirectional communication restriction implies that at least \(2n/3\) distinct time steps will be needed to deliver the packets across the link. Consequently:

**Lemma 2** On the linear array \(M_w\) with \(n\) processors, one-to-one routing takes at least \(2n/3\) time steps.

It is obvious that this lower bound argument can be extended to general one-to-one routing problems on a mesh \(M_w\) in any number of dimensions. That the upper bound for permutation routing on the linear array \(M_w\) is also \(2n/3\) time steps, is less obvious.

Consider a linear array \(M_w\) and partition the \(n\) processors of \(M_w\) into three equal contiguous segments with \(n/3\) processors in each segment. In particular, let \(A\) denote the segment of processors \(\{i : 1 \leq i \leq n/3\}\), let \(B\) denote the segment \(\{i : n/3 + 1 \leq i \leq 2n/3\}\), and let \(C\) denote the remaining segment. We shall classify the packets in \(M\), based on their origins and their final destinations in the array. Let \(X_Y\) denote the set of packets initially in segment \(X\) (\(X \in \{A, B, C\}\)) with destinations in segment \(Y\) (\(Y \in \{A, B, C\}\)).

The algorithm is divided into two phases with each phase executing for exactly \(n/3\) time steps. In phase I, packets are forwarded as follows:

- \(C_A\) packets move to processor \(n\), traverse the wrap-around link to processor 1, and then use the remaining time steps in phase I to progress towards their destinations in region \(A\).
- \(A_C\) packets undergo \textit{rearrangement} within region \(A\). In particular, an \(A_C\) packet destined for processor \(2n/3 + k\) \((1 \leq k \leq n/3)\) is sent to the intermediate processor \(k\).
- The remaining packets proceed along shortest paths toward their destinations, without using the wrap-around link.

In phase II, we continue routing as follows:

- \(C_A\) packets continue towards their destinations (now within region \(A\)).
• The rearranged $A_C$ packets move to processor 1, traverse the wrap-around link to processor $n$, and then proceed toward their destinations in region $C$.
• $A_B$, $B_A$, $B_C$ and $C_B$ packets continue towards their respective destinations, with $B_A$ packets yielding priority to the rearranged $A_C$ packets in the event of edge contention.

Lemma 3 The foregoing algorithm correctly routes any permutation routing problem on the linear array $M_w$ with $n$ processors in exactly $2n/3$ time steps with queue size at most 3.

Proof: First, it is clear that the wrap-around link is only used unidirectionally in each phase, by $C_A$ packets in phase I and by the rearranged $A_C$ packets in phase II. Since every processor initially contains at most one packet, the queue size requirement at any processor is 2, except processors in segment $A$ which may need to store an additional $A_C$ packet.

Observe that all packets can move without delays in phase I. Hence, $A_A$, $B_B$ and $C_C$ packets complete their routing within the phase. Similarly, in phase II, $A_C$, $C_A$, $A_B$, $B_C$ and $C_B$ packets suffer no delays and hence reach their respective destinations by the end of phase II. Therefore, it only remains to show that every $B_A$ packet that has not reached its destination by the end of phase I, completes its routing by the end of phase II.

Consider, without loss of generality, such a $B_A$ packet which originates at processor $(n/3 + i)$ and is destined for processor $j$, for some $1 \leq j < i \leq n/3$. It reaches processor $i$ at the end of phase I without yet reaching its final destination. However, in phase II, the packet under consideration may be delayed, but only by at most $(n/3 - i + 1)$ rearranged $A_B$ to its “right” in region $A$. Hence, counting steps after the completion of phase I, the packet finishes routing in $(n/3 - i + 1) + (i - j) = n/3 - (j - 1)$ additional time steps. Since $j \geq 1$, the packet completes its routing within phase II.

As a consequence, we obtain the following intriguing similarity in routing complexity between the linear array models $M_b$ and $M_w$.

Theorem 1 Permutation routing on both of the linear array models $M_w$ and $M_b$ takes $\Theta(2n/3)$ time steps.

The general version of one-to-one routing problem seems distinctly harder to solve on the linear array model $M_w$ as compared to $M_b$. The latter model can perform one-to-one routing in $2n/3$ time steps [8] whereas it is still open whether $M_w$ can perform one-to-one routing any faster than the classical linear array without wrap-around, viz. in less than $n$ time steps. Consider the following routing problem $P'$ on the linear array $M_w$. We have $n/2$ packets originating one per processor in processors 1 through $n/2$, and destined for arbitrary processors in the array (with possibly many packets destined for the same processor).

The special nature of the problem $P'$ makes it possible to perform the routing in $n/2$ steps on $M_w$. Specifically, packets that are at a distance of less than $n/2$ from their destinations are routed there along the shortest path, while the remaining packets are simultaneously routed one by one through the unidirectional wrap-around link in $n/2$ steps (since their wrapped destination distances are at most
n/2). Using this observation, it is possible to adapt the algorithm for permutation routing on the two-dimensional model $M_b$ as follows.

Briefly, the algorithm on $M_b$ [7] consists of three phases:

(i) Partition the mesh into blocks of size $\frac{n}{q}$ for $2 \leq q \leq n$ and sort packets in each block by destination index in snake-like row-major order.

(ii) Each column (resp. row) in the top-left and the bottom-right quadrants (resp. the top-right and the bottom-left quadrants) independently performs the routing given by problem $P'$, with the packets being sent to their correct destination row within the column (resp. the correct destination column within the row).

(iii) Each column (resp. row) in the mesh now performs a one-to-one routing of packets that arrived in the column (resp. row) in the previous phase, with packets being sent to their final row (resp. column) destinations.

For details of the complete algorithm on $M_b$, see [7]. The algorithm correctly routes all the packets in $7n/6 + O(\frac{n}{q})$ steps using a queue size of $O(q)$. A slightly improved routing algorithm has also been obtained [8], with runtime $n + o(n)$ and queue size $o(n)$.

On $M_w$, we perform the same three phases described above, using the unidirectional wrap-around links instead of buses. Phase (i) can be implemented on $M_w$ in $O(\frac{n}{q})$ steps, phase (ii) (by the observation above) in $n/2$ steps, and phase (iii) in $n$ steps. The algorithm uses intermediate queues of size $O(q)$, and its proof of correctness is identical to that for the algorithm on $M_b$ (see [7]).

**Theorem 2** The permutation routing problem on the two-dimensional mesh $M_w$ can be solved in $3n/2 + O(\frac{n}{q})$ time steps with queue size $O(q)$ for any $2 \leq q \leq n$.

### 3. Randomized Sorting with Static Buses

We show here that sorting of $n^2$ elements can be accomplished on an $n \times n$ mesh with fixed buses in time that is only $o(n)$ more than the time needed for permutation routing with high probability (abbreviated as w.h.p. from hereon). If one employs the improved routing algorithm of Leung and Shende [8] the run time for sorting will be $n + O(\frac{n}{q})$ steps w.h.p., the queue size being $O(q)$ (for any $2 \leq q \leq n$).

Many optimal algorithms have been proposed in the literature for sorting on the conventional mesh (see e.g. [5]). A $2n + o(n)$ step randomized algorithm has been discovered for sorting by Kaklamanis and Krizanc [1]. But $2n - 2$ is a lower bound for sorting on the conventional mesh. Recently Rajasekaran and McKendall [10] have presented an $n + o(n)$ randomized algorithm for routing on a reconfigurable mesh.

**Summary.** Random sampling has played a vital role in the design of parallel algorithms for comparison problems (including sorting and selection). Reischuk's [12] sorting algorithm is a good example. Given $n$ keys, the idea is to: 1) randomly sample $n^\epsilon$ (for some constant $\epsilon < 1$) keys, 2) sort this sample (using any nonoptimal algorithm), 3) partition the input using the sorted sample as splitter keys, and 4) to
sort each part separately in parallel. Similar ideas have been used in many other works as well (see e.g., [2, 1, 9, 10]).

Let $X = k_1, k_2, \ldots, k_n$ be a given sequence of $n$ keys and let $S = \{k'_1, k'_2, \ldots, k'_s\}$ be a random sample of $s$ keys (in sorted order) picked from $X$. $X$ is partitioned into $(s+1)$ parts defined as follows. $X_1 = \{\ell \in X : \ell \leq k'_1\}$, $X_j = \{\ell \in X : k'_{j-1} < \ell \leq k'_j\}$ for $2 \leq j \leq s$, and $X_{s+1} = \{\ell \in X : \ell > k'_s\}$. The following lemma [12] probabilistically bounds the size of each of these subsets, and will prove helpful to our algorithm. (We say a function $f(n)$ is $O(g(n))$ if $f(n)$ is $\leq c g(n)$ for all large $n$ and for some constant $c$ with probability $\geq (1 - n^{-2})$.

Lemma 4 The cardinality of each $X_j$ $(1 \leq j \leq (s+1))$ is $O(\frac{n}{\log n})$.

Next we describe our algorithm and prove its time bound. This algorithm is similar to the one given in [10]. We only provide a brief summary of the algorithm. More details can be found in [2] or [10]. The mesh is partitioned into blocks of size $n^{4/5} \times n^{4/5}$.

(i) A random sample of size very nearly $n^{3/5}$ is chosen and broadcast to the whole mesh, such that each block stores a copy of all the splitter keys.

(ii) We compute the partial ranks of the sample keys in each block after sorting the block.

(iii) Then we perform a prefix sum operation on these partial ranks so as to obtain the global ranks of the sample keys.

(iv) Now we route each packet to an approximate destination that is a random node in an appropriate block of size $n^{3/4} \times n^{3/4}$. This approximate destination is very close to its actual destination and depends on the two splitter keys between which it falls. In particular, the approximate destination of any packet will be at the most a block away from its actual destination w.h.p.

(v) Next we sort the individual blocks and compute the rank of each key in the mesh.

(vi) Finally we route the packets to their actual destinations.

Analysis. The key to the analysis is the observation that the global ranks of the sample keys can be computed in $o(n)$ steps. This observation was first made in [10] in connection with sorting on a reconfigurable mesh.

Step (i) takes $O(n^{3/5})$ steps, since a single key can be broadcast to the whole mesh in $O(1)$ steps using the buses.

Step (ii) involves sorting blocks of size $n^{3/4} \times n^{3/4}$ (together with the sample keys) and can be completed in $O(n^{3/4})$ time using any standard sorting algorithm (such as Schnorr and Shamir's [5]).

In step (iii), the global rank of a single key can be computed in time $O(n^{1/5})$. This can be done for instance by concentrating all the partial ranks of this key in a region of size $n^{1/5} \times n^{1/5}$. Thus the global ranks of all the keys can be determined in time $O(n^{1/5} \times n^{3/5}) = O(n^{4/5})$.

In step (iv), routing takes $n + o(n)$ steps using Leung and Shende's algorithm [8].
Sorting in step (v) takes $O(n^{4/5})$ time.
Step (vi) also can be finished in time $O(n^{4/5})$ because the actual destination of any key can be at the most one block away from where it is after step 4 (cf. lemma 4).
Thus we have the following

**Theorem 3** Sorting on an $n \times n$ mesh with buses can be performed in $n + o(n)$ steps w.h.p.

### 4. Open Problems

We have shown that the parallel model $M_b$ can perform permutation routing as efficiently as the model $M_b$ in the linear case. We conjecture that the same holds for general one-to-one routing. It is also open whether the randomized sorting result for $M_b$ (Theorem 3) also extends to the model $M_w$.

### References

10. S. Rajasekaran and T. McKendall, Randomized Routing and Sorting on the Re-
configurable Mesh, Technical Report MS-CIS-92-36, University of Pennsylvania,

Processor Arrays, to appear in Algorithmica.

12. R. Reischuk, Probabilistic Parallel Algorithms for Sorting and Selection, SIAM Jour-

13. Q. F. Stout, Meshes with Multiple Buses, in 27th IEEE Symp. on Foundations of