November 1990

Constant Queue Route on a Mesh

Sanguthevar Rajasekaran
University of Pennsylvania

Richard Overholt
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-25.

Constant Queue Route on a Mesh

Abstract

Packet routing is an important problem in parallel computation since a single step of inter-processor communication can be thought of as a packet routing task. In this paper we present an optimal algorithm for packet routing on a mesh-connected computer.

Two important criteria for judging a routing algorithm will be 1) its *run time*, i.e., the number of parallel steps it takes for the last packet to reach its destination, and 2) its *queue size*, i.e., the maximum number of packets that any node will have to store at any time during routing. We present a $2n - 2$ step routing algorithm for an $n \times n$ mesh that requires a queue size of only 58.

The previous best known result is a routing algorithm with the same time bound but with a queue size of 672. The time bound of $2n - 2$ is optimal. A queue size of 672 is rather large for practical use. We believe that the queue size of our algorithm is practical. The improvement in the queue size is possible due to (from among other things) a new $3s + o(s)$ sorting algorithm for an $s \times s$ mesh.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-25.

Constant Queue Routing On A Mesh

MS-CIS-90-25
GRASP LAB 211

Sanguthevar Rajasekaran
Richard Overholt

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

Revised
November 1990

Constant Queue Routing on a Mesh

Sanguthevar Rajasekaran
Richard Overholt

Dept. of Computer and Information Science
Univ. of Pennsylvania, Philadelphia, PA 19104

ABSTRACT

Packet routing is an important problem in parallel computation since a single step of inter-processor communication can be thought of as a packet routing task. In this paper we present an optimal algorithm for packet routing on a mesh-connected computer.

Two important criteria for judging a routing algorithm will be 1) its *run time*, i.e., the number of parallel steps it takes for the last packet to reach its destination, and 2) its *queue size*, i.e., the maximum number of packets that any node will have to store at any time during routing. We present a $2n - 2$ step routing algorithm for an $n \times n$ mesh that requires a queue size of only 58.

The previous best known result is a routing algorithm with the same time bound but with a queue size of 672. The time bound of $2n - 2$ is optimal. A queue size of 672 is rather large for practical use. We believe that the queue size of our algorithm is practical. The improvement in the queue size is possible due to (from among other things) a new $3s + o(s)$ sorting algorithm for an $s \times s$ mesh.

1 Introduction

The design of efficient packet routing algorithms on fixed connection machines has two important consequences: 1) they lead to faster inter-processor communication, and 2) ideal parallel machines can be efficiently simulated on the fixed connection machines.

Even though asymptotically optical algorithms have been designed for routing on networks with 'small' diameter (see e.g., [10, 9, 5, 6]), a direct application of these algorithms on an $(n \times n)$ mesh connected computer does not seem to yield an optimal run time (i.e., $2n - 2$ steps). Thus, people have designed algorithms that are specific to the mesh.

Some of the relevant previous results include 1) a $3n + o(n)$ steps and $\theta(\log n)$ queue size randomized algorithm of Valiant and Brebner [10], 2) a $2n + \theta(\log n)$ randomized constant queue routing algorithm due to Krizanc, Rajasekaran, and Tsantilas [1], and 3) a $2n+\theta(n/q)$ (for any $1 \leq q \leq n$) routing algorithm with a queue size of q due to Kunde [3]. Leighton, Makedon, and Tollis [4] subsequently presented an optimal algorithm that takes $2n-2$ steps, and needs a queue size of 672. A queue size of 672 is rather large.

In this paper we present a $2n-2$ step deterministic routing algorithm that needs a queue size of only 58. We believe this queue size is practical. Our improvement in the queue size is possible due to (from among other things), a new $3s + o(s)$ sorting algorithm on an $s \times s$ mesh. This paper heavily borrows ideas from [4].

In section 2 we make some definitions. In section 3 we show that sorting on an $s \times s$ mesh can be performed in $3s + o(s)$ steps. In section 4 we prove some facts connected with routing on a linear array. These results are crucial to our algorithm. In sections 5 and 6 we present our $2n - 2$ step routing algorithm and prove an upper bound of 58 for the queue size. Finally in section 7 we provide some concluding remarks.

2 Machine Model and Problem Statement

An $n \times n$ mesh is a collection of n^2 processors arranged in a square grid of size $n \times n$, one per grid point. The grid edges correspond to communication links. We assume all the links are bidirectional, and each processor can communicate with all its (four or less) neighbors in one time step. This model is called the MIMD mesh and has been assumed in all the previous works cited [1, 3, 4].

The problem of *routing* is defined as follows. Each processor has a packet of information that it wants to send to some other processor. Send all the packets to their destinations such that only one packet traverses along any edge at any time, and all the packets reach their destinations quickly. If at the most one packet originates from out of any node and at the most one packet is destined for any node, the problem of routing is called *partial permutation routing.* The algorithm we present is for routing partial permutations.

A *routing function* is a rule that specifies a path for each packet, together with a *queue discipline.* By queue discipline we mean a rule for resolving contentions for the same edge.

Two popular queue disciplines are:

$\mathcal{Q}$: The packet with the furthest origin gets priority.

$\mathcal{Q}'$: The packet with the furthest destination gets priority.

3 A $3s$ Sorting Algorithm for an $s \times s$ Mesh

One of the bottlenecks in Kunde's [3] routing algorithm is the need to perform sorting of small subsquares. [4] assume a $4s + o(s)$ algorithm for sorting an $s \times s$ mesh. A reduction in this sorting time is immediately reflected as a proportionate reduction in the queue size of [4]'s routing algorithm. In this section we show sorting can be done in $3s + o(s)$ steps on an $s \times s$ mesh. Since we require sorting to be performed in column major order, the $3s$ sorting lower bound of Kunde [2] and Schnorr & Shamir [7] holds. And hence as for as sorting subsquares is concerned, this is the best we can do.

The Algorithm

We make use of Thompson and Kung's [8] sorting algorithm. Thompson and Kung's algorithm in turn is an implementation of (a generalized) odd-even merge algorithm. The algorithm assumes a snake-like column major indexing scheme (Our routing algorithm also makes use of the same indexing scheme).

The algorithm runs as follows. The mesh is partitioned into subsquares of size $k \times k$ (where $k = s^{2/3}$), and each subsquare is sorted independently in parallel in $O(k)$ time. We are left with $s^2/k^2 = s^{2/3}$ sorted lists. These are merged recursively using a generalized odd-even (in fact an $s^{2/3}$–way) merge algorithm.

The key steps of this algorithm are the following. 1) Packets are 'unshuffled' in stages for a total of s steps; 2) the base case is performed; 3) packets are 'shuffled' for a total of s steps; 4) at this point each packet will be at the most $s^{2/3} - 1$ distance away from its final destination; $s^{2/3} - 1$ steps of the *odd-even transposition algorithm* [8] are applied to send each packet to its correct destination. The base case of the algorithm is to merge $s^{2/3}$ sorted lists in the **same** row. This can be accomplished by sorting the row. A row (of length s) can be sorted using s steps of the odd-even transposition algorithm. Each step of the odd-even transposition algorithm is a compare-exchange step (i.e., each node compares its key with its neighbor's and swaps the key if they are out of order). On the MIMD model we assume in this paper, a compare exchange step can be performed in 2 time units. If A and B are neighbors, A sends its packet in one time unit to B. B compares the keys of the packet it received and its own; the packet with a smaller key is sent back to A. In our model B can do these in one time unit.

A compare exchange can be performed in one step as follows on the MIMD model if we are allowed to copy and destroy packets. A sends a copy of its packet to B and at the same

time B sends a copy of its packet to A. In the same time step, both the nodes make a comparison and throw away the unwanted packet. But the assumption that one can copy and destroy **packets** may not be valid. We show that even if we are not allowed to copy or destroy packets, we can sort a row of length s in s steps. The only assumption we make is that **keys** (i.e., addresses) of packets can be copied and destroyed.

An s step row sorting algorithm.

If A and B are two neighboring nodes that want to perform a compare exchange, they each send a copy of their packet's key to the other at the same time. In the same time unit both perform a comparison and choose an appropriate key. It may be the case that the packets will have to be swapped. Any such swapping is done in the next time unit. Since in the odd-even transposition algorithm at any given time only half the number of edges are used to perform compare exchanges, packets swapping can be done using the other half of the edges, with a net additional delay (over the usual odd-even transposition algorithm) of only one step for the whole algorithm.

Details of the algorithm follow. Each node at any given time has two objects, namely, a packet and a key (i.e., a destination address). At the beginning, the packet each processor has is simply the one that originates there, and the key it has is the destination address of its packet.

During odd numbered cycles for all i $(0 \leq i \leq \lfloor s/2 \rfloor - 1)$ in parallel do

> Processor $2i + 1$ sends its packet left if the packet's destination is not the same as its key, and at the same time sends a copy of its key to its right neighbor. In the same time unit it receives a copy of its right neighbor's key; compares the two keys and stores the smaller.
>
> Processor $2i + 2$ sends its packet to the right if the packet's address does not match its key; it sends a copy of its key to its left neighbor; receives a key from left; compares the keys and keeps the larger.

During even cycles for all i $(0 \leq i \leq \lfloor (s-1)/2 \rfloor - 1)$ do

> Processors $2i + 2$ and $2i + 3$ perform exchanges mentioned above.

Lemma 3.1 *The above algorithm sorts a row of length s in s steps on the MIMD model.*

Proof. The keys of packets follow exactly the odd-even transposition algorithm and hence the keys will be in sorted order in $(s-1)$ steps [8]. Also, at any given time unit, each packet is at the most one distance away from its key. Thus the result follows. (An example is given in figure 7.) □

The above lemma together with the summary given for Thompson and Kung's algorithm imply the following

Theorem 3.1 *Sorting on an $s \times s$ mesh can be performed in $T_s = 3s + o(s)$ steps under the MIMD model.*

4 Routing on a Linear Array

Consider the following routing problem. Node i $(1 \leq i \leq n)$ of a linear $n-$array has $k_i \geq 0$ packets initially and each node is the destination of exactly one packet. The following lemma has been proven in [1]:

Lemma 4.1 *If the queue discipline used is 'furthest destination first', then the time needed for a packet starting at node i to reach its destination is no more than the distance between i and the boundary in the direction the packet is moving. That is, this time is no more than i or $(n-i)$ (depending on whether it is moving from right to left or left to right respectively).*

Proof. A summary of this proof is needed in order to understand the subsequent lemmas.

Consider a packet q at node i with j as its destination. W.l.o.g., let q move from left to right. q can only be delayed by packets with destinations $> j$ and which are to the left of their destinations. Let $k'_1, k'_2, \ldots, k'_n$ be the number of such packets at nodes $1, 2, \ldots, n$ respectively (at the beginning). (Note that $\sum_{l=1}^{n} k'_l \leq n - j - 1$).

Let m be such that $k'_{m-1} > 1$ and $k'_j \leq 1$ for $m \leq j \leq n$. Call the sequence $k'_m, k'_{m+1}, \ldots, k'_n$ the *free sequence*. Realize that a packet in the free sequence will not be delayed by any other packet in the future. Also, at every time step at least one new packet joins the free sequence. Thus after $(n-j-1)$ steps, all the packets that can possibly delay q would have joined the free sequence. q needs only an additional $j-i$ steps or less to reach its destination. The case in which the packet moves from right to left is similar. □

As a consequence of the previous lemma we have the following:

Lemma 4.2 *After m steps from the beginning of routing on an $n-$node linear array, the distance of any packet to its final destination can not be greater than $n-m$.*

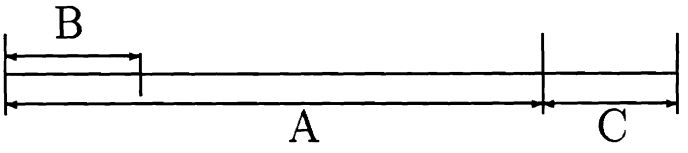

Figure 1:

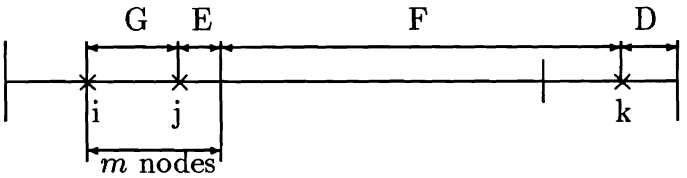

Figure 2:

The next two lemmas are crucial to our algorithm. The proof of the first one is similar to that of lemma 4.1 and hence is given without a proof.

Lemma 4.3 *Let q be a packet that is currently at node i whose destination is to the right of i. Let k_j be the number of packets that can potentially delay q and which are currently in node j (for $1 \leq j \leq n$). Then, the maximum delay the packet can suffer (in addition to the distance to its destination), is $\leq \sum_{j=1}^{i} k_j + \sum_{j=i+1}^{n} (k_j - 1)$.*

Lemma 4.4 *Let m be any integer $\leq \frac{n}{2}$. During the first m time steps, say we do not process any packet that 1) originates in the rightmost m nodes and whose destination is to the right of its origin, or 2) originates in the leftmost m nodes and whose destination is to the left of its origin. If queue discipline $\mathcal{Q}'$ is used, routing can be completed in n steps in the worst case.*

Proof. Let B stand for the set of left most m nodes, and C stand for the set of right most m nodes in the linear array. Also let A stand for all the nodes in the array except the ones in C. (see figure 1). Consider a packet q that is moving from left to right. q is unaffected by the constraint in region B. There are three cases.

Case 1. q starts in region C. The packet doesn't move during the first m steps. After these m steps, applying lemma 4.1, q needs an additional m steps or less to reach its destination. In all q will be done in $2m(\leq n)$ steps or less from the start.

Case 2. q starts in region A and has a destination in region A. Realize that the schedule of packets that originate in region A remains unchanged (from the case there is no constraint

in region C), until they leave region A. Since q has a destination within region A, the time it takes to reach its destination remains unaltered. This time is $\leq n$ applying lemma 4.1.

Case 3. The packet starts in region A and has a destination in region C. Let i, j, k be the origin, position after m steps, and the destination of q respectively. q can only be delayed by packets with destinations in region D (see figure 2; in figure 2, G stands for all the nodes to the right of i up to and including node j, and E stands for the next $m-(j-i+1)$ nodes. Regions F and D are also defined in a similar manner). After m steps from the start, all the packets with a destination address $\geq k$ and which originated in region A will be in a free sequence (see lemma 4.1). This in particular means that all such packets will be found at the most one per node. The number of such packets in the free sequence that are to the right of node j is $\geq m - |G| = |E|$.

Therefore, using lemma 4.3, q needs an additional time (after the first m steps) of $\leq |E| + |F| + (|D| - |E|) = |D| + |F| \leq (n - m)$. Thus, q will be done in $\leq n$ steps.

The case of the packet moving from right to left is similar. □

5 The Routing Algorithm

Before we give the details of the algorithm, a few definitions are in order.

The mesh is partitioned into 64 subsquares as shown in figure 3. Let 'column block j' (for $1 \leq j \leq 8$) correspond to columns $(j-1)n/8 + 1$ through $jn/8$, and 'row block i' (for $1 \leq i \leq 8$) correspond to rows $(i-1)n/8 + 1$ through $in/8$. By 'supersquare $< i, j >$' we mean the intersection of row block i with column block j. The size of any supersquare is $m \times m$ where $m = n/8$. Supersquares $< 1,1 >, < 1,8 >, < 8,1 >$, and $< 8,8 >$ will be called 'corner supersquares'.

The algorithm to be described is a recursive routing algorithm. There are $\theta(\log n)$ levels of recursion. Packets that originate in a corner supersquare that are destined for an opposite corner supersquare are the most troublesome to route. Highest priority is given to these packets while routing. This idea was first introduced in [1], and subsequently employed also in [4]. Packets are grouped into different types depending on their origins and destinations. Each type of packets executes a different algorithm.

We categorize the packets into three major types. Packets that originate in a corner supersquare and whose destinations are in an opposite corner supersquare will be called 'critical packets' (following the definitions given in [4]). The rest of the packets (called 'ordinary packets') are further categorized into two types: Any ordinary packet whose destination

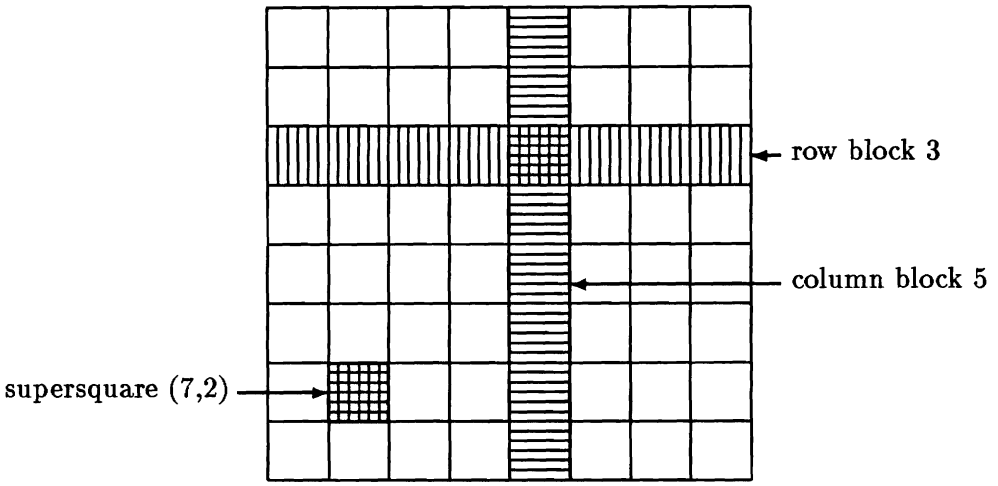

Figure 3: Partitioning of the Mesh into supersquares.

is in column block 4 or 5 will be referred to as a 'type-I' packet; the rest of the ordinary packets will be known as 'type-II' packets.

Next we describe the algorithms executed by the different types of packets.

5.1 Algorithm for type-II packets

Kunde's algorithm is used to route type-II packets. Partition each supersquare into $s \times s$ subsquares (for some s to be specified). Sort **all the ordinary** packets in each such $s \times s$ square according to their destination column (in column major order). We emphasize the fact that all the ordinary packets participate in this sorting phase. This sorting step can be performed in $T_s = 3s + o(s)$ steps (see theorem 3.1).

After the sorting step, if a type-II packet q is in node (i, j), it is sent along row i to column l (where (k, l) is the destination of this packet), and then q is sent to node (k, l) along column l. Queue discipline $\mathcal{Q}'$ is used during the column routing.

During the row routing there is no queuing of packets and hence the time needed for a packet to reach its destination is simply the distance between its origin and destination. During the column routing queues can build and the queue size at any node can not exceed $2n/s$ (for a proof see [3] or [4]). The following lemma has been proven by Leighton, Makedon, and Tollis [4]:

Lemma 5.1 *The above algorithm routes all type-II packets in time* $2n - 2 - m + T_s$ *(where*

T_s is the time needed for sorting $s \times s$ squares). Also, routing in columns c and $n - c + 1$ (for $1 \leq c \leq n/2$) will be done by step $2n - 1 - \max(m + 1, c) + T_s$ even if column routing in these columns is delayed until step $n + T_s - \max(m + 1, c)$.

5.2 Algorithm for critical packets

If there were no ordinary packets in the mesh, critical packets could be sent as blocks (in $14m$ steps) to opposite corner supersquares and they could be recursively routed there (in $2m - 2$ steps), accounting for a total of $16m - 2 = 2n - 2$ steps. But the ordinary packets can potentially interfere with the recursive routing. This is why [4] use the following algorithm for routing critical packets. 1) The critical packets are moved as blocks to the 'center' of the mesh (where there is more slack); 2) these blocks are recursively routed in $(2m - 2)$ steps, and finally 3) they are moved intact to the appropriate corner supersquares. The queue size of the algorithm is made constant by picking m and s as constant fractions of n, and making sure the region where recursive routing is done is empty (just before the critical packets enter).

We also make use of the same algorithm to route critical packets, except that we choose a different region to perform recursive routing. Details of our algorithm follow. We only describe the algorithm used by the critical packets originating from the supersquare $< 1, 1 >$. The other critical packets execute a symmetric algorithm.

The sorting algorithm presented in section 3, is such that at any given time, either only column edges are used or row edges are used for transmitting packets. We assume $m = T_s$. During the T_s steps during which the $s \times s$ squares are sorted, the critical packets are moved as blocks uniformly making use of the unused edges. The critical packets from supersquare $< 1, 1 >$ then move down all the way up to row block 8, and then move along row block 8 until they reach the supersquare $< 8, 5 >$. $11m$ steps would have passed by now. Recursive routing of critical packets from $< 1, 1 >$ is performed here in $2m - 2$ steps. After recursive routing, the block moves to supersquare $< 8, 8 >$ in $3m$ steps, accounting for a total of $11m + (2m - 2) + 3m = 16m - 2 = 2n - 2$ steps.

Just before the critical packets enter supersquare $< 8, 5 >$, this supersquare will be evacuated, so that only critical packets will be present in this supersquare when recursive routing is performed. This evacuation ensures that queues in the supersquare $< 8, 5 >$ do not build over the $\theta(\log n)$ levels of recursion (requiring a queue size of $\Omega(\log n)$). Also since the critical packets have the highest priority in the mesh, they will never be delayed by

ordinary packets. Therefore, all the critical packets can be routed within $2n - 2$ steps.

Even after accounting for the interference of critical packets with type-II packets, all the type-II packets can still be routed within $2n - 2$ steps for the following reason: Clearly, critical packets do not interfere with the row routing of type-II packets. Notice also that critical packets from supersquare $< 1,1 >$ will reach row block 8 before step $8m - 1$, and hence even if column routing is delayed until step $8m - 1$, routing of type-II packets can be completed in $2n - 2$ steps (see lemma 5.1).

5.3 Row routing of type-I packets

For routing critical and type-II packets we use essentially the same algorithms used by [4]. However for routing type-I packets we use an entirely different algorithm. Also in [4]'s algorithm, column routing is frozen for type-I packets during the $2m - 2$ steps of recursive routing of critical packets. But we do column routing even when the recursive routing is performed.

Type-I packets are those that are destined for column blocks 4 or 5. Call the type-I packets that originate from row block 1 or row block 8 as type-Ia packets, and the rest as type-Ib packets.

After the initial phase of sorting, type-Ib packets traverse along the current row to the column of their destination and join the column routing.

Type-Ia packets can not use the same algorithm since the recursive routing region has the potential of being occupied when the critical packets arrive. And hence they use a slightly different algorithm. The following algorithm for type-Ia packets pertains to those in row block 8. A symmetric algorithm is executed by packets from row block 1.

Immediately after the initial sorting phase, type-Ia packets that originate from supersquares $< 8,4 >$, and $< 8,5 >$ are moved as blocks up, to supersquares above as shown in figure 4b. For example, the leftmost 2/3rd of $< 8,4 >$ is moved to $< 7,4 >$; the rightmost 1/3rd of $< 8,4 >$ is move to $< 6,4 >$ and so on. These type-Ia packets then move along the current row to the column of their destination and join the column routing. Type-Ia packets that originate from supersquares $< 8,1 >, < 8,2 >$, and $< 8,3 >$ are moved as blocks to an appropriate supersquare in column block 4, and type-Ia packets that originate from $< 8,6 >, < 8,7 >$, and $< 8,8 >$ are moved to an appropriate supersquare in column block 5 (as shown in figure 4a). For example, packets from $< 8,1 >$ are moved to $< 7,4 >$; packets from $< 8,7 >$ are moved to $< 6,5 >$ and so on. These packets then move along the

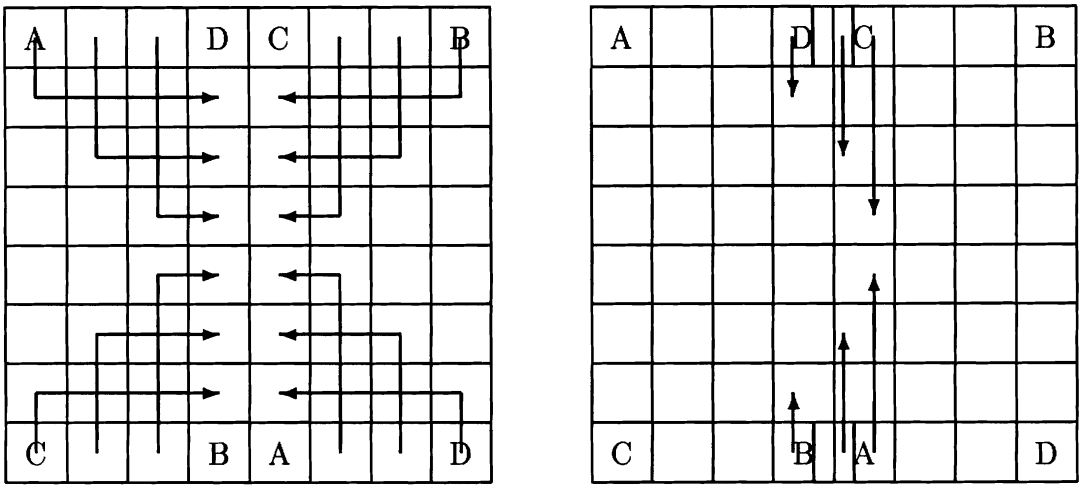

Figure 4: (a and b): Distribution of type-Ia packets

current row to the column of their destination and join the column routing.

Some of the type-Ia packets that originate from supersquares $< 1,1 >, < 1,2 >, < 1,7 >, < 1,8 >, < 8,1 >, < 8,2 >, < 8,7 >$, and $< 8,8 >$ can be delayed by the critical packets. This may increase the queue size of the algorithm. We avoid this delay by sending portions of supersquares first horizontally and then vertically. For example (see figure 5), the supersquare $< 1,1 >$ splits into two after sorting. These two portions unite in supersquare $< 2,4 >$ traversing on different paths, such that the net delay due to critical packets is zero.

It is easy to see that each type-Ib packet will be in its correct column (and ready to participate in column routing) by step 6m, and each type-Ia packet will be in its correct column by step $7\frac{2}{3}m$. For example, type-Ia packets from supersquare $< 8,1 >$ will reach supersquare $< 7,4 >$ (as a block) by step $5m$. Each packet then needs at the most $2\frac{2}{3}m$ additional steps to go to its column of destination. Thus all the type-I packets will reach their column of destination by step $7\frac{2}{3}m \leq (8m-1)$. It remains to show that column routing of type-I packets can be accomplished in $(8m-1)$ further steps.

Notice that type-I packets do not interfere with either the row routing or the column routing of type-II packets.

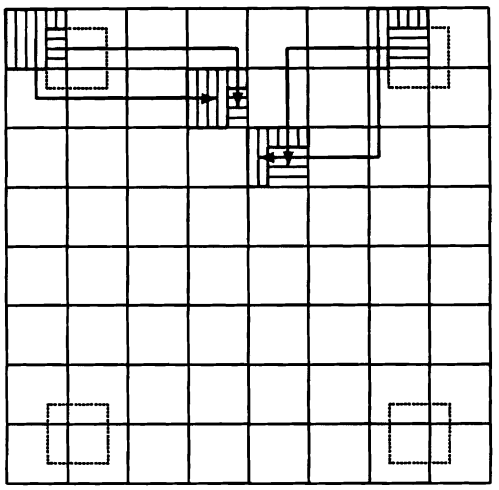

Figure 5: Avoiding delays due to critical packets

5.4 Column routing of type-I packets

Here we describe the algorithm used for column routing of type-I packets and show that column routing can be completed in $(8m-1)$ steps.

During steps $8m-1$ **through** $14m-2$ all the type-I packets participate in column routing, using queue discipline $\mathcal{Q}'$. Packets with a destination in either row block 1 or row block 8 (call these packets *special packets*) are routed as though their destination is (the corresponding node) in row block 2 or row block 7 (respectively). Special packets will have the highest priority in column routing. By step $14m-2$, each special packet will be in its corresponding node in a neighboring row block.

Any type-I packet with a destination outside of row blocks 1 and 8 will be at the most $2m$ distance away from its final destination at time step $14m-2$ (in accordance with lemma 4.2).

During steps $14m-2$ **through** $16m-2$, special packets move as blocks to the correct supersquare in m further steps. Other packets will also reach their destination within $2m$ further steps, even if the (non special) packets present in supersquares $<2,4>, <2,5>, <7,4>$, and $<7,5>$ at time step $14m-2$ are not processed during the time steps $14m-2$ through $15m-2$ (see lemma 4.4).

6 Queue Size Analysis

First we analyze the queue size of an abstract algorithm and then apply the results to our routing algorithm. Let an $n \times n$ mesh be partitioned into $s \times s$ subsquares, and let each such subsquare be sorted according to column number in column major order. Let the mesh also be partitioned into $m \times m$ supersquares (for $m \geq s$). Let a_i be the number of packets in subsquare i that are destined for a particular column say l (of length u). If r such $m \times m$ supersquares are arranged as a row block (see figure 6), and each packet traverses to its correct column along its current row, the maximum queue size of any node in column l will be

$$\leq \sum_{i=1}^{rm/s} \lceil a_i/s \rceil \leq \sum_{i=1}^{rm/s} \frac{a_i}{s} + \frac{rm}{s} \leq \frac{u}{s} + \frac{rm}{s}. \quad (1)$$

Clearly, in our algorithm, the queue size for nodes other than the ones in column blocks 4 and 5 is $2n/s$. We make use of equation 1 to compute the queue size of nodes in column blocks 4 and 5. Notice that at the most $10\frac{2}{3}$ supersquares contribute to the queue size in any column (in column blocks 4 and 5). Thus substituting $r = 10\frac{2}{3}$ and $u = 8m$ in equation 1, we conclude that the queue size is no more than $18\frac{2}{3}\frac{m}{s}$. Taking $m = 3s$, this becomes 56. But T_s is not exactly $3s$. It is $3s + o(s)$. Even for reasonably small mesh sizes, $T_s \leq 3.1s$, and hence the queue size is ≤ 58.

Thus we have the following

Theorem 6.1 *Permutation routing on an $(n \times n)$ mesh can be performed in $2n - 2$ steps with a queue size of only 58.*

7 Conclusions

The routing algorithm we presented runs in $2n - 2$ steps and has a queue size of 58. It would be nice to have an algorithm with a queue size of < 10. If one uses Kunde's routing algorithm with $s = n/24$, the queue size will be 48. The run time will be $2\frac{1}{8}n$ steps. For this value of s, the queue size we achieve in our algorithm is very close to 48 and the run time is optimal. One of the bottlenecks in reducing the queue size further is the initial sorting phase. This is a costly operation. Using Kunde's original algorithm, if one wants get a time bound of $< 3n$ steps, the queue size will have to be > 6. Even if one wants to spend $2.5n$ steps, the queue size can only be brought down to 12.

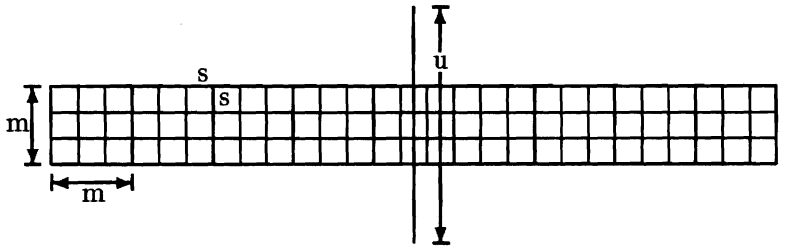

Figure 6: Queue size analysis.

The only function of the sorting phase is to distribute the packets that have the same destination column. Another faster way of distributing the packets will reduce the queue size further. We are currently exploring methods to perform such a distribution. We are also looking at the problem of routing on a mesh with wrap arounds.

References

[1] 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.

[2] Kunde, M., 'Optimal Sorting on Multi-Dimensionally Mesh-Connected Computers,' STACS 1987, Springer-Verlag Lecture Notes in Computer Science 247, pp. 408-419.

[3] 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.

[4] 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.

[5] Pippenger, N., 'Parallel Communication with Limited Buffers,' Proc. IEEE Symposium on Foundations Of Computer Science, 1984, pp. 127-136.

[6] Ranade, A.G., 'How to Emulate Shared Memory,' Proc. IEEE Symposium on Foundations Of Computer Science, 1987, pp. 185-194.

[7] Schnorr, C., and Shamir, A., 'An Optimal Sorting Algorithm for Mesh Connected Computers,' Proc. 18th Annual ACM Symposium on Theory Of Computing, 1986, pp. 255-263.

[8] Thompson, C., and Kung, H.T., 'Sorting on a Mesh-Connected Parallel Computer,' CACM, vol. 20, 1977, pp. 263-270.

[9] Upfal, E., 'Efficient Schemes for Parallel Communication,' J. ACM 31, 3, 1984, pp. 507-517.

[10] Valiant, L.G., and Brebner, G.J., 'Universal Schemes for Parallel Communication,' Proc. ACM Symposium on Theory Of Computing, 1981, pp. 263-277.

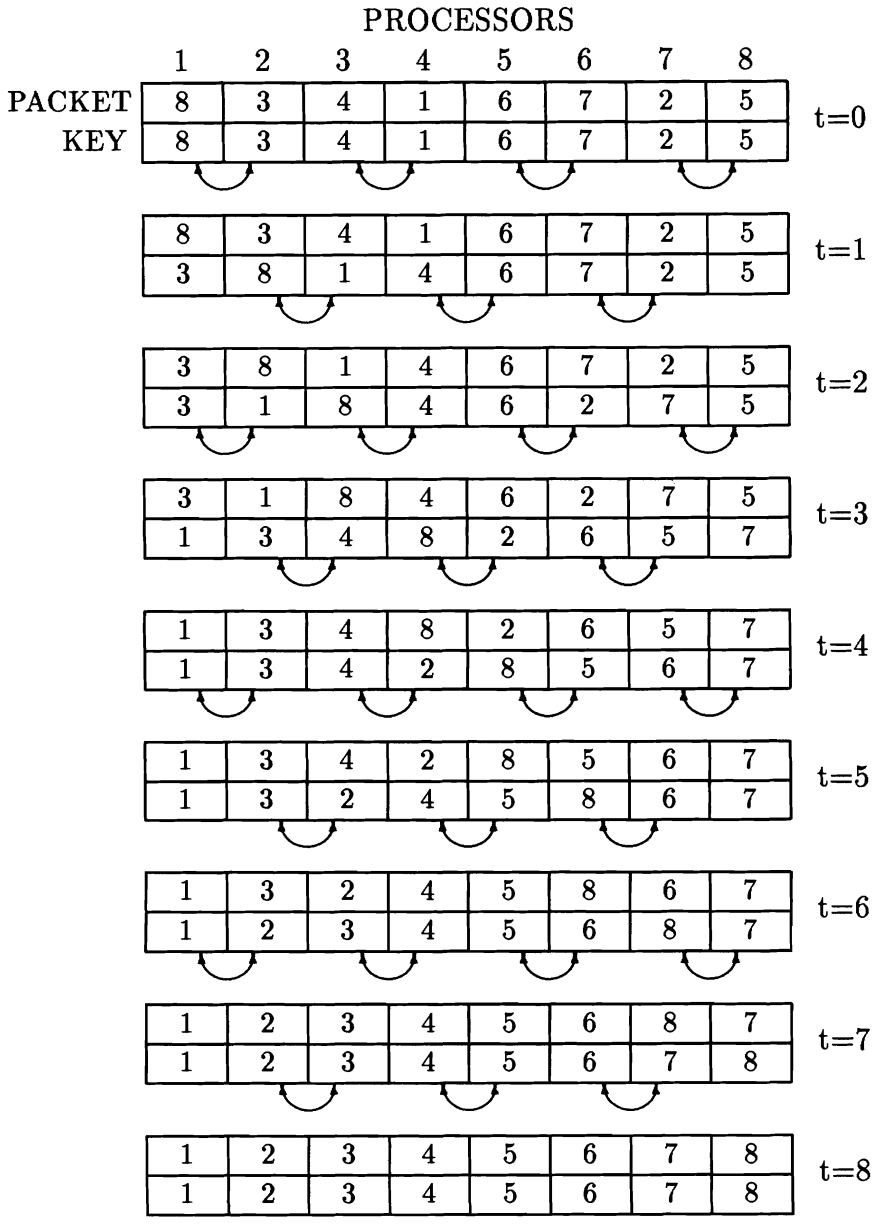

Figure 7: Routing on a Linear Array: An Example.