Heuristic Methods for Permutation Routing on Meshes and Tori

Fillia Makedon
Department of Computer Science
Dartmouth College,
Hanover NH 03755, USA
makedon@dartmouth.edu

Antonios Symvonis
Department of Computer Science
University of Sydney
N.S.W. 2006, Australia
symvonis@cs.su.oz.au

Abstract A lot of effort has been devoted to the study of the permutation routing problem on meshes and tori. There are algorithms for meshes that route any permutation in optimal time and use constant size queues. Eventhough these algorithms are theoretically optimal, the size of the queues they use is quite large to be efficient. For the torus, we do not know of any optimal algorithm. In this paper, we present heuristic methods for the permutation routing problem on meshes and tori. Our heuristics do not need any extra buffering area during the routing. We present a detailed experimental study based on randomly created data and we draw conclusion of when to use each method. The performance results are more than satisfactory. For large meshes and tori, it was a rare event when the random data generator created permutations that we couldn't route in optimal time.

1 Introduction

A crucial component of any large scale parallel machine is the algorithm that is used to route messages (also called packets) between nodes in the network. For a parallel computer to be computationally effective, it must be able to route messages from their origin processors to their destination processors quickly and with small, preferably constant size queues (queues are created while two or more packets are competing for the same communication channel). This is the task of the packet routing algorithm. An increasing number of parallel machines are being configured as 2 or 3-dimensional arrays or tori. For example, the MPP, DAP, Ametek, MIT J-machine, Intel Touchstone, and IBM Victor are all array based machines containing a large number of processors.

A lot of work has been devoted to the investigation of the permutation packet routing problem on mesh connected machines. As a result of these efforts several algorithms are available for different models of communication. The permutation packet routing problem is important because it models the communication pattern that occurs in an EREW (Exclusive Read-Exclusive Write) PRAM. The first algorithm was a probabilistic one by Valiant and Brebner [15], later improved by Krizanc, Rajasekaran and Tsantilas [7]. Kunde [8] provided the first deterministic algorithm that matched the results of the probabilistic ones. A major breakthrough was the algorithm by Makedon, Leighton and Tollis [10]. That algorithm was time and space optimal and worked on two dimensional meshes. It performed the routing in optimal time using constant size queues of 1008 packets (according to [12]). Even though that result was a theoretical breakthrough, as the authors admit:

> "... it does not appear that the constant bound on the resulting queue size will be practical for moderate values of n." (n is the sidelength of the mesh.)

That queue size was later reduced by Rajasekaran and Overholt [12] to about 150 packets (according to [13][1]). Later on, Chlebus, Kaufmann and Sibeyn [4] developed an optimal time ($2n - 2$ steps) algorithm which uses queues of at most 81 packets and a near optimal time ($2n + O(1)$ steps) algorithm which uses a maximum queue of 31 packets. Subsequently, they reduced the queue requirements of these algorithms to 33 and 12 packets, respectively [13]. However, despite the dramatic drop of the queuesize compared to the algorithm in [10], these algorithms still remain of pure theoretical interest since they are too complicated to be useful in an applied setting. A heuristic approach to the problem was presented by Symvonis and Makedon [14, 11]. Several classes of routing patterns were routed optimally while, for random permutations, the experimental behavior of the method was satisfactory.

The paper is organized as follows: Section 2 contains formal definitions of terms used in the paper. In Section 3, we present four heuristic methods (two based in distance normalization and two based on randomization) while, in Section 4 we present a detailed experimental study. In Section 5, we demonstrate a class of permutations for which the heuristics based on distance normalization perform non optimally. We conclude in Section 6.

[1] Orriginally, a queuesize of 112 packets was claimed but a flaw in the computations discovered by Chlebus, Kaufmann and Sibeyn [4] made it necessary to revise the estimated queuesize to 150 packets.

2 Preliminaries

A $n \times n$ *mesh of processors* is defined to be a graph $G = (V, E)$, where $V = \{(i, j) | 0 \leq i, j \leq n-1\}$, and an edge $e = ((i, j), (k, l))$ belongs to E if $|k - i| + |l - j| = 1$.

At any one step, each processor can communicate with all of its neighbors by the use of bidirectional links (called channels). We define the *distance* between two processors $P_1 = (i, j)$ and $P_2 = (k, l)$ as *distance*(P_1, P_2)=$|k - i| + |l - j|$. If *distance*(P_1, P_2)=1 then P_1 and P_2 are *neighbors*. It is convenient for the description of our algorithms to talk about the *east*, the *west*, the *north* and the *south* neighbors of a given processor. Formally, for processor $P = (i, j)$, $0 < i, j < n - 1$, we define the east neighbor to be processor $P_e = (i, j + 1)$, the west neighbor to be processor $P_w = (i, j - 1)$, the north neighbor to be processor $P_n = (i - 1, j)$, and the south neighbor to be processor $P_s = (i + 1, j)$. A processors $P = (i, j)$ is said to be on the *boundary* of the mesh if $i = 0$ or $i = n - 1$ or $j = 0$ or $j = n - 1$. Processors on the boundary of the mesh do not have all of their four neighbors defined. A mesh provided with wrap-around connections is called *torus*. All processors are assumed to work in a synchronous MIMD model.

In a *permutation problem*, each processor has one packet to transmit to another processor. At the end, each processor receives exactly one packet (1-1 routing). The problem here is to route all packets to their destinations fast and without the use of large additional storage area (buffers) in each processor.

A permutation problem belongs in the category of *static routing* problems. In a static routing problem a communication pattern is given. During the routing of the problem, the processors are not allowed to generate new packets and inject them into the network. When static routing is used, we can imagine the execution of a task as a sequence of computation and communication phases. The emulation of PRAM algorithms in realistic parallel machines can be done in this way. In the case that the processors can create and inject packets into the network at any time we have *dynamic routing*.

3 Heuristics for Permutation Routing

In this section we present four heuristic methods that do not require any buffering area during the routing. Extensive simulation results based on random routing patterns on meshes and tori of different sizes are presented.

In an on-line routing algorithm, at any step, each processor has to make several routing actions. These actions can be considered to follow the program skeleton bellow that is executed by any processor P at time step i.

Route(P, i)

while the routing is not over **do**

1. Receive packets from step $i - 1$.
2. Accept packets destined for processor P.
3. Assign output links to remaining packets.

Figure 1: EXCHANGE

4. Transmit packets.

Steps 1, 3 and 4 might depend on the position of the processor in the interconnection network (mesh or torus in our case) and the particular time step. Step 3 is the most important one. Each processor P decides how the packets that entered P as an intermediate destination will continue their trip. For all the algorithms in this paper we will informally describe the rules that are used in making that decision.

3.1 Heuristics based on the odd-even transposition method

The odd-even transposition method was used for sorting in parallel the elements of a linear array[2]. Assume a linear array of n processors laying down from left to right. Furthermore, assume that the edges are numbered from 1 to $n-1$ in the order they appear in a left to right traversal. When the odd-even transposition takes place, at odd (even) time steps only the edges that are marked with an odd (even) number are active.

Now, assume that the odd-even transposition takes place and the only action that two processors connected by an active edge are allowed to do is the swapping of their contents (packets from now on). The effect of this restriction is that, at any time, at most one packet is at any processor.

When the problem in hands is the sorting problem, then the swapping between the elements of two processors is based on their values. However, when we have to attack the routing problem, some other criterion must be used. A natural candidate is the distance that each packet has still to travel on a shortest path towards its destination. So, imagine two processors that contain two packets such that, if an exchange is done, the distance that each packet has still to travel is reduced by one (Figure 1). In this case we allow the swap. In the opposite case where both distances are increased by one (Figure 2) it is natural not to do the exchange. But what do we do in the case that the distance that the one packet still has to travel is increased by one while the distance the other packet has still to travel is decreased by one (Figure 3)? It seems a good idea to allow the exchange if the largest of the two distances decreases and to forbid it otherwise. The above procedure was called *distance normalization* by Symvonis and Makedon in [14, 11]. The following two heuristic methods are based on the odd-even transposition method and the notion of normalization.

[2]It is attributed to [2]. See [1] or [9] for a proof of correctness.

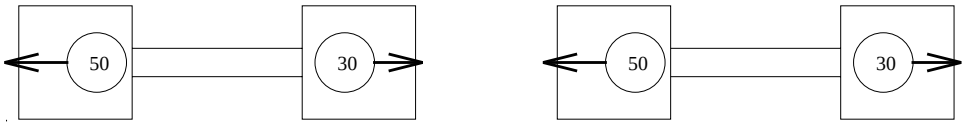

Figure 2: NO EXCHANGE

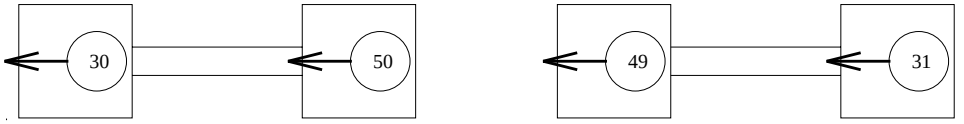

Figure 3: NORMALIZE

3.1.1 Complete Distance Normalization Heuristic

The description of any routing algorithm must include two things: The assumed communication model (what connections are active in each step) and the way routing decisions are taken. In the *complete distance normalization heuristic* it is assumed that the communication along the rows of the mesh (torus) follows the odd-even transposition pattern. On the other hand, all communication links along the columns of the mesh (torus) are active at any time step. This communication model is not utilizing all mesh (torus) connections. At any time step, each processor has 3 active links connected to it (processors on the boundary of the mesh are an exception) out of the 4 possible.

Routing decisions are taken based on distance normalization. The odd-even transposition takes places on the rows of the mesh (torus). This has the result that, eventually, each packet will reach its column destination. If, at that time, there is no other packet at that processor that also wants to move along the column in the same direction (south or north), it starts its vertical movement. In the case that some other packet was moving along the column, the two packets have to compete for one communication link. The link is assign to the packet that has to go further while the other packet will participate in the odd-even transposition that takes place along the rows. Observe that with the above procedure no packet is ever queued at any processor.

3.1.2 Partial Distance Normalization Heuristic

In the complete distance normalization heuristic a packet that starts its vertical movement can be derouted by another packet that just reached its column destination. As an effect a packet might travel along several rows of the mesh (torus) before reaching its destination. In an attempt to simplify the heuristic we performed the normalization only along the rows of the mesh (torus), and thus the name *partial distance normalization*.

Now, a packet that starts its vertical movement will never be interrupted. As a consequence, a packet that reaches its column destination but cannot start its vertical movement because of another packet, will continue to participate in the odd-even transposition that takes place along the rows.

In [14, 11] it was proven that the complete distance normalization heuristic routes optimally

rotations, inversions and transpositions. These routing patterns are also routed optimally by the partial distance normalization heuristic. In fact, the proofs are identical. An upper bound of $O(n^2)$ on the number of steps that the algorithms require in order to complete the routing was given. However, we were not able to construct a routing pattern that requires $\Omega(n^2)$ routing steps.

3.2 Heuristics Based on Randomization

Our goal is to derive a routing method that needs no buffering area. There is a simple way to achieve that. The only thing that a processor has to do is to transmit all the packets it received from the previous step. Of course, since we can transmit only one packet per communication link, some packets might have to be derouted. Despite the fact that this technique is very old (it was referenced as folklore by Borodin and Hopcroft in [3]) we are not able to make any statement regarding the number of routing steps that are required in the worst case in order to route any permutation.

3.2.1 Simple Randomization Heuristic

It is obvious that if we want the routing to terminate, we must route as many packets as possible towards their destination. For this reason, the *simple randomization heuristic* will assign output links to packets in such a way that as many as possible packets are routed closer to their destination. There might be several such possible assignments. The simple randomization heuristic chooses one randomly. It also assigns in a random way output links to the packets that have to be derouted.

3.2.2 Distance-Critical Randomization Heuristic

In the simple randomization heuristic the packets that are routed towards their destination are selected randomly out of all possible assignments. In the *distance-critical randomization heuristic* these assignments are based on the distance the packets still have to travel. The packet that has to go further has higher priority. Any remaining conflicts are resolved in a random matter. We have to emphasize that, still, the maximum possible amount of packets are routed towards their destination.

4 Performance

We present experimental results based on randomly created permutations for meshes and tori. We will refer to the four heuristic methods as *heuristic_1*, *heuristic_2*, *heuristic_3*, and *heuristic_4* with respect to the order that they were presented.

Usually, we say that a routing algorithm performs optimally if it can route any permutation in at most so many steps as the diameter of the interconnection network. However, this definition

of optimality is valid in the worst case where there exists a packet that has to travel along a path of length equal to the diameter. It is more natural to say that an algorithm is optimal if it succeeds to route a permutation in a number of steps equal to the maximum distance that some packet has to travel in that particular permutation. (This is a lower bound based on the actual routing problem in hands rather than a worst case situation.) We will present experimental data based on the diameter of the mesh ot torus and the actual maximum distance.

4.1 Routing on Meshes

We run experiments on meshes of size 10×10, 20×20, ... , 100×100. The number of experiments decreases as the sidelength of the mesh increases. We devoted a fixed amount of time to the study of each mesh and, naturally, simulating the routing on larger meshes takes more time.

Table 1 contains results from the simulations of *heuristic_1* and *heuristic_2* with respect to the maximum distance bound. In this table, as well as in all tables in this paper, we present the number of experiments that failed to complete the routing on time and the total number of delayed packets over all experiments. From the table it is obvious that *heuristic_1* was always better than the simpler *heuristic_2* but the gain was very small. An interesting thing was revealed when comparing the total number of delayed packets and the number of delayed experiments. The difference is very small. This implies that, on average, there is about one delayed packet per delayed experiment. Actually, most of the delayed experiments demonstrate a delay of one step. This delay can be contributed to the nature of the odd-even transposition. A packet that has to travel a maximum distance might not be able to move during the first step of the routing because of an non-active connection.

Table 2 presents our simulation results with respect to the diameter lower bound. The random permutations on Tables 1 and 2 are identical. We observe that in all the experiments for meshes of dimensions smaller than 40×40 the number of delayed experiments is very small compared to the total number of performed experiments. For meshes with dimensions greater or equal to 40×40 the random data generator did not produce any permutation that the heuristics couldn't route on time. A conclusion of our study is that, for meshes, the use of the simpler *partial distance normalization* heuristic gives comparable results with the *complete distance normalization* heuristic.

Table 3 presents a similar study for *heuristic_3* and *heuristic_4* based on the maximum distance bound. Now, it is revealed that if randomization is our last resort in resolving conflicts, better results can be obtained. The number of delayed experiments is at least fifty times greater if distances are not used in resolving conflicts (for meshes of dimensions larger than 20×20). Table 4 contains our experimental results with respect to the diameter lower bound. Again, *heuristic_4* has superior behavior but *heuristic_3* is also satisfactory.

An attempt to compare the four heuristic methods reveals that *heuristic_4* is preferable with respect to the maximum distance. With respect to the diameter lower bound all heuristics seem to have satisfactory performance. One reason that might justify the use of the heuristics based on distance normalization is that they perform optimally for the inversion, transposition, and rotation routing patterns.

Table 1: Comparison between *heuristic_1* and *heuristic_2* on Meshes WRT Maximum Distance

		heuristic_1		*heuristic_2*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	1159853	663763	578353	737070	615710	0.939
20X20	220478	101967	99231	112787	105121	0.944
30X30	65412	30085	29755	33078	31303	0.951
40X40	33878	15718	15577	17128	16312	0.955
50X50	14857	6897	6857	7470	7152	0.959
60X60	8763	4121	4105	4440	4275	0.960
70X70	8128	3816	3806	4101	3962	0.961
80X80	6561	3042	3036	3257	3159	0.961
90X90	5142	2323	2323	2478	2406	0.966
100X100	4099	1949	1946	2071	2013	0.967

Table 2: Comparison between *heuristic_1* and *heuristic_2* on Meshes WRT Diameter Lower Bound

		heuristic_1		*heuristic_2*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	1159853	3605	3527	4092	3962	0.890
20X20	220478	39	39	43	43	0.907
30X30	65412	2	2	2	2	1.000
40X40	33878	0	0	0	0	-
50X50	14857	0	0	0	0	-
60X60	8763	0	0	0	0	-
70X70	8128	0	0	0	0	-
80X80	6561	0	0	0	0	-
90X90	5142	0	0	0	0	-
100X100	4099	0	0	0	0	-

Table 3: Comparison between *heuristic_3* and *heuristic_4* on Meshes WRT Maximum Distance

		heuristic_3		*heuristic_4*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	508115	62267	41985	3581	3559	11.797
20X20	67908	9334	5651	200	198	28.540
30X30	9531	1489	845	16	15	56.333
40X40	5087	709	402	7	7	57.429
50X50	4740	795	444	2	2	222.000
60X60	4577	782	418	4	4	104.500
70X70	3747	580	309	1	1	309.000
80X80	2370	378	203	3	3	67.667
90X90	1686	246	129	2	2	64.500
100X100	1258	205	111	0	0	-

Table 4: Comparison between *heuristic_3* and *heuristic_4* on Meshes WRT Diameter Lower Bound

		heuristic_3		*heuristic_4*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	508115	1839	1646	2	2	823.000
20X20	67908	56	49	0	0	-
30X30	9531	10	7	0	0	-
40X40	5087	2	2	0	0	-
50X50	4740	1	1	0	0	-
60X60	4577	1	1	0	0	-
70X70	3747	0	0	0	0	-
80X80	2370	0	0	0	0	-
90X90	1686	0	0	0	0	-
100X100	1258	0	0	0	0	-

Table 5: Comparison between *heuristic_1* and *heuristic_2* on Tori WRT Maximum Distance

		heuristic_1		*heuristic_2*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	1002720	2048544	851042	2249236	908267	0.937
20X20	155370	214863	100059	298668	131093	0.763
30X30	64205	40652	24711	89485	47634	0.519
40X40	30370	10278	8422	37725	21925	0.384
50X50	14073	3678	3481	17978	10356	0.336
60X60	8360	2078	2043	11002	6272	0.326
70X70	6440	1550	1540	8909	4888	0.315
80X80	4005	978	977	5638	3049	0.320
90X90	2755	662	661	3925	2131	0.310
100X100	1890	445	445	2790	1459	0.305

Table 6: Comparison between *heuristic_1* and *heuristic_2* on Tori WRT Diameter Lower Bound

		heuristic_1		*heuristic_2*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	1002720	1678926	762533	1876192	848176	0.899
20X20	155370	161801	74666	240957	116286	0.642
30X30	64205	22639	10633	66268	38419	0.277
40X40	30370	2640	1259	26714	17101	0.074
50X50	14073	293	153	12842	8225	0.019
60X60	8360	48	28	7957	5028	0.006
70X70	6440	13	7	6558	4005	0.002
80X80	4005	2	2	4136	2452	0.0008
90X90	2755	2	2	2954	1789	0.001
100X100	1890	1	1	2127	1234	0.0008

4.2 Routing on Tori

A similar analysis of the four routing heuristics was performed for tori. The obtained results are presented in Tables 5, 6, 7, 8. The wrap around connections that the torus is provided with, have as result to reduce its diameter (compared with a mesh of the same sidelength) to about half. For this reason, the differencies between the experimental results of the four heuristics are not so dramatic. A comparison between *heuristic_1* and *heuristic_2* suggests that we should prefare *heuristic_1* (especially for large tori). We should mention that a deadlock situation may occur in the heuristics based on the odd-even transposition. This happens if the packets in any row want to travel towards the same direction and for about the same distance. In this case, the packets are already normalized and no further movement occurs. Comparing *heuristic_3* and *heuristic_4* reveals (again) that it is worthwhile to assign priorities based on distances and to resolve the remaining conflicts based on randomization. Out of the four heuristics, *heuristic_4* seems to have the best performance.

Table 7: Comparison between *heuristic_3* and *heuristic_4* on Tori WRT Maximum Distance

		heuristic_3		*heuristic_4*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	889250	185613	144723	22031	21584	6.705
20X20	200670	36519	27671	1132	1131	24.466
30X30	62565	9702	7334	156	155	47.316
40X40	31570	4285	3225	43	43	75.000
50X50	16135	2037	1514	14	14	108.143
60X60	8080	1024	764	9	9	84.889
70X70	5810	722	538	2	2	269.000
80X80	4236	468	339	2	2	169.500
90X90	2593	260	196	1	1	196.000
100X100	2005	222	166	0	0	-

Table 8: Comparison between *heuristic_3* and *heuristic_4* on Tori WRT Diameter Lower Bound

		heuristic_3		*heuristic_4*		
DIM	*Exps*	*Delayed Packets*	*Delayed Exps* (α)	*Delayed Packets*	*Delayed Exps* (β)	(α/β)
10X10	889250	94268	82095	2293	2242	36.171
20X20	200670	19627	16401	149	149	110.074
30X30	62565	5429	4495	22	22	204.318
40X40	31750	2396	1972	6	6	328.667
50X50	16135	1154	938	2	2	469.000
60X60	8080	569	466	2	2	233.000
70X70	5810	417	336	2	2	168.000
80X80	4236	279	235	0	0	-
90X90	2593	134	109	0	0	-
100X100	2005	131	105	0	0	-

5 A Lower Bound for the Heuristics Based on Distance Normalization

In this section, we demonstrate a routing pattern for which both of the heuristic methods based on the distance normalization fail to route in optimal time. The setting of the packets is presented in Figure 4. The figure is not in scale. Each packet is represented by a letter of the alphabet (accompanied by subscripts). Lower case and upper case letters correspond to the same packet. We represent the origin of a packet by a lower case letter while, the destination is represented by an upper case letter. We can assume that each of the remaining processors has a packet destined for itself. The north part of the mesh contains all the origins while, the south part of the mesh contains all the destinations. Now consider packets $\mathtt{y}_1$, $\mathtt{y}_2, \ldots, \mathtt{y}_{\mathtt{n}-\sqrt{\mathtt{n}}-1}$. Their destinations are set up in such a way that no swap is ever perfomed between them. So, the only event that can initiate their movement is a move of packet $\mathtt{z}'_{\sqrt{\mathtt{n}}/2}$ to the east. (The distances which packets $\mathtt{y}_{\mathtt{n}-\sqrt{\mathtt{n}}-1}$ and $\mathtt{z}'_{\sqrt{\mathtt{n}}/2}$ have to travel are equal and thus, since both want to move towards the east, no swap between them is possible.) The stream of $\mathtt{y}$ packets will not be able to move continously to the east before packet $\mathtt{z}'_{\sqrt{\mathtt{n}}/2}$ reaches its column destination. We show that this will occur after abour n routing steps.

During the first step of the routing packet $\mathtt{z}_1$ moves east and at the end of the step reaches the rightmost processor in its row. That processor, during the first step, transmitted an "a" packet towards the south and also received an "a" packet from the north. So, at the end of the first step it contains $\mathtt{z}_1$ and an "a" packet. Since all "a" packets have priority over $\mathtt{z}_1$, in future steps they will be transmitted towards the south while $\mathtt{z}_1$ will participate in the odd-even transposition which takes place in the rows. As a result of that, and since $\mathtt{z}_1$ will be always compared with $\mathtt{z}'_1$, it will remain in the same processor for the next $2\sqrt{n}-1$ steps. Thus, $\mathtt{z}_1$ will move south at step $2\sqrt{n}+1$.

Consider now packet $\mathtt{z}'_1$. During step 2 it will move one position to the east and it remains there (blocked by $\mathtt{z}_1$) untill step $2\sqrt{n}$. So, at the end of step $2\sqrt{n}+1$ it reaches its column destination.

Consider also the processor which held packet $\mathtt{z}'_1$ at he end of step $2\sqrt{n}$. That processor (the second from the right in $\mathtt{z}_1$'s origin row) during step $2\sqrt{n}+1$ will transmitt $\mathtt{z}'_1$ towards the east but it will also receive a "b" packet from the north (the "b" packets have arrived!). Note that the processor under consideration is at the column destination of $\mathtt{z}'_2$ and $\mathtt{z}_2$. So, after a total of $2\sqrt{n}$, we arrived at a similar situation (to the one we started with) for packets $\mathtt{z}'_2$ and $\mathtt{z}_2$. It is

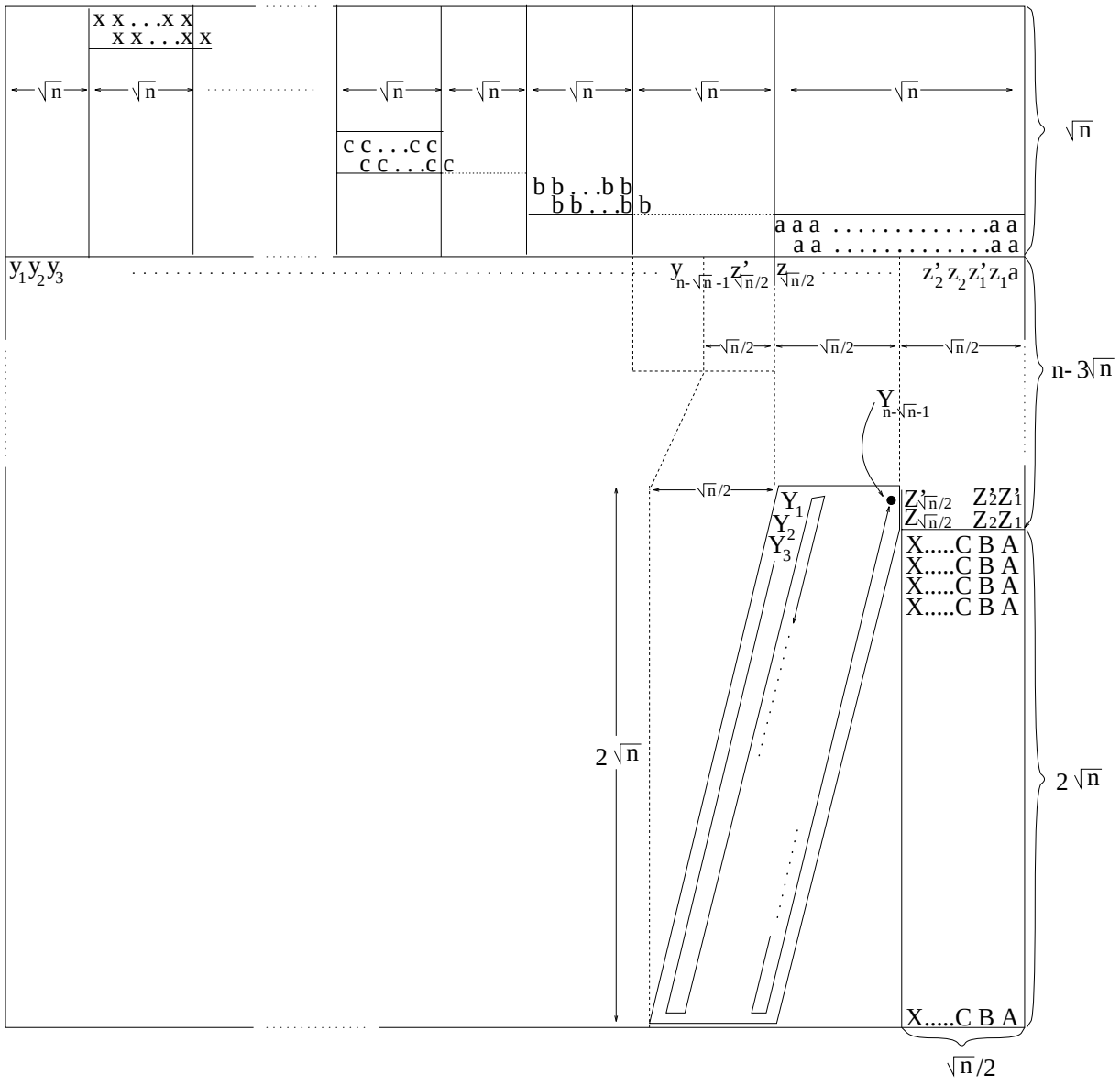

Figure 4: A routing pattern for which *heuristic_1* (and *heuristic_2*) needs $4n - o(n)$ routing steps.

easy to conclude by induction that packet $\mathtt{z}'_{\sqrt{n}/2}$ will reach each column destination after about $2\sqrt{n}\frac{\sqrt{n}}{2} = \sqrt{n}$ routing steps.

Consider now packet $\mathtt{y_1}$. It will start it uninterupted movement after about about $n - \sqrt{n}$ steps[3]. This is because it takes so many steps for the packets infront of it to start their continous movement. Then, it has to travel for $2n - 4\sqrt{n}$ more steps. So, it will arrive to its destination after $4n - o(n)$ routing steps.

During our simulations, the random data generator never created a permutation that requires such a large number of routing steps. Note that, the above construction actualy defines a class of permutations, not just one permutation. We can obtain the permutations in this class by

[3] Actually, when packet $\mathtt{y_1}$ will start its uninterupted trip towards its destination, it will have already advanced about $\frac{\sqrt{n}}{2}$ positions. This is because packet $\mathtt{z}'_{\sqrt{n}/2}$ advanced $\frac{\sqrt{n}}{2}$ positions to the east.

permuting the packets in each of the "a", "b", ..., "x" groups or by permuting the packets which didn't participate in the routing pattern used in the lower bound proof (these packets were assume to have identical origin and destination processors). The above construction enables us to state the following theorem.

Theorem 1 *There is a class of permutations for which heuristic_1 and heuristic_2 require* $4n - o(n)$ *routing steps.*

6 Conclusions

The study of heuristic methods that use small queues was motivated by the absence of appropriate algorithms. Eventhough, for meshes, there exist theoretically optimal algorithms, one that uses small constant queues, performs the routing in optimal time and is simple enough to be implemented in practice, is not known. In this paper, we experimentally compared four such heuristics all of which appeared to have satisfactory performance. We also proved a lower bound on the number of routing steps required by the two heuristics which were based on the odd-even transposition. Closing the gap between the lower bound and the $O(n^2)$ upper bound[11] appears to be a challenging problem. Analyzing the perfomance of all four heuristics in a dynamic environment is very interesting but, it appears to be difficult since the necessary mathematical tools for the analysis are not available.

References

[1] S.G. Akl, "The Design and Analysis of Parallel Algorithms", Prentice Hall, 1989.

[2] H.B. Demuth, "Electronic Data Sorting", Ph.D. Thesis, Stanford University, Stanford, California, October 1956.

[3] A. Borodin, J.E. Hopcroft, "Routing, Merging, and Sorting on Parallel Models of Computation", Journal of Computer and System Sciences, 30, pp. 130-145, 1985.

[4] B. Chlebus, M. Kaufmann, J. Sibeyn, "Deterministic Permutation Routing on Meshes", Proceedings of the Fifth Symposium on Parallel and Distributed Processing, Texas, December 1993, pp. 814-821.

[5] C. Kaklamanis, D. Krizanc, S. Rao, "Simple Path Selection for Optimal Routing on Processor Arrays", Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'92, June 1992, San Diego, pp. 23-30.

[6] M. Kaufmann, J. Sibeyn, T. Suel, "Derandomizing Algorithms for Routing and Sorting on Meshes", Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 669-679.

[7] D. Krizanc, S. Rajasekaran, Th. Tsantilas, "Optimal Routing Algorithms for Mesh-Connected Processor Arrays", VLSI Algorithms and Architectures (AWOC'88), J. Reif, editor, Lecture Notes in Computer Science 319, 1988, pp. 411-422.

[8] M. Kunde, "Routing and Sorting on Mesh-Connected Arrays", VLSI Algorithms and Architectures (AWOC'88), J. Reif, editor, Lecture Notes in Computer Science 319, 1988, pp. 423-433.

[9] F.T. Leighton, " Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", Morgan Kaufmann Publishers, 1991.

[10] F.T. Leighton, F. Makedon, I.G. Tollis, "A 2n-2 Algorithm for Routing in an $n \times n$ Array With Constant Size Queues", to appear in Algorithmica. Also in the Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89, June 1989, pp. 328-335.

[11] F. Makedon, A. Symvonis, "An Efficient Heuristic for Permutation Packet Routing on Meshes with Low Buffer Requirements", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 4, March 1993, pp. 270-276.

[12] S. Rajasekaran, R. Overholt, "Constant Queue Routing on a Mesh", Journal of Parallel and Distributed Computing, Volume 15, 2, pp. 160-166, 1992.

[13] J. Sibeyn, B. Chlebus, M. Kaufmann, "Sorter Queues for Permutation Routing on Meshes", Mathematical Foundations of Computer Science (MFCS'94), Lecture Notes in Computer Science 841, 1994, pp. 597-607.

[14] Symvonis A., " Packet Routing Problems on Mesh Connected Machines and High Resolution Layouts", Ph.D. Thesis, University of Texas at Dallas, 1991.

[15] L.G. Valiant, G.J. Brebner, "Universal Schemes for Parallel Communication", Proceedings of the 13^{th} Annual ACM Symposium on the Theory of Computing, May 1981, pp. 263-277.