数理解析研究所講究録
1185 巻 2001 年 90-99

Towards an Optimal Oblivious Routing Algorithm on 2D Meshes

Kazuo IWAMA[1] and Eiji MIYANO[2]

[1]Graduate School of Informatics,
Kyoto University
Sakyo-ku, Kyoto 606-8501, Japan
iwama@kuis.kyoto-u.ac.jp

[2] Kyushu Insititute of Design
4-9-1 Shiobaru, Fukuoka 815-8540, Japan
miyano@kyushu-id.ac.jp

Abstract: We present a deterministic, oblivious, permutation-routing algorithm on the two-dimensional, $n \times n$ mesh of constant queue-size. It runs in $(2.334 + \varepsilon)n$ steps for any $\varepsilon > 0$. The best previous oblivious algorithm for permutation required roughly $2.954n$ steps as shown in SPAA2000.

Keywords: mesh, permutation routing, oblivious.

1 Introduction

The problem of routing packets, pieces of information generated by each processor, through a network of processors is fundamental to the study of parallel and distributed computation since a number of applications require packets to be routed to their correct destinations within a reasonable amount of time. However, when a packet move along its path, it is common that the packet is delayed by other packets it encounters. Occasionally, a severe delay comes from heavy path-congestion of the critical region, which is mostly due to the lack of the sophisticated path design of a routing algorithm. Thus it has been a common perception that *oblivious* routing schemes can only provide us with poorer performance than *adaptive* routing schemes. Oblivious routing requires that the entire path of each packet has to be completely determined only by its source and destination before routing starts, whereas adaptive routing allows contention resolution at each processor by changing the moving direction of packets dynamically. Actually, several inefficiencies of oblivious routing are reported [BH85, Kri91, BRSU93].

Nevertheless, oblivious routing has received a large amount of support in practice because of its simplicity and recent research in the area of packet routing gave a significant progress: Iwama and Miyano [IM2000] presented a $(2.954 + \varepsilon_1)n$ oblivious routing algorithm which can route any *permutation* on the standard, two-dimensional mesh including $n \times n$ processors with a constant number of queue spaces for any ε_1. Permutation routing, where each processor is a source and destination of precisely one packet, is a standard benchmark for routing algorithms. In this paper we show that the $(2.954 + \varepsilon_1)n$ bound can be further reduced to $(2.334 + \varepsilon_2)n$ for any positive constant ε_2 and the queue space of every processor remains bounded above by some constant, i.e., a function of ε_2. Thus our current bound is some 16.7% above the optimal $(=2n - 2)$ determined by the network diameter. Most basic differences compared to the previous paper are in the way of dividing the whole mesh into submeshes. Also, we need more refined and/or simplified application of design-tools such as the *bit-reversal permutation* (brp), parallel construction of the brp, and priority movements of *important* packets (where the important packets move between opposite-end regions of the mesh, which are often called *critical* packets [LMT95, SCK97].

Thus our present result is obviously incremental and the apparent question is whether or not some natural modification of the current routing scheme can put the upper bound closer to the absolutely optimal $2n$ bound. An important objective of this paper is to give a negative conjec-

ture to this question. It should be noted that our main technique, the brp, causes an intrinsic delay of packets. So, our improvements of performances so far are simply due to how to minimize this unavoidable delay. In this paper we shall observe a simple but worst example for the brp scheme, which suggests that this approach is approaching to its best possible.

The permutation routing is a rather classic problem and research on more contemporary routing such as *wormhole routing* and *circuit-switching* (e.g., [Akl97, Lei92]) are apparently more popular. However we believe that this highly theoretical model is still important since it gives us several fundamental insights on efficient routing. For example, the sequence of our research reveals that the control of *timing* for packet movement is as important as the control of *path* which has long been the main target of this research community. In this sense, the result of [Kri91] showing that oblivious routing suffers from a very bad $\Omega(n^2)$ lower bound if one does not consider this timing problem or lets packets always go if possible, is also a great contribution.

So far a great deal of effort has been devoted to seeking the optimal time bound for adaptive routing: Leighton, Makedon, and Tollis [LMT95] gave a deterministic algorithm with running time $2n-2$, matching the distance bound, and constant queue-size. Rajasekaran and Overholt [RO92] and Sibeyn, Chlebus, and Kaufmann [SCK97] decreased the queue-size later. However, all of these algorithms involve a flavor of mesh-sorting algorithms and may be too complicated to implement on existing computers. Thus, there exist a lot of endeavors to simplify the routing schemes: For example, under the adaptive setting, Chinn, Leighton, and Tompa [CLT96] provided a *minimal* (i.e., shortest-path), nonsorting-based, adaptive routing algorithm which achieves $O(n)$ steps with constant size queue. Unfortunately, however, the best leading constant of the running time shown is at least 500 according to the paper. Apparently, an algorithm with a simpler path-selection and a smaller queue-size might be of more practical interest, even if its running time is slightly larger than $2n$. From this point of view, our algorithm with obliviousness does have more practical merits.

In what follows, after giving a review of the previous techniques in Section 3, we show a basic algorithm which runs in $(2.5+\varepsilon)n$ steps in Section 4, which itself is an improvement over [IM2000]. Our main result, the $(2.334+\varepsilon)n$ algorithm, is given in Section 5.

2 Models and Problems

Our model in this paper is the standard, two-dimensional (2D) $n \times n$ mesh. Each processor is connected to its four neighbors via point-to-point communication links and at any given step it can communicate with all neighboring processors. Packet routing requires that each packet must be routed correctly to its destination. In *permutation* routing, every processor is a source and destination of precisely one packet.

Each processor has four input and four output queues. Each queue can hold up to K packets at the same time. The one-step computation consists of the following two stages: (i) Suppose that there remain ℓ $(0 \leq \ell \leq K)$ packets, or there are $K-\ell$ spaces, in an output queue Q of processor P_i. Then P_i selects at most $K-\ell$ packets from its input queues, and moves them to Q. (ii) Let P_i and P_{i+1} be neighboring processors, for instance, P_i's right output queue Q_i be connected to P_{i+1}'s left input queue Q_{i+1}. Then if the input queue Q_{i+1} has a space, then P_i selects at most one packet (at most one packet can flow on each link in each time-step) from Q_i and send it to Q_{i+1}. Note that P_i can perform arbitrarily complex operations on the queues in each step (although it performs only very elementary operations in our algorithms), and P_i makes several decisions due to a specific algorithm in both (i) and (ii). When making these decisions, P_i can use in general any information such as the information of the packets now held in its queues.

Roughly speaking, routing is to determine each packet's path through the network by using various information, such as source addresses, destination addresses, and the configuration of the network. A routing algorithm, A, is said to be *oblivious* if the path of each packet is completely determined by its source and destination positions, not depending on other packets. The most popular, and simplest oblivious way for permutation routing on meshes is to route all the packets along their dimension-order paths, i.e., every

packet first moves along its row until it reaches its column destination, and then move along its column until it reaches its row destination. Oblivious routing generally makes algorithms simpler and have been considered to be more practical. However, it is hard to avoid path-congestion in the worst case since the path of each packet is completely determined before routing starts and it often takes much more time than it looks. For example, Krizanc proves [Kri91] that any oblivious algorithm on k-dimensional, constant queue-size, n^2 processor mesh networks requires $\Omega(n^2)$ steps in the worst case if the algorithm is *pure*, i.e., if packets must move whenever their next positions are empty, where k may be any constant. In order to obtain linear-time algorithms, therefore, we have to pay our great attention to the queue-scheduling and have to develop some mechanism which forces some packets "to wait" even if they can advance.

3 Previous Results

3.1 Basic Ideas

As shown in [Lei92, IM99], the dimension-order path algorithm must require $\Theta(n^2)$ steps in the worst case because heavy path-congestion may occur in the *critical positions*, where each packet changes its direction and enters its correct column. However, it is also well known that the dimension-order path algorithm performs very well on average: Due to Leighton [Lei90], if each packet has a random destination, then it can route all packets in $2n + O(\log n)$ steps with high probability only by using the most greedy queue scheduling, i.e., every packet is served in the first-in first-out (FIFO) fashion, and none of the queues contains more than four packets. Although this routing model is not necessarily permutation, this fact leads us to observe that if we can change an arbitrary sequence of packets on every row into such a sequence that packets of the same destination are almost uniformly distributed on the row, then we may obtain an algorithm with the same performance as the algorithm for random destinations. The bit-reversal permutation (brp) algorithm proposed in [IM99] can control the movement of each packet without destroying the oblivious condition, and can remove heavy path-congestion from the critical positions.

We first define the following two notations on sequences of packets on linear arrays:

Definition 1 Let $i_1 i_2 \cdots i_\ell$ denote the binary representation of an integer i. Then i^R denotes the integer whose binary representation is $i_\ell i_{\ell-1} \cdots i_1$. The *bit-reversal permutation* (*BRP*) π is a permutation from $[0, 2^\ell - 1]$ onto $[0, 2^\ell - 1]$ such that $\pi(i) = i^R$. Let $x = x_0 x_1 \cdots x_{2^\ell - 1}$ be a sequence of packets. Then the bit-reversal permutation of x, denoted by $BRP(x)$, is defined as $BRP(x) = x_{\pi(0)} x_{\pi(1)} \cdots x_{\pi(2^\ell - 1)}$.

When $\ell = 3$, i.e., when $x = x_0 x_1 \cdots x_7$, $BRP(x) = x_0 x_4 x_2 x_6 x_1 x_5 x_3 x_7$. Namely, x_j is placed at the $\pi(j)$th position in $BRP(x)$ (the leftmost position is the 0th position).

Definition 2. For a sequence x of n packets, $SORT(x) = x_{s_0} x_{s_1} \cdots x_{s_{n-1}}$ denotes a sorted sequence according to the destination column. Namely, $SORT(x)$ is the sequence such that the destination column of x_{s_i} is farther than or the same as the destination column of x_{s_j} if $i > j$.

The following lemma shows a key property of brp sequences [IM2000]:

Lemma 1. Let $x = x_0 x_1 \cdots x_{n-1}$ be a sequence of length n where $n = 2^\ell$ for some integer ℓ and $z = z_0 z_1 \cdots z_{k-1}$ be its any subsequence of length k. Also let $d = 2^{2\ell_1}$ for some integer ℓ_1 and suppose that $k \geq d + 8\sqrt{d} + 8$. Then if w is any subsequence in $BRP(x)$ of length $\left\lceil \frac{dn}{k} \right\rceil$, w includes at most $d + 8\sqrt{d} + 8$ packets in z.

Suppose that x is a sequence of n packets, and among those n packets, $a_{k_1}, \cdots, a_2, a_1$ are $k_1 \geq d + 8\sqrt{d} + 8$ packets which have the same column destination. One can see that if k_1 packets of the same destination are truly even-distributed in the n packets, then any two neighboring packets among $a_{k_1}, \cdots, a_2, a_1$ should be $\frac{n}{k_1}$ positions apart. Lemma 1 says that if, by inserting *spaces* uniformly into the original brp sequence $BRP(SORT(x))$, the length of the sequence is extended to

$$\frac{d + 8\sqrt{d} + 8}{d} = 1 + O(\frac{1}{\sqrt{d}})$$

times larger, then the distance of any two neighboring packets among $a_{k_1}, \cdots, a_2, a_1$ is to be as

much as the average $\frac{n}{k}$. Namely, by using the operation based on the bit-reversal permutation (and with the supplementary *spacing* operation), we can change the order of packets from any scrambled order into the pseudo-random order which will guarantee no delay at the critical positions.

Now our basic idea may be clear: (i) Before routing packets toward their destination, any scrambled order of the packets in their flow is changed into the brp-order to control the injecting ratio of packets into the critical positions where serious delays can occur. (ii) Each packet moves to its final destination along its dimension-order path. More precisely, however, since the simple implementation of the "true" brp sequence construction must requires long time, we construct, what we call, a *quasi-brp* sequence, which is enough for our purpose as shown in the next section.

3.2 Parallel brp Construction

Let $A = a_1a_2\cdots a_m$ and $B = b_1b_2\cdots b_m$ be sequences of m packets. The $MERGE(A, B)$ is a sequence of $2m$ packets defined by

$$MERGE(A, B) = a_1b_1a_2b_2\cdots a_mb_m.$$

$MERGE(A, B, C)$ is defined similarly and is extended to any number of sequences. For example, let $C = c_1c_2\cdots c_m$ and $D = d_1d_2\cdots d_m$ be sequences of m packets. The $MERGE(A, B, C, D)$ is a sequence of $4m$ packets defined by

$$\begin{aligned} & MERGE(A, B, C, D) \\ = \; & a_1b_1c_1d_1a_2b_2c_2d_2\cdots a_mb_mc_md_m. \end{aligned}$$

For some positive constant $c < 1$, let $X = X_0X_1\cdots X_{\frac{1}{c}-1}$ be a sequence of n packets where each X_i has length cn. Then $QBRP_c(X)$ is defined as

$$\begin{aligned} & QBRP_c(X) \\ = \; & MERGE(BRP(SORT(X_0)), BRP(SORT(X_1)), \\ & BRP(SORT(X_2)), \ldots, BRP(SORT(X_{\frac{1}{c}-1}))). \end{aligned}$$

Consider some row of n processors. Then we provide a special portion of length cn for some fixed constant $c < 1$, called a *cn-tube*, at the right-end (see Figure 1). We can obtain $QBRP_c(X)$ from X of n packets using the linear array:

Lemma 2 [IM2000]. Suppose that a linear array of n processors, P_0 through P_{n-1}, is available. Also suppose that a sequence $X = x_0x_1\cdots x_{n-1}$ of n packets is initially placed on the n processors, one on one. Then $QBRP_c(X)$ can be constructed on the right-end cn-tube in time $n+2cn$ and with queue-size $\frac{2}{c}$ for any positive constant $c < 1$.

Proof. The following operation, PARALLEL, has $\frac{1}{c}$ sub-arrays in each row, where each sub-array consists of cn processors (see Figure 1 again) and the ith sub-array initially holds X_i. Again, in each row, we provide a special portion of length c^2n ($c < 1$), called *c^2n-tube*, at the right-end of each sub-array. PARALLEL consists of the following two stages:

Operation PARALLEL

(Stage 1) Within the ith sub-array, X_i is changed to its brp sequence of cn packets: For every i in parallel, (i) X_i is first changed in sorted order within the leftmost $cn - c^2n$ processors and then (ii) its brp sequence of the cn packets is eventually placed in the right-end c^2n-tube of the sub-array, i.e., the rightmost processor holds the head $\frac{1}{c}$ packets of the short brp of the cn packets, the second rightmost processor holds the next $\frac{1}{c}$ packets, and so on.

(Stage 2) cn packets of X_i now placed on the ith c^2n-tube are squeezed out to the right to be X_i's brp sequence of cn packets. There are $1/c$ short brp sequences, each of which derives from each X_i. The $1/c$ sequences are all shifted to the right in parallel until all the sequences fit the rightmost sub-array of cn processors (i.e., the right-end cn-tube). Finally, each processor in the right-end sub-array receives one packet from each sub-array or it holds $\frac{1}{c}$ packets of the quasi-brp sequence.

Since the length of the sub-array is cn, Stage 1 requires at most $2cn$ steps with queue-size $2/c$. Stage 2 takes n steps since the neighboring sequences do not overlap in this process, i.e., no delays occur in this stage. Also, the queue-size is at most $\frac{2}{c}$. □

3.3 Routing Scheme 1×2

In the following, for simplicity, we assume that the side-length n of the mesh is 2^ℓ for some integer

ℓ. An extension to a more general case needs details but is not hard. To shorten the path length, it is a natural idea to split the whole mesh into two sub-meshes, the left mesh and the right mesh. Figure 2 illustrates how each packet moves from its source to destination. Roughly speaking, a packet whose source is in the left half and whose destination in the right half, denoted by a *LR-packets*, moves on the so-called simple path. The path of an LL-packet is a bit more complicated; it first goes to the left end of the row, changes its direction 180 degrees, returns to its correct column position and then goes to its final position. Similarly for RL-packets and RR-packets. Note that the length of the path itself is at most $2n-2$, which does not lose any optimality. Also, the path of each packet is not affected by other packets, i.e., the algorithm is oblivious.

However, how each packet moves on the path is not so simple. Figure 2 illustrates the left half of the single row (the right half is similar). LR-packets on this row are once "packed" into their cn-tube located at the right-end of the left mesh, and LL-packets are once packed into the left-end cn-tube by performing `PARALLEL` in every direction (everything is the same for RL and RR packets, but executed within the right half mesh). Packing those packets into the cn-tubes can be done in parallel. Our first goal is to move those packets so that their original sequence X of $0.5n$ packets will be changed into $QBRP_c(X)$ in the cn-tubes. The packet movement till this moment is called *Phase 1*, which can be executed in $0.5n + 2cn$ steps by replacing n in Lemma 2 with $0.5n$.

In *Phase 2*, each packet comes out of the cn-tube and moves to its correct column position. As described below we need to insert one space between any two neighboring packets. In *Phase 3*, the packet moves on the correct column and finally reaches its destination.

As for Phase 2, from Lemma 2, we need to insert one space per $\lfloor \frac{d}{8\sqrt{d}+8} \rfloor$ packets of the brp sequence in order not to cause severe path-congestion in the critical positions. Furthermore, a $0.5n$ overhead must be charged for the second phase, which comes from the spacing operation again: Recall that, for example, at most $\frac{n}{2}$ LR packets on each row are packed into their cn-tube in Phase 1 and the (quasi-)brp sequence of those $\frac{n}{2}$ packets are constructed there. If the whole length of the brp sequence of packets is reduced from n to $\frac{n}{2}$, then the distance of any two neighboring packets which have the same destination column is also reduced to a half length from Lemma 1. Hence one has to extend the length of the brp sequence to be $1.0n$ again by inserting one space between any two packets, which imposes $0.5n$ overhead, called *space-overhead.*

Theorem 1 [IM2000]. There is an oblivious algorithm on 2D meshes which can correctly routes all packets within $(3.0+\frac{8\sqrt{d}+8}{d}+2c)n$ steps using queues of size $d+8\sqrt{d}+8+\frac{1}{c}$ for some constants c and d such that $c < \frac{1}{2}$ and $d = 2^{2\ell}$ for some integer ℓ.

Proof. Phase 1 requires $0.5n + 2cn$ steps. Phase 2 requires $1.0n+\frac{8\sqrt{d}+8}{d}$ steps, where $0.5n+\frac{8\sqrt{d}+8}{d}$ comes from the spacing. Phase 3 takes $1.5n$ steps, $0.5n$ steps for row routing and $1.0n$ steps for final column routing (see [IM2000] for more details). □

4 A $(2.5+\varepsilon)n$ Algorithm

4.1 Routing Scheme 2×2

In the previous $(3+\varepsilon)n$ algorithm, called `RS2`, we had to pay the $0.5n$ space-overhead in the second phase. In this section, it is shown that our new algorithm, denoted by `RS4`, can eliminate almost all the space-overhead. In the following, for simplicity, we may omit the description of the delay caused by constructing short brp sequences (denoted by cn previously).

Recall that `RS2` has to insert $0.5n$ spaces into every brp sequence constructed in Phase 2 since its length is $0.5n$ while $1.0n$ packets move through each column in Phase 3, i.e., `RS2` has to extend the length of the brp sequence to be $1.0n$ by spacing in order to avoid heavy path-congestion at the critical positions. However, alternatively, it is also true that if the number of packets flowing on each column can be decreased, then we only need to insert fewer spaces. Here is our basic idea: The whole $n \times n$ mesh is divided into four $\frac{n}{2} \times \frac{n}{2}$ sub-meshes, the top-left TL, the top-right TR, the bottom-left BL, and the bottom-right BR sub-meshes as illustrated in Figure 3. (i) All packets whose sources and destinations are

both within the upper half mesh (the lower lower half mesh), called *h-packets*, move along the same paths as the paths of RS2. For example, by using the first $0.5n$ steps, all LR h-packets which move from TL to TR are once packed into their cn-tubes located at the right-end of the left mesh, and then the packets go out of the cn-tubes in the brp order *without* inserting spaces by using the second $0.5n$ steps. Finally, those packets move horizontally and then vertically towards their correct positions in TR in the next n steps as before. On the other hand, (ii) all packets which move from the upper half mesh to the lower half mesh (from the lower half to the upper half), called *v-packets*, once move vertically into the lower (upper) half mesh, then move horizontally to their correct columns, and finally move vertically again to their final goal positions (see Figure 3). For example, all LR v-packets which move from TL to BR first move downward into the sub-mesh BL by using the first $0.5n$ steps and then move to the final goals in BR along the same paths as (i) in the next $2.0n$ steps. Note that the first horizontal action of (i) and the first vertical action of (ii) can be initiated at the same time and can be performed completely in parallel. Also, note that the final column movements within the upper half mesh and ones within the lower half mesh are independently executed, i.e., the number of packets flowing on each column can be regarded as $0.5n$. Thus, it would be possible for the four sub-mesh strategy to provide us an algorithm which runs in $(2.5+\varepsilon)n$ steps for small positive ε. Unfortunately, however, a simple implementation of the strategy does not work efficiently.

Take a look at a cn-tube, for example, at the right-end of TR in more detail. If we follow the stages as described above, then RR v-packets originally placed in BR start to enter into the cn-tube at the $0.5n$th step (i.e., right after their vertical movements), and the last packet of their brp sequence stays in the cn-tube until the $1.5n$th step in the worst case. Recall that, in parallel, LR h-packets originally placed in TL are coming from the left. As a worst example, if almost all the destinations of the LR h-packets are positions in the same cn-tube, then they arrive at the cn-tube roughly at the nth step, which causes heavy path-congestion there since the some RR v-packets are still moving within the cn-tube. Thus, the following special treatment is required only for the h-packets whose destinations are in the cn-tubes, called *tube-packets*: All the tube-packets are once moved to their intermediate positions which are placed on the same rows as their final destination rows but outside the cn-tubes, and then move horizontally to their final positions. Those intermediate positions are scattered evenly in the whole mesh except for the cn-tubes. Here is the rule (see Figure 4): The intermediate positions for cn tube-packets in the right-end on the top row are placed in the $(\frac{3n}{4}+1)$th column, the $(\frac{3n}{4}+\frac{0.25-c}{c}+1)$th column, $(\frac{3n}{4}+2\frac{0.25-c}{c}+1)$th column, and so on. The intermediate positions for cn tube-packets on the second top row are shifted (cyclically) one position to the right. Similarly for the other intermediate positions.

Note that the number of packets flowing on each column increases from the previous $0.5n$ to $0.5n+\frac{2cn}{1-4c}$. Thus, our main algorithm RS4 has to insert a small number of spaces between the brp sequences again.

Algorithm RS4

The algorithm consists of the following five phases, in each of which two tasks are performed at the same time. (i) One task is to move the h-packets which move within the upper half mesh (within the lower half mesh). (ii) The other is to move the v-packets which move from the upper (lower) half mesh to the lower (upper) half mesh:

Phase 1 ($(0.5+2c)n$ steps). (i) Everything is the same as Phase 1 in RS2 for h-packets. (ii) All the v-packets moving from the upper (lower) half mesh to the lower (upper) half mesh are shifted downward (upward) exactly $0.5n$ positions by using exactly $0.5n-1$ steps.

Phase 2 ($\max\left\{0.5n(1+O(\frac{1}{\sqrt{d}}))(\frac{1}{1-4c}),(0.5+2c)n\right\}$ steps). (i) h-packets start to get out of cn-tubes. Here one space is inserted per $\frac{1-4c}{4c}$ packets and furthermore, one space per $\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor$ packets. (ii) As for the v-packets, the same operation as Phase 1-(i) is performed.

Phase 3 ($0.5n(1+O(\frac{1}{\sqrt{d}}))(\frac{1}{1-4c})$ steps). (i) The same operation as Phase 3 in RS2 is performed for the h-packets but tube-packets are shifted horizontally and temporally enter into their intermediate columns defined by the above rule. (ii) Ev-

erything is the same as Phase 2-(i) for the v-packets.

Phase 4 ($0.5n$ steps). (i) All h-packets except for tube-packets move vertically to their final positions, and tube-packets move vertically to their intermediate positions. (ii) All v-packets are moved into their critical positions.

Phase 5 ($0.5n$ steps). (i) All tube-packets currently placed on their intermediate positions move horizontally to their final positions. (ii) All v-packets move vertically to their final positions.

Theorem 2. RS4 correctly routes all packets within $(2.5 + \max\left\{6c, \frac{8\sqrt{d}+8+4cd}{d-4cd}\right\})n$ steps using queues of size $d + 8\sqrt{d} + 8 + \frac{1}{c}$ for some constants c and d such that $c < \frac{1}{4}$ and $d = 2^{2\ell}$ for some integer ℓ.

Proof. Although we shall omit to prove why RS4 works in those time-steps, making just a change of parameters in the proof of Theorem 1 in [IM2000] leads us to this theorem. □

4.2 Worst Case

As mentioned before, the basic idea of this algorithm is to reduce the delay due to the brp by decreasing the number of processors into which a sequence of brp-ordered packets enter. To do so, however, we had to divide the whole mesh into four sub-meshes. Now take a look at some row, R, and a position, P, on R that is close to the right end (say, a distance of $0.1n$ from the right-end) in the upper-right sub-mesh. Then one can see that the number of packets which flow from left to right on this point P can be as large as approximately $2n$ ($0.5n$ ones originally on this row at the right-hand side which are to move in this sub-mesh, $0.5n$ ones at the left-hand side which are to move some columns to the right of P, and the same number of packets which come from the lower half). This means that we need at least $2n$ steps for those packets to go through this point and another $0.5n$ steps for the last packet to go to its vertical position. In other words, the $2.5n$ bound cannot be broken essentially.

To overcome this difficulty, one possible way is to further reduce the delay due to the brp by further decreasing the number of the target processors described above. This can be done, for example, by dividing the mesh into a larger number of sub-meshes of say 3×3. However, one can easily see that this will also increase the number of packets who share some single row as mentioned above. A simple extension of the current algorithm, for instance, will make the number of such packets $3n$, which is much worse than before. Thus the sub-mesh structure does not appear to continue to give us merits or 2×2 is probably the optimal. So, what we should do is, if possible, to reduce the brp delay without further subdividing the mesh. The next algorithm (which might seem to be more subdivided, but actually not) is probably the best we can do towards this goal.

5 A $(2.334 + \varepsilon)n$ Algorithm

In this section we shall show that the upper bound for oblivious routing is further reduced down to roughly $2.334n$. We provide several special portions in the mesh (see Figure 5). The whole $n \times n$ mesh is divided into three zones, called the *top-zone*, the *mid-zone*, and the *bottom-zone*, whose widths are all $\frac{n}{3}$. Furthermore, each zone is sliced vertically into three $\frac{n}{3} \times \frac{n}{3}$ sub-meshes. Again, we prepare special portions for the packing operation, but in this case we provide six cn-tubes on each row, i.e., each sub-array of length $\frac{n}{3}$ has two cn-tubes at both ends. Furthermore, each row is divided into $\frac{1}{c}$ sub-arrays of length cn, each of which has two cn^2-tubes at both ends. A packet is said to be *critical* if it is originally placed in the top-zone (bottom-zone) and its destination is in the bottom-zone (top-zone). The other packets are said to be *noncritical*.

Here is an outline of the path design: (i) All important packets in the top-zone (bottom-zone) first move exactly $\frac{2n}{3}$ positions vertically, and enter into the bottom-zone (top-zone). Then they move horizontally to their correct columns, and finally move vertically again to their final destinations. Note that the path length of each critical packet is at most $2.0n$, and more importantly, the length of the vertical sub-path of final movement is bounded by $\frac{n}{3}$. This shorter sub-path plays an important role in decreasing the delay by the brp construction in our main algorithm. (ii) Each noncritical packet first moves horizontally and then vertically. However, as de-

scribed below, the present brp construction is a little bit different from the previous one. Loosely speaking, every short brp sequence of length cn marches independently without merging into its quasi-brp sequence. Recall that the path-length of every noncritical packet is at most $(1+\frac{2}{3})n$ from the definition:

Algorithm RS

The algorithm executes two tasks at the same time: (1) One task is to move the noncritical packets, and (2) the other is to move the critical packets.

(1) All noncritical packets first move horizontally and then vertically. (1-1) The following is executed on every row in parallel: In each sub-array of length cn, the short brp of the cn packets is constructed and eventually placed in the right-end (or left-end according to the moving direction of packets) cn^2-tube of the sub-array in $2cn$ steps. Then the cn packets are squeezed out to the right to be its brp sequence. There are $\frac{1}{c}$ short brp sequences which should move to the right. The $\frac{1}{c}$ sequences are all shifted to the right in parallel. Recall that in the previous algorithm *RS4*, the left half $\frac{1}{2c}$ brp sequences (i.e., $cn \times \frac{1}{2c} = \frac{n}{2}$ packets) are packed at the right-end cn-tube of the left half mesh. However, in the present algorithm, each packet in its short brp goes rightward nearer to its destination column with keeping the relative position in the brp sequence. Suppose for example that a packet x heads for a position in some column in the ith sub-array X_i from the left and also suppose that x is the jth packet in its short brp. Then the packet x moves to the right-end cn^2-tube of the $(i-1)$th sub-array X_{i-1} with keeping the jth position in its brp (see Figure 6). If another packet in the same brp, say, y, heads for the different cn^2-tube, then the brp sequence must include one space instead of y. (1-2) The short brp now placed on the correct cn^2-tube starts to get out of the cn^2-tube. Here a lot of spaces are inserted into the brp since the length of the short brp is at most cn, but it is enough that the length of the brp is extended to be $\frac{2n}{3}$. The reason is as follows: Take a look at a position P in the bottom-zone. Then the number of packets which move downward from P is obviously at most $\frac{n}{3}$, furthermore, the number of packets which move upward from P is at most $\frac{2n}{3}$ since the bottom-zone does include no critical packets. The distance between any position in the mid-zone and the top row or the bottom row is also at most $\frac{2n}{3}$. As a result, if we extend the length of the brp to be $\frac{2n}{3}$, then we can avoid heavy path-congestion at the critical positions.

(2) As for critical packets, we only look at critical packets originally placed in the top-zone since critical packets from the bottom-zone to the top-zone move in the similar way but the vertical direction as the following description. (2-1) In the first phase each critical packet in the top-zone shifts downward exactly $0.5n$ positions in $0.5n$ steps. (2-2) In the second phase the following is executed on every row of the bottom-zone in parallel: Packets from the leftmost sub-array of length $\frac{n}{3}$ to the other two sub-arrays are once packed into the cn-tube located at the right-end of the leftmost sub-array, and in parallel packets moving from the center sub-array of length $\frac{n}{3}$ to the rightmost sub-array of length $\frac{n}{3}$ are once packed into the right-end cn-tube of the center sub-array by simulating **PARALLEL**. Everything but the moving direction is the same for packets from the rightmost sub-array to the center one, and packets from the center sub-array to the leftmost one. However, packets moving within the leftmost sub-array and packets moving within the rightmost sub-array postpone starting the same actions for $\frac{n}{3}$ steps, since during those steps noncritical packets originally placed in the leftmost sub-array of the bottom-zone may go though the rightmost sub-array. This $\frac{n}{3}$ overhead is not so bad since the path of the critical packet moving within the leftmost sub-array is pretty shorter than the others. (2-3) In the third phase, every critical packet starts to get out of cn-tubes. Here one space per $\left\lfloor \frac{d}{8\sqrt{d}+8} \right\rfloor$ packets is inserted. (2-4) The critical packets are shifted horizontally and eventually enter into their correct columns. (2-5) Finally the packets move vertically to their final positions.

Theorem 3. RS correctly routes all packets within $(2.334 + \frac{8\sqrt{d}+8}{d} + 4c)n$ steps using queues of size $2+16\sqrt{d}+16+\frac{4}{c^2}$ for some constants c and d such that $c < \frac{1}{2}$ and $d = 2^{2\ell}$ for some integer ℓ.

Proof. We shall give only a time analysis any bad collision of packets do not occur. The path-length of every noncritical packet is at most $\frac{5n}{3}$

and there is roughly $\frac{2n}{3}$ overhead which comes from the brp construction. In total, the packet can travel in $\frac{7n}{3}$ steps.

Take a look at a critical packet which starts from the leftmost sub-array in the top-zone to the leftmost sub-array in the bottom-zone. The packet first goes downward vertically and then arrives at the intermediate position of the bottom-zone in $\frac{2n}{3}$ steps. In the next $\frac{n}{3}$ steps, it does nothing and stays there. One can see that the packet has to move $\frac{2n}{3}$ positions further, $\frac{n}{3}$ positions horizontally and $\frac{n}{3}$ positions vertically, and $\frac{n}{3}$ is used for constructing brp sequence. As a result, we need $\frac{5n}{3}$ steps in total for routing the critical packet. Each critical packet moving from the leftmost sub-array in the top-zone to the rightmost sub-array in the bottom-zone moves along its path of at most length $2.0n$ and gets $\frac{n}{3}$ overhead for the brp construction. Thus we requires $\frac{7n}{3}$ steps. □

References

[Akl97] S.G. Akl, *Parallel Computation: Models and Methods*, Prentice Hall (1997).

[BH85] A. Borodin and J.E. Hopcroft, "Routing, merging, and sorting on parallel models of computation," *J. Computer and System Sciences* 30 (1985) 130-145.

[BRSU93] A. Borodin, P. Raghavan, B. Schieber and E. Upfal, "How much can hardware help routing?," In *Proc. ACM Symposium on Theory of Computing* (1993) 573-582.

[CLT96] D.D. Chinn, T. Leighton and M. Tompa, "Minimal adaptive routing on the mesh with bounded queue size," *Journal of Parallel and Distributed Computing* 34 (1996) 154-170.

[IM99] K. Iwama and E. Miyano, "An $O(\sqrt{N})$ oblivious routing algorithms for 2-D meshes of constant queue-size," In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms* (1999) 466-475.

[IM2000] K. Iwama and E. Miyano, " $(2.954+\varepsilon)n$ Oblivious Routing Algorithm on 2D Meshes" In *Proc. Symposium on Parallel Algorithms and Architectures* (2000) to appear.

[Kri91] D. Krizanc, "Oblivious routing with limited buffer capacity," *J. Computer and System Sciences* 43 (1991) 317-327.

[Lei90] F.T. Leighton, "Average case analysis of greedy routing algorithms on arrays," In *Proc. ACM Symposium on Parallel Algorithms and Architectures* (1990) 2-10.

[Lei92] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann (1992).

[LMT95] F.T. Leighton, F. Makedon and I. Tollis, "A $2n-2$ step algorithm for routing in an $n \times n$ array with constant queue sizes," *Algorithmica* 14 (1995) 291-304.

[RO92] S. Rajasekaran and R. Overholt, "Constant queue routing on a mesh," *J. Parallel and Distributed Computing* 15 (1992) 160-166.

[SCK97] J.F. Sibeyn, B.S. Chlebus and M. Kaufmann, "Deterministic permutation routing on meshes," *J. Algorithms* 22 (1997) 111-141.

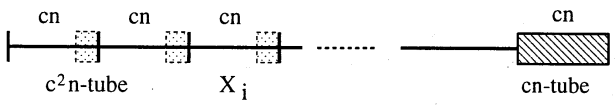

Figure 1: $\frac{1}{c}$ sub-arrays

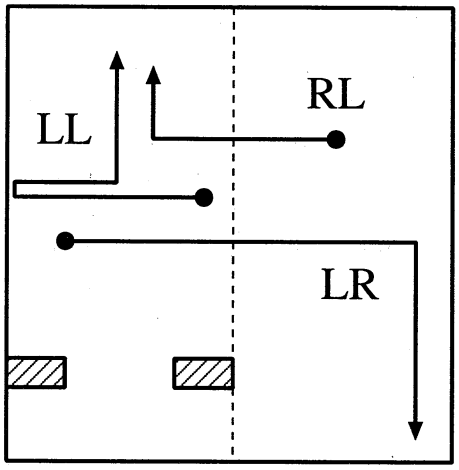

Figure 2: LL packets, LR packets, RL packets, RR packets

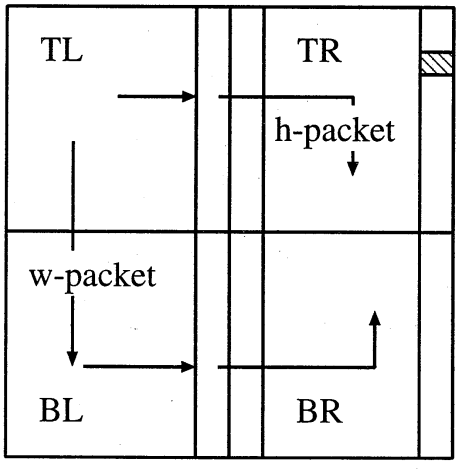

Figure 3: $\frac{n}{2} \times \frac{n}{2}$ sub-meshes, TL, TR, BL, and BR

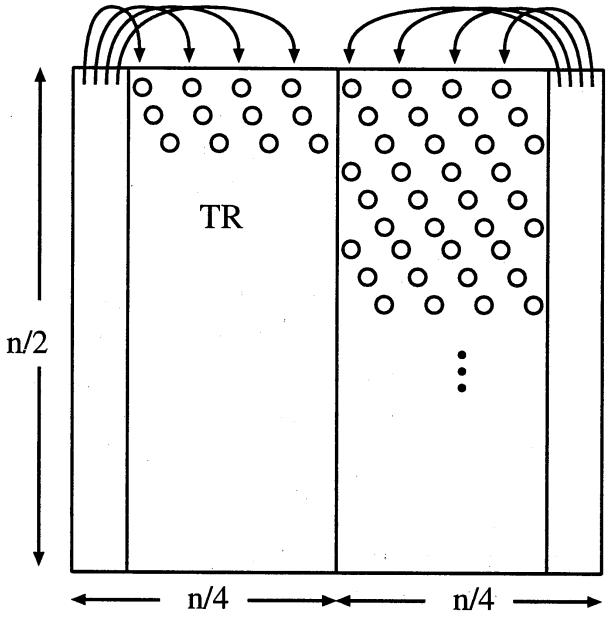

Figure 4: Intermediate positions for tube-packets

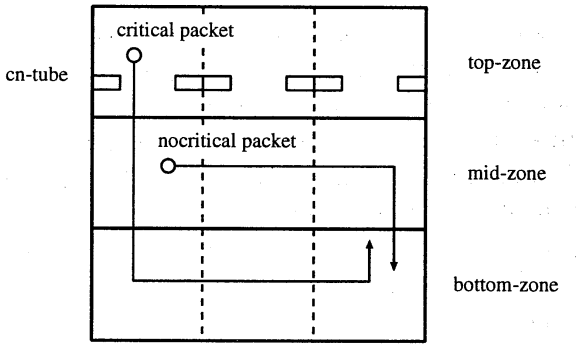

Figure 5: Top-zone, mid-zone, bottom-zone

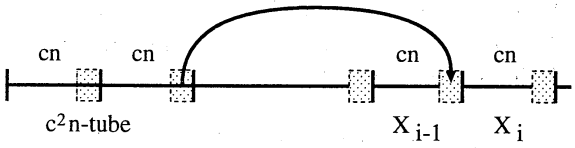

Figure 6: (1-1) in RS