アルゴリズム 73－3
（2000. 5. 19）

メッシュ上での高速な無情報ラウティングアルゴリズム

宮野 英次
九州芸術工科大学
〒815-8540 福岡市南区塩原 4-9-1
miyano@kyushu-id.ac.jp

岩間 一雄
京都大学大学院情報学研究科
〒606-8501 京都市左京区吉田本町
iwama@kuis.kyoto-u.ac.jp

あらまし:
本稿では，2次元，定数サイズキュー，$n \times n$ メッシュ計算機上での決定性オブリビアウスラウティングアルゴリズムを示す．本アルゴリズムは，任意の ε に対して，$(2.5+\varepsilon)n$ 時間で動作する．基本的には，ビット反転置換を利用しており，メッシュネットワークの直径の 1.25 倍程度の動作時間を実現している．

キーワード： 2次元メッシュ，無情報ラウティング，ビット反転置換

Fast Oblivious Routing Algorithms on Meshes

Eiji MIYANO
Kyushu Institute of Design
Fukuoka 815-8540, JAPAN
miyano@kyushu-id.ac.jp

Kazuo IWAMA
School of Informatics
Kyoto University
Kyoto 606-8501, JAPAN
iwama@kuis.kyoto-u.ac.jp

Abstract:
We present a deterministic, oblivious, permutation-routing algorithm on the $n \times n$ mesh of constant queue-size which runs in $(2.5+\varepsilon)n$ steps for any $\varepsilon > 0$. The algorithm basically makes use of the bit-reversal permutation and its running time is approximately 1.25 times as large as the network diameter of the mesh.

Key words: two-dimensional mesh, oblivious routing, bit-reversal permutation

1 Introduction

Mesh routing has received considerable attention for the last decades [GHKS98, Lei92, MS96, Tom94]. However, there still remain several open questions: For example, until recently, little had been known whether one can achieve an optimal, linear-time bound for *oblivious* permutation routing on two-dimensional (2D for short), $n \times n$ meshes of constant queue-size. Oblivious routing means that the path of each packet is determined only by its source and destination positions, not depending on other packets. In [IKM98, IM99a], Iwama and Miyano made a significant progress on this open question by giving an affirmative answer; they proposed a new algorithm, using derandomization based on the *bit-reversal permutation*, which was the first optimal (up to constant factor) algorithm for oblivious routing on 2D meshes.

As for the queue-size of each processor, they need only two in [IM99a], which is probably optimal. However, as for the time complexity, they only proved that their algorithm runs in linear time and it involves large constant factors partly due to making their proof simpler. Actually, the leading constant hidden in the big-O notation is at least 3000; it is obviously questionable if this algorithm can claim much practical importance.

In this paper we focus mainly on the running time and present a $(2.5 + \varepsilon)n$ oblivious algorithm which can route any permutation on 2D meshes including $n \times n$ processors for any positive constant ε. Its queue-size is a function of ε, which is quite moderate. The algorithm basically makes use of the bit-reversal permutation as before and its running time is approximately 1.25 times as large as the network diameter (or *distance bound*) of the mesh. Computation in each processor is not complicated either. Thus, this result shows that the oblivious algorithm based on the bit-reversal permutation does have practical merits.

It should be noted that any oblivious algorithm on n^2 processor networks needs $\Omega(n^2)$ steps if it is *pure* [Kri91], i.e., if packets must move whenever their next positions are empty. In order to obtain linear-time algorithms, therefore, some mechanism is needed which forces some packets "to wait" even if they can advance. In other words, we must intrinsically suffer from a certain kind of a delay, which makes oblivious routing laborious. The bit-reversal permutation in [IM99a] can control the movement of each packet almost perfectly to this goal. One drawback of this approach, however, is that we need a long path of processors for these operations. To create such long paths, the algorithm in [IM99a] suffers from serious detours and big constant factors. Fortunately, there is a standard technique to reduce this path length, i.e., simulating several processors by a single processor with a sacrifice of the queue-size (but still within some constant). This easily allows us to design an algorithm which runs in, say, $8n$ steps. However, it is usually a challenging topic to achieve a tighter bound on the distance bound and several new techniques are indispensable. Our present algorithm indeed needs a sequence of careful improvements. They include a tighter analysis and parallel construction of the bit-reversal permutation.

In what follows, we present three important lemmas for faster algorithms in Sections 2 and 3. In Section 4, we show a basic algorithm which runs in $(3.0 + \varepsilon)n$ steps. Its improvement to the $(2.5 + \varepsilon)n$ algorithm, given in Section 5, is moderate but involves several technical details.

2 Models

Our model is the standard, two-dimensional, $n \times n$ mesh. 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 sends at most one packet (at most one packet can flow on each link in each time-step) from Q_i to Q_{i+1}.

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

Definition 1 (see e.g., [Lei92]). 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 $BRP(x) = x_{\pi(0)}x_{\pi(1)} \cdots x_{\pi(2^t-1)}$.

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 will be often used in the rest of the paper:

Lemma 1. For any positive constant c, a linear array of n processors, P_0 through P_{n-1}, of queue-size K, can be simulated by a shorter linear array of $\lceil cn \rceil$ processors, Q_0 through $Q_{\lceil cn \rceil -1}$, of queue-size $\lceil \frac{1}{c} \rceil \cdot K$. (This is a standard technique. See, e.g., [MS96].)

3 Parallel brp Construction

We first give an easy (but slow) algorithm to get a rough idea of our final algorithm: (i) each packet first moves to the right, (ii) changes its direction at the right-end of each row, then (iii) moves back to its column position, (iv) turns there, and goes vertically to its final position (see Figure 1).

In each row 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). All packets in each row are once packed into this tube: The cn packets originally placed in the right-side cn-tube are once squeezed out of the cn-tube since the sorting operation for all the n packets are executed in the first array of length $n - cn$. The brp operation is simulated in the second array of length cn, where each processor of the second array finally holds $1/c$ packets, i.e., the rightmost processor holds the head $1/c$ packets of the brp sequence, the second rightmost processor holds the second head $1/c$ packets, and so on. These ideas can be implemented as follows: (i) At the first step, the cn packets in the cn-tube start to move leftward and then they are shifted exactly cn positions to the left in cn steps. At the same time, we start to execute the sorting operation by moving other packets sequentially from the leftmost packet. Then the cn packets of the cn-tube (currently placed outside the tube) meet the sorting operation progressing from the left and come back again into the tube. Since the farthest packet among all the n packets has the highest priority and never delay, it enter into the cn-tube exactly at the $(n - cn)$th step. Also, at the next step, the second farthest packet enter into the cn-tube, and so on. Finally, the last (nearest) packet arrives there at the $(n - cn + n)$th step. The brp can be implemented in cn steps since the length of the second array is now shorten to cn. Thus the brp sequence can be formed from any sequence in $2n$ steps as a whole by using the queues of size $2/c$ by Lemma 1 since $K = 2$. (ii) This brp sequence resumes moving rightward at the $(2n + 1)$st step, then changes its direction 180 degrees at the right-end of the row, and returns to its correct column position. Here three spaces are inserted between neighboring two packets as described in Section 2, and hence the last packet leaves the right-end at time $2n+4n$ since the total length of the sequence grows up to $4n$. (iii) It may furthermore move (at most) n positions to the left and (iv) at most n positions upward or downward before the packet arrives at its final position, so the total time complexity is at most $8n$.

Recall that it takes $2n$ steps to construct a brp sequence in (i) of the previous algorithm. This time complexity can be reduced to $(1+\varepsilon)n$ for some small constant ε as shown below. However, the resulting sequence is no longer a brp sequence but, what we call, a *quasi-brp* sequence, which will be shown to be enough for our purpose in the next section. The idea is to employ more parallelism.

Let $A = a_1a_2 \cdots a_n$ and $B = b_1b_2 \cdots b_n$ be sequences of packets. The $MRG(A, B)$ is a sequence of $2n$ packets defined by $MRG(A, B) = a_1b_1a_2b_2 \cdots a_nb_n$. $MRG(A, B, C)$ is defined similarly and is extended to any number of sequences. 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

$$MRG(BRP(SORT(X_0)), \ldots, BRP(SORT(X_{\frac{1}{c}-1}))).$$

Lemma 2. $QBRP_c(X)$ 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. Now our new operation, **PARALLEL**, has $\frac{1}{c}$ subarrays in each row, where each subarray consists of cn processors (see Figure 2) and the ith subarray 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 subarray. **PARALLEL** consists of the following two stages:

Operation PARALLEL

(Stage 1) Within the ith subarray, X_i is changed to its brp sequence by simulating the algorithm in Section 3 by replacing previous n with cn: 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 subarray, i.e., the rightmost processor holds the head $\frac{1}{c}$ packets of the short brp of the cn packets, the second 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 subarray of cn processors (i.e., the right-end cn-tube). Finally, each processor in the right-end subarray receives one packet from each subarray or it holds $\frac{1}{c}$ packets of the quasi-brp sequence.

Since the length of the subarray is cn, Stage 1 requires at most $2cn$ steps with queue-size $2/c$ by Lemma 1. 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 $\frac{2}{c}$ again by Lemma 1. □

Lemma 3. Let $x = x_0x_1 \cdots x_{n-1}$ be a sequence of length n where $n = 2^\ell$ for some integer ℓ and $z = z_0z_1 \cdots z_{k-1}$ 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 $\lceil \frac{dn}{k} \rceil$, w includes at most $d + 8\sqrt{d} + 8$ packets in z.

This lemma says that it is enough to extend a brp sequence

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

times in order that the distance of d packets from z is to be as much as the average.

4 $(3.0 + \varepsilon)n$ Algorithm

Now we have two major developments, parallel brp construction and the tighter analysis of brp sequences. The latter allows us to remove wasteful spaces among the brp sequences; Lemma 3 implies that it is enough to insert one space per $\left\lfloor \frac{d}{8\sqrt{d}+8} \right\rfloor$ packets of the brp sequence in order not to cause severe path-congestion in the critical positions (details will be furnished in the proof of Theorem 1). Thus the time complexity of the primitive algorithm can be now reduced to $(4 + \varepsilon)n$ for some small constant ε, namely, it takes $(1 + 2c)n$ steps (for the constant c described in Lemma 2) for constructing quasi-brp sequences by using PARALLEL as shown in Section 3.2, $(1 + O(\frac{1}{\sqrt{d}}))n$ steps for the last packet to get out of the cn-tube from Lemma 3, at most n steps for its moving leftward to its correct column, and at most n steps for its moving upward or downward to the final position.

$(4+\varepsilon)n$ is still larger than the optimal $2n$ by $(2+\varepsilon)n$. First, εn is used for constructing brp sequences and the related spacing. Among the other $2n$, n comes from the path length: If a packet is placed at the left-upper corner and its destination is the left-down corner, then it moves along the path of length $3n$ (see Figure 1), which is longer by n than the optimal $2n$. The final n is again related to the brp construction: This delay is obviously due to the amount of time needed to pack n packets into the cn-tube for merging. Let us call the first n overhead the *path-overhead*, and the second n overhead the *pack-overhead*.

To prevent the path-overhead, it is a natural idea to split the whole mesh into two sub-meshes, the left mesh and the right mesh. An LR packet which moves from the left half to the right half and an LL packet which moves inside the left half travel along the paths illustrated in Figure 3, respectively. (Similarly for RR and RL packets.) In each row of the left half, we provide two cn-tubes, one for LL packets and the other for LR packets. Packing those packets into the cn-tubes can be done in parallel (but we require only minor modifications as described below). Thus the path-overhead is completely eliminated since every packet moves along its path whose length is at most $2n$, and the pack-overhead will be one half since we only have to pack at most $\frac{n}{2}$ packets. Unfortunately, however, a new $0.5n$ overhead is created since 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 3. 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 another $0.5n$ overhead, called *space-overhead*. Thus the total time complexity becomes $(4+\varepsilon)n - 1.0n - 0.5n + 0.5n = (3+\varepsilon)n$.

Algorithm ROUT-B

The algorithm has the following four phases:

Phase 1 ($(0.5 + 2c)n$ steps). The following is executed on every row in parallel: Each row is divided into $\frac{1}{c}$ subarrays for some constant c where each subarray consists of cn processors. Note that each half mesh has only $\frac{1}{2c}$ subarrays. Then LR packets are once packed into cn-tubes located at the right-end of the left mesh, and in parallel LL packets are once packed into the left-end cn-tubes by performing PARALLEL in every direction (everything is the same for RL and RR packets, but executed within the right half mesh): As an example, consider the left mesh and LR packets. (i) The original sequence of LR packets in each subarray is changed to its brp sequence of length cn and placed in the right-end c^2n-tube of the subarray (see Figure 2 again). (ii) $\frac{1}{2c}$ brp sequences of LR packets are shifted rightward in parallel and merged into their quasi-brp sequence of length $0.5n$ in the right-end cn-tube of the left mesh.

Since the farthest LR packet of each subarray among LR packets never delay, it enter into the right c^2n-tube at the $(cn - c^2n)$th step. Also, at the next step, the second farthest packet enter into the c^2n-tube, and so on. Finally, at the $(2cn - c^2n)$th step, the nearest packet arrives at the c^2n-tube. The brp can be implemented in c^2n steps. Then, all the $\frac{1}{2c}$ short brp sequences are shifted to the right-end in parallel and are placed at the right-end cn-tube of the left mesh. As a result, this phase can be executed in $0.5n + 2cn$ steps and the queue-size is at most $\frac{1}{2c} \times 2 = \frac{1}{c}$.

Phase 2 ($1.0n(1 + O(\frac{1}{\sqrt{d}}))$ steps). Packets start to get out of cn-tubes. Here one space is inserted between any neighboring two packets and we further need to insert one space per $2\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor$ packets for the constant d described in Lemma 3 (i.e., one space per $\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor$ packets since one space has been already inserted between any two packets). Thus, the last packet of each cn-tube leaves at time $1.0n \times (\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor + 1)/\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor = 1.0n(1 + O(\frac{1}{\sqrt{d}}))$

The packets move as shown in Figure 3, namely, LL packets change the direction at the left-end of the left-half mesh and LR packets just continue to move rightward up to their correct columns.

There are two important observations: (i) After getting out of its own cn-tube, the sequence of LL packets may enter the other cn-tube if their destination columns are there (see Figure 4). Some LR packets may remain in the latter cn-tube and are now being squeezed. It may seem that this collision of LL and LR packets may cause some problem, but actually not. The reason is that both LL and LR streams include a space between two packets. Inside the cn-tube, one flow can be combined with the other using these spaces and we do not loose any speed of their movement. (ii) Since original $\frac{n}{2}$ packets in a single row are divided into LL and LR streams, at least one stream must include less than $\frac{n}{2}$ packets or must include spaces originally. This is not harmful either. We can regard these spaces as "null packets" whose destination column is the closest one when applying the sorting operation. Then these null packets are scattered in the brp sequence, which again does nothing harmful.

Phase 3 ($0.5n$ steps). All packets are shifted horizontally and eventually enter into their critical positions (some of packets have already entered into their critical positions in Phase 2). As the contention resolution rule, turning packets are always given a higher priority than straight-moving packets.

Phase 4 ($1.0n$ steps). All packets move vertically to their final positions.

Theorem 1. ROUT-B 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. We shall only prove that (i) there is no congestion at critical positions, and (ii) the queue-size is at most $d + 8\sqrt{d} + 8 + \frac{1}{c}$.

(i) The basic idea is similar to [IM99a], but a bit more complicated. Consider an arbitrary column j and denote a packet whose column destination is j by α_j. In each row, these α_j's come from both directions. However, it turns out that the number of directions is not important. Hence we can assume that all α_j's come from the left by increasing the number of rows, and can also assume that they move upward in the column. Let k_i be the number of α_j's in row i. Then k_i can be written as $k_i = k_{i,1} + k_{i,2} + \cdots + k_{i,1/2c}$, where $k_{i,\ell}$ is the number of α_j's which exist in the ℓth cn-tube before merged.

Take a look at the first cn-tube. There are $k_{i,1}$ α_j's, denoted by $\alpha_{j,1}$, in that tube. Since they become a subsequence of length $k_{i,1}$ after sorted, there are at most $d{+}8\sqrt{d}{+}8$ $\alpha_{j,1}$'s in any brp subsequence of length $\frac{d \cdot cn}{k_{i,1}}$ by applying Lemma 3. Recall that the small brp sequences are merged into the quasi-brp sequence. Since $\frac{1}{2c}$ brp sequences are merged, $\frac{1}{2c}-1$ packets are inserted between any two $\alpha_{j,1}$'s. Furthermore, brp sequences are extended by inserting one space between any two packets and one space per $2\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor$ packets. Therefore, in the final quasi-brp sequence, there are at most $d+8\sqrt{d}+8$ $\alpha_{j,1}$'s in any subsequence of length

$$\begin{aligned} & \frac{dcn\cdot 2}{k_{1,1}\cdot 2c}\left\{\left(\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor+1\right)/\left\lfloor\frac{d}{8\sqrt{d}+8}\right\rfloor\right\} \\ \geq\ & \frac{(d+\sqrt{d}+8)n}{k_{1,1}}. \end{aligned}$$

Similar results are also obtained for other $\alpha_{j,\ell}$'s.

Now consider a processor $P_{i,j}$ at the cross-point of the ith row and jth column. Let $N=d+8\sqrt{d}+8$. Then, the number of $\alpha_{j,\ell}$'s which $P_{i,j}$ receives from the ℓth cn-tube during an arbitrary time-window Δ_i of $\frac{N\cdot n}{k_i}$ steps is at most

$$\left(\frac{N\cdot n}{k_i}/\frac{N\cdot n}{k_{i,\ell}}\right)\times N=N\frac{k_{i,\ell}}{k_i}.$$

Thus the total number of α_j's which $P_{i,j}$ receives from all cn-tubes during Δ_i is at most

$$N\frac{k_{i,1}+k_{i,2}+\cdots+k_{i,1/2c}}{k_i}=N$$

since $k_i=k_{i,1}+k_{i,2}+\cdots+k_{i,1/2c}$. If none of those N packets α_j's can move upward through the jth column at some step, then there must be a packet which is now ready to enter the jth column by making a turn at some upper position than $P_{i,j}$, and which can enter the column in the next time-step. Let us call such a packet a *blocking packet against α_j's*. In the following, we shall show that the total number of α_m's $(1\leq m\leq i-1)$ which $P_{1,j}$ through $P_{i-1,j}$ can receive during the window Δ_i is at most $\frac{Nn}{k_i}-N$. In other words, there must be at least N time-slots such that no packets flow on the jth column within $\frac{Nn}{k_i}-N$ steps. Since there are no blocking packets at those time-slots, the N α_j's currently held in $P_{i,j}$ can enter the column during the window Δ_i.

Since the similar argument implies that $P_{m,j}$ can receive at most N α_j's in $\frac{N\cdot n}{k_m}$ steps for each $m<i$, the number of α_j's which $P_{m,j}$ receives during Δ_i of $\frac{N\cdot n}{k_i}$ steps is at most

$$\left(\frac{N\cdot n}{k_i}/\frac{N\cdot n}{k_m}\right)\times N=\frac{Nk_m}{k_i}.$$

Hence, the total number of α_j's which $P_{1,j}$ through $P_{i-1,j}$ can receive is at most

$$\frac{N(k_1+k_2+\cdots+k_{i-1})}{k_i}\leq\frac{N\cdot n}{k_i}-N$$

since $k_1+k_2+\cdots+k_{i-1}\leq n-k_i$. The same argument can apply for any column j $(1\leq j\leq n)$ and for any window Δ_i $(1\leq i\leq n)$. As a result, any delay does not happen in the column routing phase.

(ii) Since each queue at any column edge becomes empty at least once every $\frac{N\cdot n}{k_i}$ steps as shown above, the queue-size does not exceed size N. Also, each queue at any row edge is bounded from above by $\frac{1}{c}$. Thus each processor holds at most $N+\frac{1}{c}$ packets at the same time in its queue. □

5 $(2.5+\varepsilon)n$ Algorithm

In the previous section, we have improved the time complexity down to $(3+\varepsilon)n$ by reducing the path-overhead from $3n$ of the primitive algorithm to $2n$, and by removing the pack-overhead, from the previous $1.0n$ to $0.5n$. However, in compensation for those improvements, we had to pay the $0.5n$ space-overhead in the second phase of `ROUT-B`. In this section, it is shown that our main algorithm, denoted by `ROUT-Q`, 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 `ROUT-B` has to insert $0.5n$ spaces into every brp sequence constructed in Phase 1 since its length is $0.5n$ while $1.0n$ packets move through each column in Phase 4, i.e., `ROUT-B` 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 5. (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 `ROUT-B`. 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 *w-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 5). For example, all LR w-packets which move from TL to BR first move downward into the submesh 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 w-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 w-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 6): 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 `ROUT-Q` has to insert a small number of spaces between the brp sequences again.

Algorithm `ROUT-Q`

The algorithm has 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 w-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 `ROUT-B` for h-packets. (ii) All the w-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 ($\mathtt{max}\left\{0.5n(1+O(\frac{1}{\sqrt{d}}))(\frac{1}{1-4c}),\ (0.5+2c)n\right\}$ 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 w-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 `ROUT-B` 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) Everything is the same as Phase 2-(i) for the w-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 w-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 w-packets move vertically to their final positions.

Theorem 2. ROUT-Q 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 ℓ.

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

References

[GHKS98] M.D. Grammatikakis, D.F. Hsu, M. Kraetzl, J.F. Sibeyn, "Packet routing in fixed-connection networks: A survey," *JPDC* 54 (1998) 77-132.

[IKM98] K. Iwama, Y. Kambayashi and E. Miyano, "New bounds for oblivious mesh routing," In *Proc. ESA98* (1998) 295-306.

[IM99a] K. Iwama and E. Miyano, "An $O(\sqrt{N})$ oblivious routing algorithms for 2-D meshes of constant queue-size," In *Proc. SODA99* (1999) 466-475.

[Kri91] D. Krizanc, "Oblivious routing with limited buffer capacity," *JCSS* 43 (1991) 317-327.

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

[MS96] R. Miller and Q.F. Stout, *Parallel algorithms for regular architectures: meshes and pyramids,* The MIT Press (1996).

[Tom94] M. Tompa, *Lecture notes on message routing in parallel machines,* U of Washington (1994).

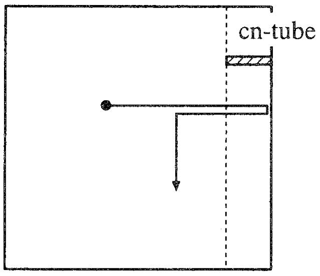

Figure 1: Paths and cn-tubes

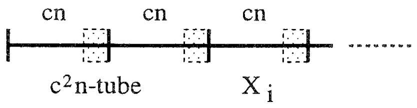

Figure 2: $\frac{1}{c}$ subarrays

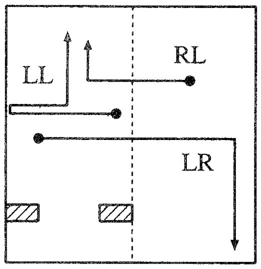

Figure 3: LL, LR, RL, and RR packets

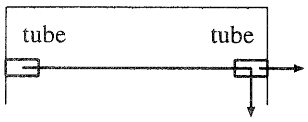

Figure 4: Observation-(i) in Phase 2 of ROUT-B

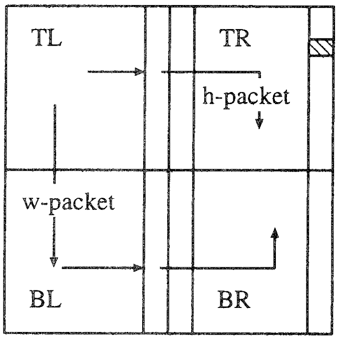

Figure 5: $\frac{n}{2}\times\frac{n}{2}$ sub-meshes, TL, TR, BL, and BR

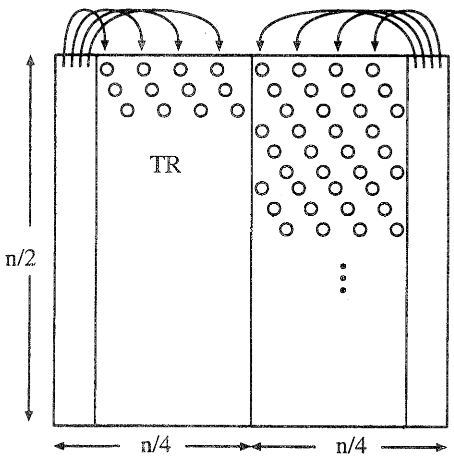

Figure 6: Intermediate positions for tube-packets