ELSEVIER

Theoretical Computer Science 261 (2001) 227–239

Theoretical Computer Science

www.elsevier.com/locate/tcs

Efficient randomized routing algorithms on the two-dimensional mesh of buses

Kazuo Iwama[a,1], Eiji Miyano[b,2,*], Satoshi Tajima[c], Hisao Tamaki[d,3]

[a] *School of Informatics, Kyoto University, Kyoto 606-8501, Japan*
[b] *Kyushu Institute of Design, Fukuoka 815-8540, Japan*
[c] *Systems and Software Research Laboratories, Research and Development Center, Toshiba Corporation, Kawasaki 210-8501, Japan*
[d] *Department of Computer Science, Meiji University, Kawasaki 214-8571, Japan*

Accepted 14 March 2000

Abstract

The mesh of buses (MBUS) is a parallel computation model which consists of $n \times n$ processors, n row buses and n column buses but no local connections between two neighboring processors. As for deterministic (permutation) routing on MBUSs, the known $1.5n$ upper bound appears to be hard to improve. Also, the information theoretic lower bound for any type of MBUS routing is $1.0n$. In this paper, we present two randomized algorithms for MBUS routing. One of them runs in $1.4375n + o(n)$ steps with high probability. The other runs $1.25n + o(n)$ steps also with high probability but needs more local computation.

Keywords: Permutation routing; Mesh of buses; Randomized algorithm

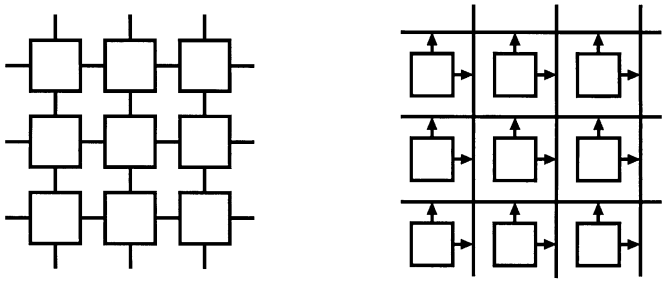
1. Introduction

The two-dimensional mesh is widely considered to be a promising parallel architecture in its scalability [12, 14]. In this architecture, processors are naturally placed at intersections of horizontal and vertical grids, while there can be two different types of communication links: The first type is shown in Fig. 1(1). Each processor is connected

* Corresponding author. Tel.:/Fax: +81-92-553-4432.
E-mail addresses: iwama@kuis.kyoto-u.ac.jp (K. Iwama), miyano@kyushu-id.ac.jp (E. Miyano), tajima@ssel.toshiba.co.jp (S. Tajima), tamaki@cs.meiji.ac.jp (H. Tamaki).

[1] Supported in part by Scientific Research Grant, Ministry of Japan, 10558044, 09480055 and 08244105, and Kayamori Foundation of Informational Science Advancement, Japan.
[2] Supported in part by Scientific Research Grant, Ministry of Japan, 10780198.
[3] Supported in part by Scientific Research Grant, Ministry of Japan, 10680365 and 10205225.

0304-3975/01/$ - see front matter
PII: S0304-3975(00)00141-9



(1) (2)

Fig. 1. (1) MC, (2) MBUS.

to its four neighbors and such a system is called a *mesh-connected computer* (*MC*). Fig. 1(2) shows the second type: Each processor is connected to a couple of (row and column) buses. The system is then called a *mesh of buses* (*MBUS*).

Permutation routing (simply *routing* in this paper) is apparently a basic form of communication among the processors: The input is given by n^2 packets that are initially held by the $n \times n$ processors, one by each. Routing requires that all n^2 such packets be moved to their destinations that are mutually distinct. In the case of MCs, a $2n-2$ lower bound comes from a fundamental nature of the model, i.e., the physical distance between the farthest two processors. Also, the same $2n-2$ upper bound can be achieved by an elementary algorithm based on the *dimension-order* strategy [12, 16]. Thus, there remains little for further research in the case of MCs. (This is not true for limited queue-size as mentioned later.) In the case of MBUSs, on the other hand, there is a wide margin between the known upper and lower bounds. First of all, unlike the case of MCs, the dimension-order strategy only gives us a poor algorithm which takes trivial $2n$ steps. The best upper and lower bounds known are $1.5n$ and $(1-\varepsilon)n$, respectively [8]. The $1.5n$ bound appears to be hard to improve; it is also known to be a (tight) lower bound if we impose the so-called "source-oblivious" condition [5].

The main purpose of this paper is to decrease this $1.5n$ upper bound by allowing randomization. Two randomized algorithms are given: One of them runs in $1.4375n+o(n)$ steps with high probability. The other runs in $1.25n+o(n)$ steps also with high probability but needs more local computation. The idea is an efficient use of the buses and a reduction of packet collisions. Consider, for example, the (deterministic) dimension-order routing where each packet first moves horizontally (in the order of original position) using the first n steps and then moves vertically (in the order of destination position) in the second n steps. One can see that column buses are completely idle in the first n steps and so are row buses afterwards in this algorithm.

A simple attempt to avoid this inefficiency is to try to move vertically the first n packets immediately after they move horizontally. If those n packets go to all different n columns, i.e., they have all different column destinations, then we can do this in a

single step without collision. If collision happens in some column, then we can use randomization techniques to resolve the collision. If few collisions occur, then we might achieve an approximately $1.0n$ upper bound. Unfortunately, however, this observation is too optimistic. Some experiments show that a lot of collisions occur for even random permutation. It seems that this approach gives us no better bounds than $3n$; it is much worse than the deterministic version. Thus, an efficient use of buses tends to imply more collisions. Our first algorithm avoids this difficulty in a tricky way. The second one is based on a novel use of the technique that allows us to generate many pseudo-random numbers deterministically from a few random numbers.

Research on mesh routing has a long history and has a huge amount of literature. Nevertheless there still remain a lot of unknowns. For example, our knowledge on the three-dimensional (3-D) mesh is much weaker than the 2-D mesh. Recently, it is shown [6] that minimum-bending oblivious routing on the 3-D mesh needs $\Omega(N^{2/3})$ steps that is much *more* than $O(N^{1/2})$ for the 2-D mesh (N is the total number of processors). It is not known either whether we can improve $O(n^2)$ upper bounds substantially for 2-D oblivious routing on MCs with *constant queue-size* [3, 11]. (However, very recently there was a progress, see [4, 7].) There is also a gap between the known upper and lower bounds, $(1+\varepsilon)n + o(n)$ and $0.691n$, respectively, for 2-D routing on the mesh equipped with both buses and local connections [2, 13]. Again, however, there is a recent result which shows that we can improve upper bounds greatly on this model using randomization [10].

2. Models and problems

An MBUS consists of n^2 processors, $P_{i,j}$, $1 \leqslant i, j \leqslant n$, and n row and n column buses. $P_{i,j}$ is connected to the ith row bus and the jth column bus. The problem of *permutation routing* on the MBUS is defined as follows: The input is given by n^2 *packets* that are initially held by the n^2 processors, one by each. Each packet, (s, d, σ), consists of three portions; s is a *source* address that shows the initial position of the packet, d is a *destination* address that specifies the processor to which the packet should be moved, and σ is a data portion that is not important in this paper. No two packets have the same destination address. Routing requires that all n^2 such packets be moved to their correct destinations.

Our discussion throughout this paper is based on the following four rules on the model: (i) We follow the common practice on how to measure the running time of MBUSs: One-step computation of each processor P consists of (a) reading the current data on both row and column buses P is connected to, (b) executing arbitrarily complicated instructions using the local memory and (c) if necessary, writing data to the row and/or column buses. The written data will be read in the next step. (ii) The queue-size is not bounded, namely, an arbitrary number of packets can stay on a single processor temporarily. (iii) What can be written on the buses by the processor P must be the packet originally given to P as its input packet or one of the packets that have

been read so far by P from its row or column bus. (Nothing other than packets can be written.) This means that any kind of data compression is not allowed. (iv) We allow the simultaneous write. However, if two or more packets are written on the same bus simultaneously, then a special value flows on the bus, which has no information other than collision.

As mentioned in the previous section, the $2n$-step dimension-order routing moves horizontally the leftmost n packets initially placed on $P_{1,1}, P_{2,1}, \ldots, P_{n,1}$ in step 1, $P_{1,2}, P_{2,2}, \ldots, P_{n,2}$ in step 2 and so on. Namely, packets are moved in their "source-order" in this first stage. In the second stage, n packets whose destinations are the uppermost n processors, $P_{1,1}, P_{1,2}, \ldots, P_{1,n}$, are moved vertically in step 1, then, $P_{2,1}, P_{2,2}, \ldots, P_{2,n}$, and so on. Thus they are moved in the "destination-order" regardless of their current positions. It should be noted that this destination-order transmission can only be used after all the packets have moved horizontally. That is why column buses are completely idle in the first stage. If we do not wait, then we have to give up the destination-order transmission and encounter the more serious problem, i.e., packet collisions, as described in Section 1.

The $1.5n$-step algorithm, called *DR4* from now on, reduces the number of first-stage steps from n to $0.5n$ as follows: The whole $n \times n$ plane is divided into four $0.5n \times 0.5n$ subplanes. Packets in the upper-left $0.5n \times 0.5n$ and the lower-right $0.5n \times 0.5n$ subplanes are moved horizontally and those in the upper-right and the lower-left subplanes vertically, both in the source-order. Thus, all the buses are used in the first stage, which reduces the computation time by one half. The second stage is almost the same as before.

3. $1.4375n + o(n)$ randomized algorithm

Note that there are n^2 packets, $2n$ buses and each packet has to ride on buses twice (in general). Thus $2 \times n^2/2n = n$ steps are needed even if we have no idle buses. In the $1.5n$-step algorithm *DR4*, Stage 1 has no idle buses. However, it is impossible to improve Stage 2 since we can create an instance as an "adversary" which leaves n packets on a single bus after Stage 1. (Detours might help but it seems difficult to design an algorithm that exploits the possibility.)

Our first randomized algorithm, *RR*, is based on *DR4*. The basic idea is as follows: (i) We should avoid, for any instance, the bad case where n packets are gathered on a single bus after Stage 1. (ii) In other words, we should distribute packets evenly so that each single bus has approximately $0.5n$ packets at Stage 2. (iii) Then it is not so hard to design a randomized algorithm for Stage 2 that needs more than optimal $0.5n$ steps but some $0.75n$ steps are enough. (iv) In order to accomplish the even distribution in (ii), we now have to give up the very efficient Stage 1 of *DR4*. Some loss of performance is inevitable, but if we can keep it less than $0.75n$ steps, then the $1.5n$ bound in total can be improved.

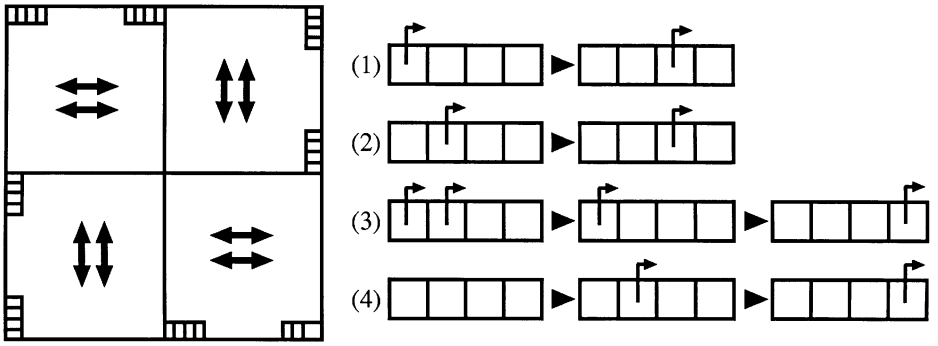


Fig. 2. Two packets written on the row bus in Stage 1–1.

Algorithm: *RR*

Stage 1: The whole plane is divided into four subplanes as *DR4*. This stage consists of Stages 1–1 and 1–2.

Stage 1–1: For a while, we only look at the upper-left subplane. The $0.5n$ processors in each row are divided into $0.125n$ *blocks*; each block includes four consecutive processors. For example, $(P_{1,1}, P_{1,2}, P_{1,3}, P_{1,4})$ is the first block of row 1, $(P_{1,5}, P_{1,6}, P_{1,7}, P_{1,8})$ is the second block and so on. Now from $j = 1, 2, \ldots,$ through $0.125n$, namely for each block from left to right, the following Phases 1 and 2 are executed. The same operation is executed on each row bus in parallel; the description below is for row i:

Phase 1: The leftmost two processors of block j, i.e., $P_{i,4j-3}$ and $P_{i,4j-2}$, write their initial packets on the row bus with probability $\frac{1}{2}$.

Phase 2: One of the following four operations is selected due to the result of Phase 1. Note that all the processors on the bus can figure out which case occurred.

1. If the packet whose source address is $(i, 4j-3)$ was on the bus, i.e., if only $P_{i,4j-3}$ wrote the packet, then $P_{i,4j-1}$ writes its initial packet on the row bus. Go to the next block (see Fig. 2(1)).
2. If only $P_{i,4j-2}$ wrote the packet, then $P_{i,4j-1}$ writes its initial packet on the row bus. Go to the next block (see Fig. 2(2)).
3. If collision occurred, i.e., if both $P_{i,4j-3}$ and $P_{i,4j-2}$ wrote the packets, then $P_{i,4j-3}$ again writes its packet and then $P_{i,4j}$ writes its packet on the row bus. Thus we need three steps in this (and next) case. Go to the next block (see Fig. 2(3)).
4. If the bus was idle, i.e., if neither $P_{i,4j-3}$ nor $P_{i,4j-2}$ wrote the packets, then $P_{i,4j-2}$ writes its packet and then $P_{i,4j}$ writes its packet on the row bus. Go to the next block (see Fig. 2(4)).

Thus, exactly two out of four packets in each block are moved horizontally. The remaining two packets, called *m-packets*, are moved vertically in Stage 1–2.

Stage 1–2: Now the $0.5n$ processors in each *column* are divided into $0.25n$ *blocks*, i.e., each block has two processors. From $i = 1, 2, \ldots,$ through $0.25n$, namely for each block from top to bottom, Phases 1 and 2 are executed.

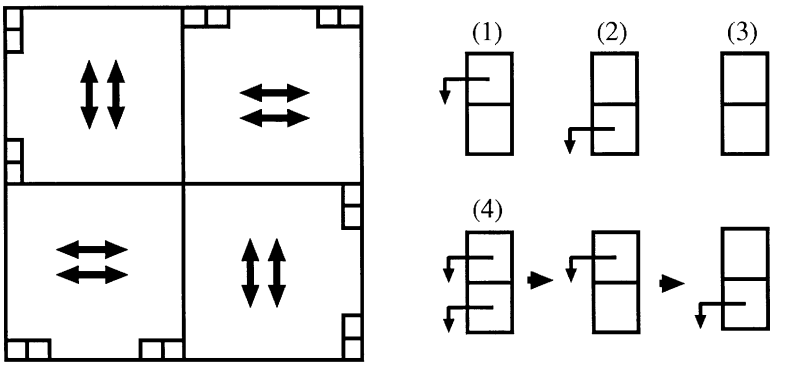


Fig. 3. The remaining packets written on the column bus in Stage 1–2.

Phase 1: Each of the two processors, $P_{2i-1,j}$ and $P_{2i,j}$, in block i writes its m-packet (if any).

Phase 2: Again one of the four operations is selected:

1. If only $P_{2i-1,j}$ wrote the packet, then go to the next block (see Fig. 3(1)).
2. If only $P_{2i,j}$ wrote the packet, then go to the next block (see Fig. 3(2)).
3. If neither $P_{2i-1,j}$ nor $P_{2i,j}$ wrote the packet, then go to the next block (see Fig. 3(3)).
4. If both $P_{2i-1,j}$ and $P_{2i,j}$ wrote the packets, then $P_{2i-1,j}$ writes its packet and then $P_{2i,j}$ writes its packet on the column bus. Go to the next block (see Fig. 3(4)).

This concludes Stage 1 for the upper-left subplane. The algorithm is exactly the same for the lower-right subplane. As for the upper-right and the lower-left subplanes, we just switch "rows" and "columns", i.e., Stage 1–1 uses columns and Stage 1–2 rows.

Stage 2: Every packet has already moved to its row or column destination. Thus the situation is the same as at the beginning of Stage 2 of *DR4*. The difference is that the number of packets held by processors on each single bus is evenly distributed, i.e., is about $0.5n$ (the proof is given later). Let us look at some single bus, say, row 1. The basic idea of Stage 2 is to use the destination-order counterpart of Stage 1–2. Namely, at the first step, if a processor has a packet whose destination is $(1,1)$ or $(1,2)$ then it writes that packet on the row bus. If no collision occurs, i.e., if at most one of the two packets exists on this row, then we move forward. Otherwise, two more steps are used for sending each of those collided packets.

Unfortunately, this algorithm does not work. The reason is that the two packets whose destinations are $(1,1)$ and $(1,2)$ may be held by some single processor. If this happens, then that processor puts either one on the bus but other processors following the above algorithm assume that there is only one packet on this row whose destination is in the first block.

To solve this problem, we introduce a "special packet" (SP) whose purpose is to broadcast special information. (If one does not like to use a packet for such a special

purpose, then we can give another algorithm whose performance is a little bit worse than the current one. See the end of this section.) As an *SP*, we use the packet whose destination is (n,n). At the beginning of Stage 2, we introduce two extra steps to let all the processors know this *SP*. Now from $i=1,2,\ldots,$ through $0.5n$, i.e., for each block from left to right, the following two phases are executed.

Phase 1. Let Q_1 and Q_2 be packets whose destinations are in block i. For each processor P on this row, if P holds either Q_1 or Q_2, then P writes that packet on the bus. If P holds both Q_1 and Q_2, then P writes the *SP*.

Phase 2. One of the following four operations is selected:

1. If Q_1 flowed on the bus in Phase 1, then all the processors move on to the next iteration, namely, for $i+1$.
2. If Q_2 flowed, then all the processors move on to the next iteration also.
3. If nothing flowed, then all the processors move on to the next iteration also.
4. If *SP* flowed or collision occurred, then we need two more steps; the processor holding Q_1 writes it first and then the processor holding Q_2 follows.

We have not yet stated when Stage 2 should be started. Our design of *RR* determined to start Stage 2 at some fixed step that is no later than $0.6875n+o(n)$. At this moment all processors have finished Stage 1 with high probability (see the proof of Theorem 1). However, there is a slight chance that some processor is still executing Stage 1. In this case, some unexpected data-collision will happen in Stage 2 and the algorithm fails. To improve it so as not to fail is possible but needs some technical details. For this purpose, we can use the *SP* again, which can also be avoided at the expense of extra $o(n)$ steps (omitted).

Theorem 1. *With high probability, RR can rout any instance within* $1.4375n+o(n)$ *steps.*

Proof. We first calculate the expected number of steps each stage takes. Then it is proved that the probability that *RR* takes essentially more steps than the average is very low.

Stage 1–1: See Phase 2. The probabilities for cases 1–4 are all the same, i.e., $\frac{1}{4}$. Cases 1 and 2 take two steps and cases 3 and 4 take three steps. Therefore, it takes 2.5 steps for each block on average or $0.3125n$ steps for $0.125n$ blocks.

Stage 1–2: Consider an arbitrary column in the upper-left subplane. It is not hard to see that each processor on this column holds an m-packet with probability $\frac{1}{2}$ and furthermore that this occurs independently between any pair of processors on this column. In Phase 2, cases 1–3 take one step and case 4 three steps. Hence, we need 1.5 steps on average per block or $0.375n$ steps for $0.25n$ blocks.

Stage 2: Let us calculate the probability of cases 1–4 in Phase 2 where we have to be a bit more careful than before.

Case a: Q_1 and Q_2 come from different blocks of Stage 1–1. Then one can easily see that the probabilities for cases 1–4 are the same, i.e., $\frac{1}{4}$ for each.

Case b: Q_1 and Q_2 come from the same block. Then we should check many different possibilities, such as coming from the top two positions, from the middle two positions, and so on. However, it turns out that in any possibility, the probability for case 4 is at most $\frac{1}{4}$ ($\frac{1}{4}$ or 0, in fact).

Recall that cases 1–3 need one step and case 4 needs three steps. Hence, we need at most 1.5 steps on average per block or at most $0.75n$ steps for $0.5n$ blocks.

Now we shall evaluate the probability for bad behaviors using the Chernoff bound [1].

Lemma 1. *Let $X_1, X_2, \ldots, X_n$ be independent Bernoulli trials having binomial distribution $B(n, p)$, $0 < p < 1$. Let $X = \sum_{i=1}^{n} X_i$, $\mu = E[X]$. Then, for any $0 < \varepsilon < 1$,*

$$\Pr[X > (1+\varepsilon)\mu] < \exp\left(\frac{-\varepsilon^2\mu}{3}\right).$$

Stage 1-1: For a time being, we only consider some single bus. For block i, let $X_i = 2$ when the case 1 or 2 occurs and $X_i = 3$ otherwise. The $X = \sum X_i$ has a binomial distribution $B(0.125n, \frac{1}{2})$, and $\mu = E[X] = 0.3125n$. Apply the Chernoff bound with $\varepsilon = c_1\sqrt{n \ln n}/\mu$ for $c_1 > 0$. Then,

$$\Pr[X > 0.3125n + c_1\sqrt{n \ln n}] < \exp\left(-\frac{16}{15}c_1^2 \ln n\right) = n^{-d_1}$$

for some $d_1 > 0$. Namely, the number of steps for Stage 1–1 is at most $0.3125n + c_1\sqrt{n \ln n}$ with probability $1 - n^{-d_1}$.

Stage 1–2: Our analysis is almost the same. Let $X_i = 1$ when one step is needed and $X_i = 3$ when three steps are needed, then the number of steps for this stage is at most $0.375n + c_2\sqrt{n \ln n}$ with probability $1 - n^{-d_2}$. Thus Stage 1 takes at most $0.6875n + o(n)$ steps with high probability.

Stage 2: Let $Y_i = 3$ when case 4 occurs (i.e., three steps are needed) and $Y_i = 1$ otherwise. This time, however, $Y_1, Y_2, \ldots, Y_{0.5n}$ may not be totally independent. For example, if two packets heading for block 1 come from the top and third positions of some block (on the upper-right or the lower-left subplane) in Stage 1–1, then there is no possibility that two packets heading for block 2 come from the second and fourth positions of the same block. Namely, if $Y_1 = 3$ (with probability $\frac{1}{4}$), then Y_2 must be 1. However, we can show that the sum of those non-independent random variables does not deviate from its expected value with high probability using the following lemma [15].

Lemma 2. *Let $X_0, X_1, X_2, \ldots$ be a martingale sequence such that for each k,*

$$|X_k - X_{k-1}| \leqslant c,$$

where c is some constant independent of k. Then, for all $t \geqslant 0$ and any $\lambda > 0$,

$$\Pr[|X_t - X_0| \geqslant \lambda c\sqrt{t}] \leqslant 2\exp(-\lambda^2/2).$$

Let $Y = \sum_{i=1}^{0.5n} Y_i$ and define a martingale sequence $X_0, X_1, \ldots, X_{0.5n}$ by setting $X_0 = E[Y]$ and for $1 \leqslant i \leqslant 0.5n$ $X_i = E[Y | Y_1, Y_2, \ldots, Y_i]$. Now we shall evaluate the value $|X_i - X_{i-1}|$ for $1 \leqslant i \leqslant 0.5n$. Note that the difference between X_i and X_{i-1} depends only on the value of Y_i, which is determined by the behavior of two packets heading for the ith block. The behavior of each of those two packets affects the behavior of at most three other packets in its block of Stage 1 and hence at most three other random variables Y_j's. Therefore, conditioning the value of Y_i affects at most seven Y_j's (including Y_i itself) and the difference between X_i and X_{i-1} is bounded by some constant $c_3 \leqslant 3 \times 7 = 21$ since $Y_i \leqslant 3$. By applying Lemma 2 with $t = n/2$ and $\lambda = \sqrt{2d_3 \ln n}$ for some $d_3 > 0$, one can conclude that the number of steps for Stage 2 is at most $0.75n + c_3\sqrt{d_3 n \ln n}$ with probability $1 - 2n^{-d_3}$.

We have $2n$ buses. So, the probability that the bad behavior occurs in at least one bus can be as large as $2n$ times. However, since its probability can be written as n^{-d} for a large enough constant d, we do not have to worry about that. As a result, the whole algorithm takes at most $1.4375n + o(n)$ with high probability. □

Remark 1. We can also design Stage 2 without using an *SP*: *Phase* 1: If P holds both Q_1 and Q_2, then P writes Q_1 and then writes Q_2 at the next step. *Phase* 2: 1–3 are the same [4]. If collision occurred, then we examine which case happened in the previous block. If the case 1 happens, then the collision might be caused by the packet Q_2 of the previous block. So, we insert an extra step to send this Q_2 (but it may be idle if the collision is caused by two packets of the current block). Although details are omitted, the algorithm runs in at most $1.46875n + o(n)$ steps with high probability, which is a little worse than Theorem 1 but is still better than $1.5n$.

Remark 2. Local computation in each step is very simple in *RR*. That is obviously a good point of this algorithm compared to the next algorithm.

4. $1.25n + o(n)$ randomized algorithm

Recall that there are n processors on each bus and roughly one half of these n processors put their packets on its bus in the previous algorithm *RR*. However, it takes much more time than $0.5n$ to finish the transmission. An obvious reason is a lot of packet collisions: If each processor P would know whether or not each other processor on the same bus is now trying to write its packet, then P could calculate a proper time-slot at which P should write its own packet without collision or waste of the bus.

This is in fact possible if P would know all the random numbers the other processors have generated. To this goal, one can use the technique of generating many pseudo-random numbers deterministically from a few random numbers. Some preliminaries are needed: Let $X_1, X_2, \ldots, X_n$ be discrete random variables defined on the same probability space. Such a set of random variables is said to be *pairwise independent* if for all $i \neq j$,

$$\Pr[X_i = x \,|\, X_j = y] = \Pr[X_i = x].$$

This pairwise independence is naturally extended to the *k-wise independence*: A set of similarly defined random variables $X_1,\ldots,X_n$ are said to be k-wise *independent* if for all different $i_1,i_2,\ldots,i_k$,

$$\Pr[X_{i_1}=x_{i_1}\mid X_{i_2}=x_{i_2},X_{i_3}=x_{i_3},\ldots,X_{i_k}=x_{i_k}]=\Pr[X_{i_1}=x_{i_1}].$$

Lemma 3 (see Joffe [9]). *Let m be a prime number and Z_m denote the field of integers modulo m. Then the set of $n<m$ random variables $X_1,\ldots,X_n$ calculated by the following equation from the k numbers, $a_1,\ldots,a_k$, which are randomly chosen from Z_m are k-wise independent*:

$$X_i=a_1i^{k-1}+a_2i^{k-2}+\cdots+a_{k-1}i+a_k \bmod m. \tag{1}$$

Consider for example the n processors $P_1,\ldots,P_n$ on the first row. The leftmost processor P_1 generates k (truly) random numbers $a_1,\ldots,a_k$ and transmits them to P_2 through P_n. Then each P_i can generate its own random number X_i by Eq. (1). The set of X_i's are guaranteed to be k-wise independent. The degree of randomness for each X_i is smaller than before, but we can show that it is enough for our purpose when $k=6$.

We have another technical problem; how to transmit $a_1,\ldots,a_k$. Recall that our rule is that only packets can be transmitted on the bus. Fortunately, the amount of information carried by $a_1,\ldots,a_k$ is not too large since k is a constant. Moreover, we can set m to be nearly equal to n^2, so the number of the total bits of $a_1,\ldots,a_k$ is $O(\log n)$. Consequently, the following simple algorithm works: (i) P_1 creates $a_1,\ldots,a_k$. (ii) Suppose that the bit sequence of $a_1,\ldots,a_k$ (may be encoded) is $b_1,b_2,b_3,\ldots$. Then P_1 puts its original packet repeatedly on the bus at time-slot i if $b_i=1$ and puts nothing if $b_i=0$. This takes only $O(\log n)$ steps.

Now we are ready to give our new algorithm RR_k which consists of two stages as before. In the first stage, about one half of the whole packets move horizontally to their column destinations, and the rest to their row destinations. The second stage is exactly the same as the previous algorithm RR.

Algorithm: RR_k

Stage 1: Choose any prime number $m>n^2$. Let $Z_m=\{1,2,\ldots,m-1\}$, $Z_m^0=\{1,\ldots,m/2\}$ and $Z_m^1=\{m/2+1,\ldots,m-1\}$. Note that $|Z_m|=m-1$ and $|Z_m^0|=|Z_m^1|=(m-1)/2$ ($m-1$ is even since m is a prime number).

Phase 1: $P_{1,1}$ generates k prime numbers $a_1,\ldots,a_k\in Z_m$ (k will be set to six when the probability of success is calculated).

Phase 2: $P_{1,1}$ transmits $a_1,\ldots,a_k$ to all the processors on the first column in the way described above.

Phase 3: $P_{i,1}$ $(1\leqslant i\leqslant n)$ transmits $a_1,\ldots,a_k$ to all the processors on the ith row in the same way.

Phase 4: Now each processor $P_{i,j}$ has $a_1,\ldots,a_k$, from which it computes $f(i,\ j)=a_1\{n(i-1)+j\}^{k-1}+a_2\{n(i-1)+j\}^{k-2}+\cdots+a_k \bmod m$. Then set $X_{i,j}=1$ if $f(i,\ j)\in Z_m^1$ and $X_{i,j}=0$ if $f(i,\ j)\in Z_m^0$.

Phase 5: If $X_{i,j}=1$, then $P_{i,j}$ puts its packet on the row bus first in the following way: $P_{i,j}$ computes how many processors among $P_{i,1},\ldots,P_{i,j-1}$ also write to the row bus first by simulating Phase 4 of each processor (we need local computation proportional to n here). If that number is t, then $P_{i,j}$ writes to the row bus at step $t+1$. If $X_{i,j}=0$, then $P_{i,j}$ puts its packet on the column bus first. It uses the similar calculation to decide when it should do so.

Stage 2: Note that all the processors can calculate when the first stage ends (by calculating the last processor which accesses to the row or column bus at Stage 1). After the first stage is finished, all the processors enter Stage 2 that is exactly the same as Stage 2 of *RR*. One might think why we cannot use the same technique as Stage 1. Recall that there are approximately $0.5n$ packets on each bus at the beginning of this stage. If each processor knows the current positions or the destinations of all those packets on the bus, then we can use the same technique as before. Unfortunately, neither is known.

Theorem 2. *For* $k=6$, RR_k *halts within* $1.25n+o(n)$ *steps with high probability.*

Proof. Note that Phases 1 and 4 include only local computation. Also, as mentioned before, Phases 2 and 3 take only $O(\log n)$ steps. Therefore, what we have to prove is that the number of packets written to a row bus first (and to a column bus first also) is sufficiently close to $0.5n$ and furthermore, at the beginning of Stage 2, the number of packets to move on each single bus is also close to $0.5n$.

Fix some row, say row j. Let P_i be the ith processor on this row and X_i be the random variable such that $X_i=1$ or 0 is determined at Phase 4. Let $X=\sum_{i=1}^{n} X_i$. Then it turns out [15] that the expected value of X is $0.5n$. What we wish to know is the probability that X differs from $0.5n$ by a certain amount of value. Note that X is a random variable that is a sum of (not necessarily independent) random variables X_i, for which we cannot use Chernoff bound. Instead we use the (generalized) Chebyshev bound: For an integer $k\geqslant 2$, let $\mu_X^k=E[(X-0.5n)^k]$. (This is called *kth central moment*, which does not exist for some probability space. It obviously exists in the present case.)

Lemma 4 (see Motwani and Raghavan [15] for example). *For any* $t>0$,

$$\Pr[|X-0.5n|>t\sqrt[k]{\mu_X^k}]\leqslant\frac{1}{t^k}.$$

In order to prove this theorem, it is enough to consider only the case where $k=3$ (the reason will be described later). Let us evaluate the third central moment $\mu_U^3=\mathrm{E}[(U-\mathrm{E}[U])^3]$, where $U=\sum U_i$ is the sum of n 3-wise independent binary random variables. Expand $\mu_U^3=E[(\sum U_i-\sum E[U_i])^3]$, and consider each term. Such a term involves up to three variables from U_i's. However, we can claim that terms involving more than one variable cancel each other. To see this, let T be a term of the expansion that involves more than one variable. Then, T contains a variable U_i that appears in T exactly once. Thus, T can be written in the form $E[T_1U_i]$ or $E[T_2E[U_i]]$, where T_1 or T_2 does

not contain U_i. Note that the terms in these two forms (for fixed U_i) are in one to one correspondence, each $E[T_1U_i]$ corresponding to $E[-T_1E[U_i]]$. Due to the 3-wise independence we can write $E[T_1U_i]$ as $E[T_1]E[U_i]$ and $E[-T_1E[U_i]]$ as $-E[T_1]E[U_i]$ and thus cancel them out. The only remaining terms are of the form $E[U_i^3]$, $E[U_i^2]E[U_i]$, or $E[U_i]^3$, involving only one variable, with some constant coefficients. The contribution of these terms is a constant per variable and thus $O(n)$ in total.

Note that the 6-wise independent random variables X_i's satisfy 3-wise independency. Using Lemma 4 with $\mu_X^3 = O(n)$,

$$\Pr[|X - 0.5n| > t\sqrt[3]{O(n)}] \leqslant \frac{1}{t^3}.$$

If we set $t = n^{c-1/3}$ for a positive constant c, then

$$\Pr[|X - 0.5n| > O(n^c)] \leqslant \frac{1}{n^{3c-1}}.$$

This means $X \leqslant 0.5n + o(n)$ with probability at least $1 - n^{-(3c-1)}$. Then it follows that the number of steps needed in Stage 1 for the fixed single bus is at most $0.5n + o(n) + O(\log n)$ with at least the same probability. This holds for all other buses.

For the analysis of Stage 2, we can apply the same argument as above since random variables Y_i's are 3-wise independent. By the above calculation we can get $\mu_Y^3 = O(n)$, i.e.,

$$\Pr[|Y - 0.75n| > O(n^c)] \leqslant \frac{1}{n^{3c-1}},$$

which says Stage 2 takes at most $0.75n + o(n)$ steps with probability at least $1 - n^{-(3c-1)}$ per bus.

Since there are $2n$ buses, the unsuccessful probability can be up to $2n$ times. However, we can still get a sufficiently small probability by setting $\frac{2}{3} < c < 1$. As a result, RR_k requires $1.25n + o(n)$ steps with high probability. □

5. Concluding remarks

It is known that routing can be done in $1.0n$ steps if all the processors know the source and destination addresses of all the packets (so-called off-line routing) [8]. This $1.0n$ is also an absolute lower bound. The question is how close we can be to this bound in the normal routing. Further improvements may be possible for Stage 2 of both RR and RR_k and for the local computation in Stage 1 of RR_k. Also an interesting question is whether we can apply our randomization technique to improve the upper bound for the mesh equipped with both buses and local links.

References

[1] H. Chernoff, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations, Ann. Math. Statis. 23 (1952) 493–507.

[2] S. Cheung, F.C.M. Lau, A lower bound for permutation routing on two-dimensional bused meshes, Informat. Process. Lett. 45 (1993) 225–228.
[3] D.D. Chinn, T. Leighton, M. Tompa, Minimal adaptive routing on the mesh with bounded queue size, J. Parallel Distrib. Comput. 34 (1996) 154–170.
[4] K. Iwama, Y. Kambayashi, E. Miyano, New bounds for oblivious mesh routing, Proc. 6th European Symp. on Algorithms, 1998, pp. 295–306.
[5] K. Iwama, E. Miyano, Oblivious routing algorithms on the mesh of buses, Proc. Internat. Parallel Processing Symp., 1997, pp. 721–727.
[6] K. Iwama, E. Miyano, Three-dimensional meshes are less powerful than two-dimensional ones in oblivious routing, Proc. European Symp. on Algorithms, 1997, pp. 154–170.
[7] K. Iwama, E. Miyano, An $O(\sqrt{N})$ oblivious routing algorithms for 2-D meshes of constant queue-size, Proc. 10th ACM-SIAM Symp. on Discrete Algorithms, 1999, pp. 466–475.
[8] K. Iwama, E. Miyano, Y. Kambayashi, Routing problems on the mesh of buses, J. Algorithms 20 (1996) 613–631.
[9] A. Joffe, On a set of almost deterministic k-independent random variables, Ann. Probab. 2 (1974) 161–162.
[10] M. Kaufmann, R. Raman, J.F. Sibeyn, Routing on meshes with buses, Algorithmica 18 (1997) 417–444.
[11] D. Krizanc, Oblivious routing with limited buffer capacity, J. Comput. System Sci. 43 (1991) 317–327.
[12] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann, Los Altos, 1992.
[13] L.Y.T. Leung, S.M. Shende, On multidimensional packet routing for meshes with buses, J. Parallel Distrib. Comput. 20 (1994) 187–197.
[14] R. Miller, Q.F. Stout, Parallel Algorithms for Regular Architectures: Meshes and Pyramids, MIT Press, Cambridge, 1996.
[15] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, Cambridge, 1995.
[16] M. Tompa, Lecture Notes on Message Routing in Parallel Machines, Technical Report No: 94-06-05, Department of Computer Science & Engineering, University of Washington, 1994.