Better Deterministic Routing on Meshes

Jop F. Sibeyn*

December 23, 1998

Abstract

Optimal randomized and deterministic algorithms have been given for k-k routing on two-dimensional $n \times n$ meshes. The deterministic algorithm is based on "column-sort" and exploits only part of the features of the mesh. For small n and moderate k, the lower-order terms of this algorithm make it considerably more expensive than the randomized algorithm. In this paper, we present a novel deterministic algorithm, which, by exploiting the topology of the mesh, has lower-order terms that are almost negligible, even smaller than those of the randomized algorithm. An additional advantage of the new algorithm is that it routes average-case packet distributions twice as fast as worst-case distributions. In earlier algorithms this required additional steps for globally probing the distribution.

1 Introduction

Problem. Communication on interconnection networks is a fundamental problem that has to be solved optimally in order to optimize the usage of a parallel computer. One of the basic communication problems is k-k routing, in which every processing unit, *PU*, sends and receives exactly k packets. If the distribution of the destinations is homogeneous, then generally it is quite easy to perform the routing efficiently, but if the source/destination pattern is irregular, than trivial strategies may lead to congestion during the routing, resulting in packets getting delayed and overflowing buffers.

Applying the idea of Valiant and Brebner [8], arbitrary source/destination patterns can be turned into more regular patterns: first all packets are routed to a randomly chosen intermediate destination; then they are routed to their real destinations. This approach has several disadvantages, but is still close to the best possible for routing arbitrary k-k distributions. A central question is whether the same performance can be achieved deterministically. In general the answer to this question is probably negative (for hypercubes the question is still open), but for some networks there are alternative, deterministic, algorithms, which perform at least as good. Some of these algorithms are independent, others are more or less deterministic translations of the same idea.

Closest to a deterministic equivalent of the idea from [8], comes the so-called *column-sort* algorithm by Leighton [1]. For a routing problem involving K packets in total, it proceeds as follows: (1) all packets are divided into subsets of size $K^{2/3}$; (2) for every subset, the packets with destinations in the same subset are redistributed evenly over the subsets; (3) the packets are routed to the subsets of their destinations. The even redistribution of the packets, *unshuffling*, is an analogue of sending packets to random destinations.

Machine. Of the many possible interconnection topologies, grids are among the most considered. A two-dimensional $n \times n$ *mesh* consists of n^2 PUs arranged as a square grid. Each PU is only connected with its at most four neighbors. In this paper we assume the store-and-forward communication model, in which in each step each PU can send one packet to each of its neighbors.

Routing on Meshes. Optimal algorithms for k-k routing and sorting are presented in [2, 3, 5, 7]. The implementation of Valiant and Brebner's algorithm [8] which is given in [3], consists of four phases:

Algorithm RAND_4_PHASE_ROUTE

1. Route all packets to a random position in their rows.

2. Route all packets to a random position in their columns.

3. Route all packets along the rows to their destination columns.

4. Route all packets along the columns to their destination rows.

In [3] it is shown that each phase can be performed in $k \cdot n/4 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ steps, with high probability. If initially the packets are colored white or black independently and uniformly with probability 1/2, then the routing time can be reduced by a factor two by routing the white packets as described above and the black packets orthogonally to them. Thus it follows that

Lemma 1 [3] RAND_4_PHASE_ROUTE *with coloring routes k-k distributions on an $n \times n$ mesh in $k \cdot n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ steps, with high probability.*

RAND_4_PHASE_ROUTE can easily be generalized for higher dimensional meshes and tori (meshes with additional wrap-around connections). For two-dimensional meshes, however, there is an alternative version consisting of only three phases. The time consumption of

*Max-Planck-Institut für Informatik, Im Stadtwald, Saarbrücken, Germany. Email: jopsi@mpi-sb.mpg.de. Url: http://www.mpi-sb.mpg.de/~jopsi/.

RAND_3_PHASE_ROUTE does not change if the order of Phase 3 and Phase 4 are exchanged. But then, there are two consecutive phases of routing along the columns, which can be replaced by one:

Algorithm RAND_3_PHASE_ROUTE

1. Route all packets to a random position in their rows.

2. Route all packets along the columns to their destination rows.

3. Route all packets along the rows to their destination columns.

Lemma 2 [2] RAND_3_PHASE_ROUTE *with coloring routes k-k distributions on an $n \times n$ mesh in $k \cdot n/2 + \mathcal{O}((k \cdot n \cdot \log n)^{1/2})$ steps, with high probability.*

Optimal deterministic algorithms (for sorting) have been given in [7, 5]. They can be viewed as implementations of column-sort [1], that is, as deterministic versions of RAND_4_PHASE_ROUTE: all packets are routed twice across the whole mesh. We call this algorithm DET_4_PHASE_ROUTE:

Lemma 3 DET_4_PHASE_ROUTE *with coloring deterministically routes k-k distributions on an $n \times n$ mesh in $k \cdot n/2 + \mathcal{O}(n^{2/3}/k^{1/6})$ steps.*

Because of the rearrangements that have to be made in submeshes of size $n^{2/3}/k^{1/6} \times n^{2/3}/k^{1/6}$, DET_4_PHASE_ROUTE is not particular efficient. Only for very large k, $k = \omega(n^4)$, all rearrangements become internal operations. In that case the algorithm will be better than the randomized algorithms because of its perfectly regular communication pattern.

We strive for a more efficient deterministic algorithm, outperforming the randomized algorithms already for much smaller values of k. In Section 3, we present a deterministic version of RAND_3_PHASE_ROUTE. Already for $k = \omega(n^2 \cdot \log n)$ all routing decisions can be made internally. In Section 4 we show that without modification this algorithm has a large degree of adaptivity: average-case distributions are routed optimally in $(1 + o(1)) \cdot k \cdot n/4$ steps. Finally we show how this algorithm can be extended for small k.

2 Preliminaries

Indexing. The $n \times n$ mesh has n rows and n columns, indexed from 0 to $n - 1$, counting from the lower-left corner. The PU at position (i, j), is the PU in row i and column j. Occasionally we refer to PU p, for $0 \leq p < n^2$. Here p is the index of the PU with respect to the *row-major indexing*: under this indexing scheme, the PU at position (i, j) has index $i \cdot n + j$. Under the *column-major indexing*, the PU at position (i, j) has index $j \cdot n + i$.

Cost measures and Goals. The principle cost measure of algorithms on a store-and-forward network is the number of routing steps. Furthermore, the maximum number of packets stored at the same time in a PU must be bounded. Typically, one would like that this *queue size* is at most $(1 + o(1)) \cdot k$.

Our algorithm is intended to give improvements for the case of rather small n and rather large k. In practice we currently find parallel computers with at most a few thousands of PUs. In the typical coarse-grain applications, k is mostly considerably larger than n, but not infinite. A condition such as $k = \omega(n^4)$, as needed for effcient application of DET_4_PHASE_ROUTE may be a serious obstacle.

Regularly Extracting Subsets . Suppose that we have to extract x elements out of a sorted set of elements of size s. Suppose that the elements are indexed from 0 to $s - 1$. Then we can do this in a *regular* way by taking the x elements with indices $\lfloor i \cdot s/x \rfloor$, for all $0 \leq i < x$. This takes $\mathcal{O}(x)$ time.

In Section 3, we will need to decompose a sorted set regularly in subsets whose sizes are powers of two. In that case the number $m = \lfloor \log_2 s \rfloor$ is determined and $x = 2^m$ elements are regularly extracted from the s elements. Then this procedure is repeated with the remaining set of $s - x$ elements. In total this takes $\mathcal{O}(s)$ time. The number of created subsets is at most m.

Routing in Rows and Columns. All algorithms considered in this paper consist of phases in which the routing is performed only in rows or columns (possibly one routing in the rows is overlaid with an independent routing in the columns). The time for such an operation on one-dimensional submeshes has been analyzed in detail in [2].

Consider a linear processor array with n PUs, indexed from left to right from 0 to $n - 1$. If, while routing on this network, several packets want to use the same connection at the same time, then the packet that still has to travel farthest gets priority. This is called the *farthest-first strategy*, it is tacitly applied in all presented routing algorithms.

For a packet distribution $\mathcal{D}$, let $h_r(\mathcal{D}, i, j)$ be the number of packets that have to move from the set of PUs with index smaller than i to the set of PUs with index larger or equal than j, and let $T_r(\mathcal{D})$ denote the number of rightwards steps that must be performed for routing $\mathcal{D}$.

Lemma 4 [2] *Given an arbitrary packet distribution $\mathcal{D}$ on a linear array with n PUs, $T_r(\mathcal{D}) = \max_{i<j}\{j - i - 1 + h_r(i, j)\}$.*

There is an analogous lemma for the leftward routing. If there are only few packets, it may make sense to choose $i < j - 1$, but for the packet densities we will consider, Lemma 4 can be simplified to a form that will be used in the remainder of this paper:

> The time for routing on a one-dimensional array equals the maximum number of packets that have to travel over any connection.

In this paper all logarithms are taken to the base two.

3 Better Deterministic Routing

Meshes have very special features, which make that techniques can be applied that would be far too expensive on more powerful networks. The most important feature is that many basic operations performed in $n' \times n'$ submeshes, with $n' = o(n)$ have, at least in theory, negligible cost. DET_4_PHASE_ROUTE exploits this feature of the mesh. Less frequently exploited is the topology of the mesh, the fact that it is composed of rows and columns. In this section we present a new algorithm for k-k routing on $n \times n$ meshes, exploiting the topology. Here we assume that $k = \omega(n^2 \cdot \log n)$. In Section 5 the algorithm is extended to general k by adding operations in submeshes.

Algorithm. The algorithm, DET_3_PHASE_ROUTE, is almost the same as RAND_3_PHASE_ROUTE, the only difference being that in Step 1, the intermediate position is not determined by randomization. It remains to settle how this intermediate position can be determined so that all of the following goals are achieved:

- The routing in Phase 1 and Phase 3 takes at most $(1 + o(1)) \cdot k \cdot n/4$ steps.
- The routing in Phase 2 takes at most $(1 + o(1)) \cdot k \cdot n/2$ steps.
- The queue size at the end of Phase 1 and Phase 2 is limited to $(1 + o(1)) \cdot k$.

Each PU performs the following steps to determine the intermediate positions of its packets:

Algorithm ALLOCATE

1. Create n buckets, indexed from 0 to $n-1$. Sort the k packets into the buckets corresponding to their destination rows.

2. Sort the packets in each bucket on their destination columns.

3. Out off each bucket with size $B \geq n$, regularly extract $n \cdot \lfloor B/n \rfloor$ packets. For i running from 0 to n, allocate the $\lfloor B/n \rfloor$ packets with indices from $i \cdot \lfloor B/n \rfloor$ up to $(i+1) \cdot \lfloor B/n \rfloor - 1$ to position i.

4. Let α_j denote the number of packets that this PU allocates to position j of its row. Initialize all α_j on 0.

5. Regularly divide all buckets into subbuckets whose sizes are powers of two by the procedure described in Section 2. Let m denote the number of subbuckets, and let β_l, $0 \leq l < m$, denote the size of subbucket l.

6. Sort the subbuckets on their sizes in decreasing order. Hereafter, $\beta_l \geq \beta_{l'}$, for all $l < l'$.

7. Sort all subsets of subbuckets with the same sizes on the destination rows of their packets.

8. For l running from 0 to $m-1$, allocate the packets in subbucket l, as follows: divide the indices from 0 to $n-1$ in β_l approximately equal intervals. For every i, $0 \leq i < \beta_l$, determine the index j in interval i, for which α_j is minimal. Allocate packet i from subbucket l to position j and set $\alpha_j = \alpha_j + 1$.

Step 5 serves two purposes. As we will see, it allows to perform ALLOCATE in $\mathcal{O}(n)$ time. This appears hard to achieve if all possible bucket sizes may occur. Furthermore, Step 5 makes it relatively easy to give tight estimates of the rounding errors. The disadvantage of Step 5 is an increased number of buckets, resulting in more rounding errors. The function of the reordering in Step 2 and Step 6 and the sorting in Step 7, will become clear from the analysis.

Analysis. In order to simplify the analysis, we assume that n is a power of two and that k is a multiple of n. First we determine a bound on the number m of subbuckets we are dealing with:

Lemma 5 $m \leq n \cdot \lfloor \log n \rfloor$.

Proof: After Step 3, the buckets holds at most n packets. Each of these n buckets is split up into at most $\lfloor \log(n-1) \rfloor$ subbuckets. □

Now we show that ALLOCATE leads to a perfectly even allocation of the packets. The proof essentially relies on the following facts which are assured because of Step 5 and Step 6:

- $\beta_l \geq \beta_{l-1}$ for all l, $1 \leq l < m$;
- If $\beta_l > \beta_{l-1}$, then β_l is a multiple of β_{l-1}, for all l, $1 \leq l < m$.

Lemma 6 *Performing* ALLOCATE*, each PU allocates exactly k/n packets to each PU in its row.*

Proof: Let $\alpha_j(l)$ be the value of α_j after allocating the packets in subbucket i, and define $\Delta(l) = \max_{j,j'} \{\alpha_j(l) - \alpha_{j'}(l)\}$. We will show that $\Delta(l) \leq 1$, for all l. For $l = m-1$, this implies $\Delta(l) = 0$, because, when k is a multiple of n, k balls cannot be distributed over n holes with a maximum difference of exactly one.

We apply induction on l. Initially, we have $\Delta(-1) = 0$. As a second invariant we need that the number of packets allocated to each of the current β_l intervals is exactly the same. By intervals we mean the intervals of size n/β_i in which the n indices are divided in Step 8. Initially this is true. Now assume that both facts hold after allocating the packets in subbucket $l-1$. For the allocation of subbucket l, there are two cases to distinguish. First consider the case $\beta_l = \beta_{l-1}$. In this case, the size of the intervals remains the same, and in each of the β_l intervals, one packet is allocated to a PU j which has minimal α_j. Afterwards again all intervals have the same number of packets allocated to them. Inside each interval, the differences between the α_j is at most one because this was true before and because the new packet was allocated to a PU j with minimal α_j; between the intervals this is guaranteed because they have the same size and the

same number of packets has been allocated to them. On the other hand, if $\beta_l < \beta_{l-1}$, then each of the β_l new intervals consists of β_{l-1}/β_l old intervals. Thus, the property that all intervals have the same number of packets allocated to them carries over. □

Corollary 1 *The routing in Phase 1 takes exactly $k \cdot n/4$ steps. At the end of Phase 1 each PU holds exactly k packets.*

Proof: The connection in the middle is the heaviest loaded. Rightwards it has to transfer $k/2$ packets for each of the $n/2$ PUs to its left, and leftwards $k/2$ packets for each of the $n/2$ PUs to its right. Each PU receives k/n packets from each of the n PUs in its row. □

Now we analyze the routing in Phase 2.

Lemma 7 *At the end of Phase 2 a PU holds less than $k + n^2 \cdot \log n$ packets.*

Proof: Let $\gamma_{p,i}$ be the number of packets in PU p with destination in row i. Because every PU is the destination of k packets, $sum_p \gamma_{p,i} = k \cdot n$. In Step 3, PU p allocates $\lfloor \gamma_{p,i}/n \rfloor$ of these packets to the PU in a given column j of its row. So, in total $\sum_p \lfloor \gamma_{p,i}/n \rfloor \leq \sum_p \gamma_{p,i}/n = k$ of these packets are allocated to PUs in column j. Now consider Step 8. There are less than $n^2 \cdot \log n$ subbuckets whose packets have destination in row i. Of each subbucket, at most one packet is allocated to a PU in column j. Thus, the number of packets that may gather in the PU at position (i, j) is bounded by $k + n^2 \cdot \log n$. □

The estimate of Lemma 7 is rather tight. It makes that we must take $k = \omega(n^2 \cdot \log n)$, to achieve the queue size of $(1+o(1)) \cdot k$ we are striving for. The factor $\log n$ is due to the division of buckets into subbuckets in Step 5 of ALLOCATE, the factor n^2 is an inevitable consequence of the fact that the PUs allocate their packets in a non-coordinated way.

In a similar way, we determine the maximum number of packets that may have to be transferred over a connection. Here we need the reordering of Step 7, which assures that packets with destinations in adjacent rows are treated in a more or less coordinated way.

Lemma 8 *The number of steps for the routing in Phase 2 is bounded by*

$$k \cdot n/2 + n^2 \cdot \log n/2.$$

Proof: Consider the packets with destination in the highest i rows of the mesh. We will put a bound on the number $h(i)$ of these packets that travel upwards over the connection between the PU at position $(i-1, j)$ and the PU at position (i, j) within some column j.

The allocation of a batch of at most n subbuckets with common size is called an *allocation round*. Because after Step 3 a PU holds at most n packets with destinations in the same row, there are at most $\lfloor \log n \rfloor$ allocation rounds. Because of Step 7, in every allocation round, the subbuckets with packets going to the highest i rows stand together. Thus, their allocation is performed as if we had one large subbucket with size equal to the sum of their sizes. Let $\Gamma_{p,l}$, for $0 \leq p < n^2$ and $0 \leq l < \log n$, denote this sum for PU p in allocation round l. This gives:

$$h(i) \leq \sum_{p=0}^{(n-i)\cdot n} \sum_{l=-1}^{\lfloor \log n \rfloor} \lceil \Gamma_{p,l}/n \rceil.$$

Here we assumed a row-major indexing, which means that the PU p with $0 \leq p < (n-i) \cdot n$ are the PUs in the lowest $n - i$ rows. By $\Gamma_{p,-1}$ we denoted the packets that have been allocated in Step 3. Because every PU is the source and destination of at most k packets, $\sum_{p=0}^{(n-i)\cdot n} \sum_{l=-1}^{\lfloor \log k \rfloor} \Gamma_{p,l} \leq \min\{k \cdot (n-i) \cdot n, k \cdot i \cdot n\}$. Thus,

$$h(i) < \min\{k \cdot (n-i), k \cdot i\} + (n-i) \cdot n \cdot \log n.$$

The maximum is assumed for $i = n/2$. □

It only remains to analyze the routing in Phase 3. So far, we have not seen why the packets had to be sorted on their destination columns in Step 2 of ALLOCATE. This step was added to guarantee that the routing in Phase 3 is at worst as hard as routing a uniform distribution (in Section 4 we see that in most cases it is much easier). The proof of the following lemma is based on the fact that the allocation is performed so that at least half of the packets with destination in the right half of row i already stand in the right half of row i at the beginning of Phase 3.

Lemma 9 *The number of steps for the routing in Phase 3 is bounded by*

$$k \cdot n/4 + n^2 \cdot \log n.$$

Proof: Consider the packets with destinations in the rightmost j PUs of some row i. We will put a bound on the number $h(j)$ of these packets that travel rightwards over the connection between the PU at position $(i, j-1)$ and the PU at position (i, j).

$h(j)$ is maximized, if all $k \cdot j$ packets stand in PUs that do not hold packets with destinations in the left part of row i as well. In that case, a subbucket l of size β_l is distributed so that at most $\lceil \beta_l \cdot (n-j)/n \rceil$ packets are allocated to PUs in the leftmost $n - j$ columns. Summing over all (less than) $n^2 \cdot \log n$ relevant subbuckets gives

$$h(j) \leq \sum_l \lceil \beta_l \cdot (n-j)/n \rceil < k \cdot j \cdot (n-j)/n + n^2 \cdot \log n.$$

The maximum is assumed for $j = n/2$. □

Linear Work. All sorting operations in ALLOCATE can be performed in $\mathcal{O}(k \cdot \log k)$ (parallel) time. However, if $k \geq n^2$,

all sorts can be implemented as bucket-sorts and performed in $\mathcal{O}(k)$ time.

More interesting is the organization of Step 8. During this step, each PU should maintain for every interval a list of indices j for which the α_j are smaller than the maximum value. If a list gets exhausted, a new list containing all indices of the interval is created. Clearly these operations take constant (amortized) time per allocated packet. If the size of the intervals becomes larger, then the lists of the intervals that are unified into one new interval are unified as well. This takes $\mathcal{O}(n)$ additional time in total.

Summarizing all results of this section,

Theorem 1 *On an $n \times n$ mesh Applying* ALLOCATE *and coloring,* DET_3_PHASE_ROUTE *routes k-k distributions in $k \cdot n/2 + \mathcal{O}(n^2 \cdot \log n)$ steps. The maximum queue size is bounded by $k + n^2 \cdot \log n$. If $k \geq n^2$, the amount of internal work is linear in k.*

Proof: For the coloring, each PU first determines all packets with common destination. Then, half of these are colored white and the other half black, taking care that in total $k/2$ packets get each color. Hereafter the white packets are treated as described above, while the black packets are routed orthogonally. □

Thus, for $k = \omega(n^2 \cdot \log n)$, the queues are small, the routing time is close to optimal and the internal work is linear: all goals are achieved.

4 Average-Case Distributions

A k-k routing algorithm guaranteeing a worst-case time of $(1 + o(1)) \cdot k \cdot n/2$, while achieving $(1 + o(1)) \cdot k \cdot n/4$ for average-case distributions, is given in [6]. However, this was achieved by somewhat ad-hoc changes to a standard k-k routing algorithm. In this section we show that DET_3_PHASE_ROUTE routes average-case distributions optimally without any modification: Phase 1 and Phase 2 each cost about $k \cdot n/4$ (coloring reduces this by a factor of two), while Phase 3 is almost for free. Due to a lack of space all proofs had to be omitted in the rest of the paper.

Lemma 10 *For a uniform distribution of the destinations, the number of steps for the routing in Phase 2 is bounded by*

$$k \cdot n/4 + n^2 \cdot \log n/2 + \mathcal{O}((k \cdot n^2 \cdot \log n)^{1/2}),$$

with high probability.

So far, it was of no importance that in Step 5 of ALLOCATE, the subbuckets were extracted from the buckets in a regular way. This regular selection assures that, if a PU holds approximately the same number of packets going to either half of the mesh, this is also true for the subbuckets. But this in turn implies that in Step 8 most packets are allocated to the half of their destination and that in Phase 3 only few of them have to be routed far.

Lemma 11 *For a uniform distribution of the destinations, the number of steps for the routing in Phase 3 is bounded by*

$$\mathcal{O}(n^2 \cdot \log n + (k \cdot n^3 \cdot \log n)^{1/2}),$$

with high probability.

Theorem 2 *On an $n \times n$ mesh, applying* ALLOCATE *and coloring,* DET_3_PHASE_ROUTE *routes uniform k-k distributions in $k \cdot n/4 + \mathcal{O}(n^2 \cdot \log n + (k \cdot n^3 \cdot \log n)^{1/2})$ steps, with high probability. The maximum queue size is bounded by $k + n^2 \cdot \log n$. If $k \geq n^2$, the amount of internal work is linear in k.*

As before, all goals are achieved for $k = \omega(n^2 \cdot \log n)$.

5 Small k

In most practical cases, the condition $k = \omega(n^2 \cdot \log n)$ will not be a serious limitation, but theoretically we would like to have a more general result. In this section, we show an easy extension for arbitrary k.

The idea is that the operations that were previously performed internally before Phase 1 are now performed within $n' \times n'$ submeshes for a suitable n' to be determined hereafter. In addition operations to minimize queue sizes and routing delays are performed at the beginning of Phase 2 and Phase 3. After Phase 3, some local rearrangement will bring the packets that so far only reached their destination submeshes to their destinations. For k-k routing and sorting in the submeshes, we can use any algorithm that achieves this in $\mathcal{O}(k \cdot n')$ steps with small maximum queue sizes, for example the algorithm from [5]. This idea is conventional and has been applied in all mentioned deterministic algorithms, so we will not go into detail here.

For given n', there are n^2/n'^2 submeshes, each holding $k \cdot n'^2$ packets. The submeshes play the role of the PUs. The role of the rows is now played by the n/n' *row-bundles*: bundles of n' rows consisting of n/n' submeshes each. Analogously defined *column-bundles* take over the role of the columns. ALLOCATE is generalized to this new context. In every submesh the following steps are performed:

Algorithm ALLOCATE′

1. Create n/n' buckets, indexed from 0 to $n/n' - 1$. Sort the $k \cdot n'^2$ packets into the buckets corresponding to their destination row-bundles.

2. Sort the packets in each bucket on their destination column-bundles.

3. Out off each bucket with size $B \geq n/n'$, regularly extract $n/n' \cdot \lfloor B \cdot n'/n \rfloor$ packets. For i running from 0 to n/n', allocate the $\lfloor B \cdot n'/n \rfloor$ packets with indices from $i \cdot \lfloor B \cdot n'/n \rfloor$ up to $(i+1) \cdot \lfloor B \cdot n'/n \rfloor - 1$ to position i.

4. Let α_j denote the number of packets that this PU allocates to submesh j of its row-bundle. Initialize all α_j on 0.

5. Regularly divide all buckets into subbuckets whose sizes are powers of two by the procedure described in Section 2. Let m denote the number of subbuckets, and let β_l, $0 \leq l < m$, denote the size of subbucket l.

6. Sort the subbuckets on their sizes in decreasing order. Hereafter, $\beta_l \geq \beta_{l'}$, for all $l < l'$.

7. Sort all subsets of subbuckets with the same sizes on the destination row-bundles of their packets.

8. For l running from 0 to $m-1$, allocate the packets in subbucket l, as follows: divide the indices from 0 to $n/n' - 1$ in β_l approximately equal intervals. For every i, $0 \leq i < \beta_l$, determine the index j in interval i, for which α_j is minimal. Allocate packet i from subbucket l to submesh j and set $\alpha_j = \alpha_j + 1$.

Once it is known how many packets are going to every submesh, Step 3 can be performed mainly by computation. Thus, ALLOCATE takes only $\mathcal{O}(k \cdot n')$ steps. Hereafter the packets in the submeshes are sorted in column-major order on the index of the submesh to which they are allocated, and then Phase 1 is performed. Before Phase 2, the packets in the submeshes are sorted in row-major order on their destination row-bundles, and before Phase 3 in column-major order on their destination column-bundles. Finally they are routed in the submeshes. All these additional operations together take $\mathcal{O}(k \cdot n')$ steps. In order to minimize the queue sizes, the packets are routed at the end of the phases to the first PU in the submesh to which they are traveling that is currently storing less than Q packets, were $Q = (1 + o(1)) \cdot k$ is the maximum number that we are willing to allow.

Lemma 12 *At the end of Phase 2 a PU holds less than* $k + n^2/n'^4 \cdot \log(n/n') + n/n'^2$ *packets.*

Lemma 13 *The number of steps for the routing in Phase 2 is bounded by*

$$k \cdot n/2 + n^2/n'^3 \cdot \log(n/n') + n/n'.$$

As Corollary 1 carries over, and the analogue of Lemma 9 can be proven similarly to Lemma 13, we get:

Theorem 3 *On an $n \times n$ mesh Applying* ALLOCATE′, *coloring and the rearrangements in submeshes,* DET_3_PHASE_ROUTE *routes k-k distributions in $k \cdot n/2 + \mathcal{O}(k \cdot n' + n^2/n'^3 \cdot \log n)$ steps. The maximum queue size is bounded by $k + \mathcal{O}(n^2/n'^4 \cdot \log n)$.*

Corollary 2 *Taking $n' = \omega(\sqrt{n} \cdot (\log n/k)^{1/4})$, results in a queue size bounded by $(1 + o(1)) \cdot k$ and a routing time bounded by $k \cdot n/2 + \mathcal{O}(k^{3/4} \cdot \sqrt{n} \cdot \log^{1/4} n)$.*

Our algorithm has very modest additional terms. Comparing the result of Corollary 2 with that given in Lemma 2, we see that for $k < \log n$ the new algorithm is even better than the randomized algorithm. No earlier deterministic algorithm has achieved this.

6 Conclusion

We have presented a new deterministic algorithm for k-k routing on meshes. It reduces to a very simple form, only consisting of one-dimensional subroutines, for a much smaller number of packets than earlier deterministic algorithms. For packet distributions that happen to be uniform, it is twice as fast as for worst-case distributions. In this respect it even outperforms the randomized algorithm.

The factors $\log n$ that appear in the time consumption and queue size of our algorithm are introduced by Step 5 of ALLOCATE. Possibly the packets with destination in the same row could be allocated all at once, but one has to be careful not to increase the routing time of Phase 3.

The algorithm can also be applied for routing on tori. ALLOCATE is performed in the same manner. For the one-dimensional routing operations in Phase 2 and Phase 3, one should apply the efficient algorithm from [4]. All obtained results carry on with leading terms half as large as for meshes.

References

[1] Leighton, F.T., 'Tight Bounds on the Complexity of Parallel Sorting,' *IEEE Transactions on Computers, C-34(4)*, pp. 344–354, 1985.

[2] Kaufmann, M., S. Rajasekaran, J.F. Sibeyn, 'Matching the Bisection Bound for Routing and Sorting on the Mesh,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 31–40, ACM, 1992.

[3] Kaufmann, M., J.F. Sibeyn, 'Optimal Multi-Packet Routing on the Torus,' *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, LNCS 621, pp. 118–129, Springer-Verlag, 1992.

[4] Kaufmann, M., J.F. Sibeyn, 'Deterministic Routing on One-Dimensional Arrays,' *Proc. 4th Symposium on Parallel and Distributed Processing*, pp. 376–383, IEEE, 1992.

[5] Kaufmann, M., J.F. Sibeyn, T. Suel, 'Derandomizing Algorithms for Routing and Sorting on Meshes,' *Proc. 5th Symposium on Discrete Algorithms*, pp. 669–679, ACM-SIAM, 1994.

[6] Kaufmann, M., J.F. Sibeyn, T. Suel, 'Beyond the Bisection Bound: Fast Ranking and Counting on Meshes,' *Proc. 3rd European Symposium on Algorithms*, LNCS 979, pp. 75–88, Springer-Verlag, 1995.

[7] Kunde, M., 'Block Gossiping on Grids and Tori: Deterministic Sorting and Routing Match the Bisection Bound,' *Proc. European Symposium on Algorithms*, LNCS 726, pp. 272–283, Springer-Verlag, 1993.

[8] Valiant, L.G., G.J. Brebner, 'Universal Schemes for Parallel Communication,' *Proc. 13th Symposium on Theory of Computing*, pp. 263–277, ACM, 1981.