Deterministic 1-k Routing on Meshes

With Applications to Worm-Hole Routing

Jop F. Sibeyn* Michael Kaufmann†

Abstract

In 1-k routing each of the n^2 processing units of an $n \times n$ mesh connected computer initially holds 1 packet which must be routed such that any processor is the destination of at most k packets. This problem reflects practical desire for routing better than the popular routing of permutations. 1-k routing also has implications for hot-potato worm-hole routing, which is of great importance for real world systems.

We present a near-optimal deterministic algorithm running in $\sqrt{k} \cdot n/2 + \mathcal{O}(n)$ steps. We give a second algorithm with slightly worse routing time but working queue size three. Applying this algorithm considerably reduces the routing time of hot-potato worm-hole routing.

Non-trivial extensions are given to the general l-k routing problem and for routing on higher dimensional meshes. Finally we show that k-k routing can be performed in $\mathcal{O}(k \cdot n)$ steps with working queue size four. Hereby the hot-potato worm-hole routing problem can be solved in $\mathcal{O}(k^{3/2} \cdot n)$ steps.

Keywords: theory of parallel and distributed computation, meshes, packet routing, hot-potato worm-hole routing.

1 Introduction

Parallel computation is an area of intensive development during the last decade. Various models for parallel machines have been designed. One of the simplest and therefore best studied machines with a fixed interconnection network, is the MIMD **mesh**. In this model the processing units, **PUs**, form an array of size $n \times n$ and are connected by a two-dimensional grid of communication links. In Section 2.1 the model is described in more detail.

Generally, the problems concerning the exchange of information packets among the PUs are called **routing problems**. Here the destinations of the packets are known beforehand. The task is to send each packet to its destination such that at most one packet passes through any wire during a single step. The quality of a routing algorithm is determined by (1) its **running time**, i.e., the maximum time a packet may need to reach its destination, and (2) its **queue length**, which is defined to be the maximum number of packets any PU may have to store during the routing.

A special case of the routing problem is **permutation routing**. In permutation routing, each PU is the origin of at most one packet and each PU is the destination of at most one packet. Permutation routing has been considered extensively. Optimal randomized and deterministic algorithms were found [18, 13, 2].

When the size of the packets is so large that they cannot be transferred over a connection in a single step, the packets have to be split into several **flits**. The routing of these flits is considered in the k-k **routing problem**: each PU is assumed to send and receive at most k packets. If the flits are routed independently of each other we speak of **multi-packet routing**. Multi-packet routing is also important when the PUs have to route packets to several destinations. Multi-packet routing algorithms [12, 8, 11, 9] solve this task much faster than routing the packets one-by-one. Alternatively, the flits can be routed as a kind of worm such that consecutive flits of a packet reside in adjacent PUs during all steps of the routing: **cut-through routing** [8]. If there is the additional condition that the worms may be expanded and contracted only once, then this variant is called **worm-hole routing** [6, 15]. Unlike the other more theoretical models, worm-hole routing has direct applications in many parallel machines [1, 16, 20, 3].

We consider an original variant of the routing problem: the routing of 1-k **distributions**, under which every PU is sending at most one packet, but may be the destination of up to k packets. 1-k **routing** reflects practical purposes better than the routing of permutations: if the PUs are working independently of each other and generate packets that have to be transferred to other PUs, then it is unrealistical to assume that every PU is the destination of at most one packet. The parameter k, $1 \leq k \leq n^2$, need not to be known by the PUs, but is needed for stating the complexity of the problem. The 1-k routing problem also has implications for hot-potato worm-hole routing. **Hot-potato routing** is a routing paradigm in which packets may

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. E-mail: jopsi@mpi-sb.mpg.de. This research was partially supported by EC Cooperative Action IC-1000 (Project ALTEC: Algorithms for Future Technologies).

†Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, 72076 Tübingen, Germany. E-mail: mk@informatik.uni-tuebingen.de

never be queued at a PU but have to keep moving at all times until they reach their destination [21, 5]. Like worm-hole routing, this model is used in many systems [7, 22]. In a recent paper of Newman and Schuster [15] it is demonstrated that under a light condition any efficient 1-k routing algorithm with working queue size at most four is useful as a subroutine for the hot-potato worm-hole routing problem. As far as we know we are the first to perform a serious analysis of the 1-k routing problem for meshes. For certain expander networks, the related l-k routing problem has been studied in [17].

In our first algorithm (Section 3 and Section 4) a packet is routed either **row-first**: along the row to its destination column and then along this column to its destination; or **column-first**: first along the column and then along the row. The central point is the decision which packets will be routed row-first and which packets will be routed column-first. Straight-forward strategies result in an algorithm requiring $\Omega(k \cdot n)$ steps. A connection-availability argument gives a lower bound of $\sqrt{k} \cdot n/2$. The essential idea which makes possible the break-through to a $\mathcal{O}(\sqrt{k} \cdot n)$ time algorithm, is to count the numbers of packets with destination in every row and column, and to make the decision on basis of these data: if many packets are going to a row, it is better to route most of the packets row-first. Most easily this idea is worked out by tossing a biased coin, but with some extra routing steps we can do it deterministically as well. An intricate algorithm has close to optimal performance, $\sqrt{k} \cdot n/2 + 4 \cdot n$ steps, for $k = o(n^{1/3})$, a much simpler algorithm works well for general k and requires only slightly more steps. The queues do not get longer than $2 \cdot k + 2$ packets. Without modification the basic algorithm can be applied to the general l-k **routing problem**. This is the problem of routing l-k **distributions**, distributions of packets such that every PU sends at most l packets and receives at most k packets. This problem is interesting because of its generality but also because it appears as a subproblem to the routing of 1-k distributions on higher dimensional meshes. We show that the algorithm is near-optimal when $l = o(k)$.

It turns out (Section 5) that under a natural condition l-k distributions can be routed in $\mathcal{O}(l \cdot n)$ time for l upto $n \cdot k$. Informally, the condition is that the 'density of the destinations increases gradually', excluding large areas in which all PUs receive k packets.

In our second algorithm (Section 6) we aim for a working queue size four or less, in order to create a subroutine for the hot-potato worm-hole routing algorithm of [15]. This is achieved with routing time $5 \cdot \sqrt{k} \cdot n + o(\sqrt{k} \cdot n)$. In the algorithm the mesh is subdivided into k squares of size $n/\sqrt{k} \times n/\sqrt{k}$ and the packets are redistributed such that every square holds at most one packet for each destination. Then these squares are rotated along a Hamiltonian cycle and after each $n/\sqrt{k}$ steps the packets that reached their destination square are routed to their destination. Performing the algorithm with a subdivision of the mesh into $k/2$ squares, we can reduce the routing time to $6/\sqrt{2} \cdot \sqrt{k} \cdot n + o(\sqrt{k} \cdot n)$. A theorem of [15] implies that using either of these algorithms as a subroutine, the hot-potato worm-hole routing problem for worms of maximal length k can be solved in $\mathcal{O}(k^2 \cdot n)$ steps, coming close to the obvious lower bound of $\Omega(k \cdot n)$.

As an extension we consider the routing of 1-k distributions on d-dimensional meshes (Section 7). For $d = 3$, the algorithm requires less than twice as much as the lower bound. Also the algorithm for routing 1-k distributions with short working queues is generalized. Compared to the lower bound, the algorithm requires an additional factor of $\mathcal{O}(d)$.

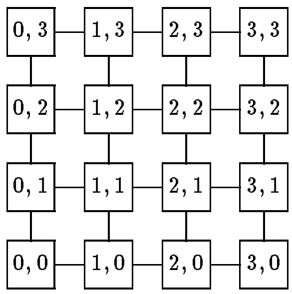
Finally (Section 8) we apply the new techniques to the problem of k-k routing with short queues. With a *randomized* algorithm this is rather easy. With a minor extension of the model, k-k distributions can even be routed with a *deterministic* algorithm in $\mathcal{O}(k \cdot n)$ steps and with working queue size four. This reduces the routing time for the hot-potato worm-hole routing problem to $\mathcal{O}(k^{3/2} \cdot n)$.

2 Preliminaries

Definition 1 *We call an algorithm near-optimal if the leading term of its runtime matches the leading term of a lower bound.*

2.1 Machine Model

As computer model we assume a two-dimensional $n \times n$ MIMD mesh without wrap-around connections. We refer to this machine simply by **mesh**. It consists of n^2 PUs, each of which is connected to (at most) four other PUs by a regular square grid. The PU at position (i, j) is referred to by $P_{i,j}$, and $(0, 0)$ is in the lower-left corner. For $n = 4$ the mesh can be represented by



The PUs are synchronized. In a single **step** each PU can perform arbitrary internal computation and communicate with all its neighbors. The only restriction is that each PU can send and receive at most one packet of bounded length per edge and per step. Thus a PU may send and receive during a step (at most) four packets.

Neglecting the computation time is motivated by the fact that generally computation steps can be performed much faster than routing steps.

Each PU has a **working queue**, in which packets are stored temporarily, and an **internal queue** in which packets are stored that do not yet move or that have reached their destination. Only packets in the working queue can be transferred to a neighboring PU. On the packets in the internal queues any operation can be performed and new packets can generated. The operations that can be performed on the packets in the working queues are limited to checking and comparing their keys. This allows sorting and routing operations. For a 1-k routing problem the internal queues have size k. The size of the working queue Q might become much larger but with good management it can be kept small, $\mathcal{O}(k)$ or even three.

2.2 Indexing Schemes

Next to the popular indexing schemes, row-major column-major and the snake-like variants of these, we use some less common schemes.

We define a **k-layer column-major scheme** (also known as 'layer-first scheme'). Let P be a PU of an $n \times n$ mesh, with index i with respect to the column-major scheme. Then in a k-layer column-major scheme P has k indices: $i + j \cdot n^2$, $0 \leq j \leq k-1$. That is, P gets the indices of the PUs that would stand over it in an $n \times n \times k$ mesh. This indexing scheme is useful for specifying the destinations of packets when performing k-k sorting. We illustrate it with an example. In a 4-layer scheme on a 3×3 mesh, the PUs have the following indices:

11	29	14	32	17	35
2	20	5	23	8	26
10	28	13	31	16	34
1	19	4	22	7	25
9	27	12	30	15	33
0	18	3	21	6	24

Under a **blocked scheme**, the mesh is regularly subdivided into squares, and the index of a PU is constituted firstly by the index of its square and secondly by its index within this square. Under a **blocked snake-like row-major scheme**, the blocks are indexed by the snake-like row-major scheme and the indexing of the blocks is left unspecified. As an example we give the indexing that is obtained with 2×2 blocks which are indexed by the row-major scheme in a 6×6 mesh:

26	27	30	31	34	35
24	25	28	29	32	33
22	23	18	19	14	15
20	21	16	17	12	13
2	3	6	7	10	11
0	1	4	5	8	9

2.3 Basics of Routing

We assume that a packet not only contains some message but also the information that is necessary for routing the packet to its destination.

We speak of **edge contention** when several packets residing in a PU have to be routed over the same connection. Contentions are resolved using a priority scheme. The **farthest-first strategy** gives priority to the packet that has to go farthest.

For the analysis of the routing on higher dimensional meshes we need the 'routing lemma' for routing a distribution of packets on a one dimensional mesh [8]. Define for a given distribution of packets over the PUs $h_{\text{right}}(i,j) = \#\{$ *packets passing from left to right through both P_i and P_j* $\}$, where P_i denotes the PU with index i. Define $h_{\text{left}}(j,i)$ analogously.

Lemma 1 *Routing a distribution of packets on a linear array with n PUs, using the farthest-first strategy, takes* $\max_{i<j}\{\max\{h_{\text{right}}(i,j), h_{\text{left}}(j,i)\} + j - i - 1\}$ *steps. This bound is sharp.*

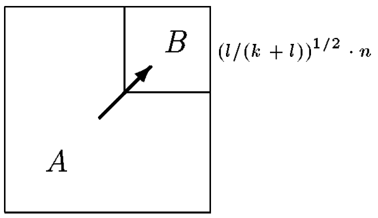
2.4 Lower Bounds

Considering the maximal distance a packet may have to travel, it follows that $d \cdot n - d$ is a lower bound for routing on d dimensional meshes: the **distance bound**. For routing k-k distributions on a mesh $k \cdot n/2$ steps are required when all packets residing in the lower half of the mesh have destination in the upper half: the **bisection bound**. We generalize these bounds to the l-k routing problem.

Considering the number of packets that may have to go over a limited number of connections, we find the following lower bounds for routing l-k distributions:

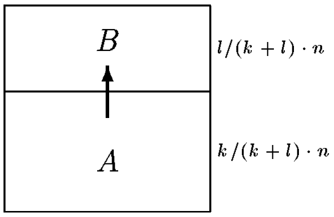
Lemma 2 *The l-k routing problem requires at least* $\max\{\frac{k}{2} \cdot (\frac{l}{l+k})^{1/2},\ \frac{l \cdot k}{l+k},\ \frac{l}{2} \cdot (\frac{k}{l+k})^{1/2}\} \cdot n$ *steps.*

Proof: For the first bound we consider the following subdivision of the mesh:



Suppose that every PU in A holds l packets with destination in B. It is easy to check that these packets just fit into B and that routing them across the boundary of B takes $k/2 \cdot (l/(l+k))^{1/2} \cdot n$ steps.

The second bound is a generalization of the bisection bound. Consider the following subdivision of the mesh:



If all PUs in A hold l packets with destination in B, then it takes for these $l \cdot k/(k+l) \cdot n^2$ packets $l \cdot k/(k+l) \cdot n$ steps to pass across the boundary between A and B.

The third bound is obtained analogously to the first, considering the time that is required when all PUs of a $(k/(k+l))^{1/2} \cdot n \times (k/(k+l))^{1/2} \cdot n$ corner send l packets to the PUs outside this corner. □

Which of the bounds in Lemma 2 is strongest, depends on the relation between l and k: the first bound is strongest when $l \leq k/3$, the second when $k/3 \leq l \leq 3 \cdot k$, and the third when $l \geq 3 \cdot k$. When $l = k$, the second bound gives $k \cdot n/2$, the bisection bound for the k-k routing problem. In the extreme cases the bound becomes very simple:

Corollary 1 *When $l = o(k)$, or $k = o(l)$, then routing l-k distributions requires at least $(1 - o(1)) \cdot \sqrt{l \cdot k} \cdot n/2$ steps.*

3 A Randomized Algorithm

One might think that the following algorithm has good performance:

Send all packets to a random destination;
route a packet to its destination:
  with probability 1/2 row-first,
  with probability 1/2 column-first.

However, algorithms of this type require $\Omega(k \cdot n)$ time for a distribution of packets under which all packets have destination in the highest n/k rows. Such a distribution can be routed in $\mathcal{O}(n)$ steps when all packets are routed row-first. This illustrates the utmost importance of the decision along which axis a packet is routed first. Clearly such a decision cannot be based only on information that is available locally: the 1-k routing problem is essentially more difficult than the permutation routing problem, for which the greedy algorithm has optimal routing time.

3.1 Algorithm

We describe a simple randomized algorithm for the 1-k routing problem that requires $\sqrt{k} \cdot n/2 + o(\sqrt{k} \cdot n)$ routing steps. It proceeds as follows:

Algorithm RANDROUTE;
1. **for** every PU $P_{i,j}$ with packet p **pardo**
   make two copies of the routing information of p;
   send one copy along row i to its dest. column,
   send one copy along column j to its dest. row;
2. **for** every i, j, $0 \leq i, j \leq n-1$, **pardo**
   determine the number r_i of packets that have to move along row i, and broadcast r_i within row i,
   determine the number c_i of packets that have to move along col i, and broadcast c_i within col i;
   discard all copies of packets,
   send r_i along the column to all PUs,
   send c_i along the row to all PUs;
3. **for** every PU $P_{i,j}$ with packet p **pardo**
   if the destination of p is $P_{i',j'}$, **then**
     take $r_{i'}$ and $c_{j'}$ out of the stream;
     color p white with probability $r_{i'}/(r_{i'} + c_{j'})$;
     if p is not colored white **then**
       color p black;
4. **for** all packets p **pardo**
   if p is white **then**
   a. randomize p within its column;
   b. route p along the row to its dest. column;
   c. route p along the column to its destination
   else proceed analogously.

Randomizing a packet p within its column means that the packet is routed to a randomly and uniformly selected position within its column. The randomization is intended to bound the size of the queues.

3.2 Analysis

In the following analysis we do not go into the details of the randomization. The purpose of this section is to express the underlying ideas and to indicate that the performance is near-optimal.

It is easy to implement Step 1 and Step 2 to run in $3 \cdot n - 3$ steps. Step 4 consists of three phases of one-dimensional routing. During phase 4.a the packets are not delaying each other and they all reach the selected PU within $n - 1$ steps. During phase 4.b a delay of $o(n)$ may occur because a PU may hold more than one packet. The delay during phase 4.c is of a different nature and may be much larger than the delay in phase 4.b. It depends on the distribution of the packets over the colors.

Lemma 3 *The number of packets that have to move within a row or column during phase 4.c is bounded by $\sqrt{k} \cdot n/2 + \mathcal{O}((\sqrt{k} \cdot n \cdot \log n)^{1/2})$.*

Proof: Let $a_{i,j}$ be the number of packets with destination in $P_{i,j}$. Let $z_i = \sum_j a_{i,j} \cdot c_j/(r_i + c_j)$, and $s_j = \sum_i a_{i,j} \cdot r_i/(r_i + c_j)$. $c_j/(r_i + c_j)$ is the probability that a packet with destination $P_{i,j}$ is colored black, and $r_i/(r_i + c_j)$ is the probability that a packet is colored white. Hence z_i, s_j respectively, equals the expected number of packets that will move through row i, column j respectively, during phase 4.c. Without loss of generality we may concentrate on s_j. $0 \leq a_{i,j} \leq k$, $0 \leq r_i \leq k \cdot n$. Further, $\sum_{i=0}^{n-1} a_{i,j} = c_j$, and $\sum_{i=0}^{n-1} r_i = n^2$. Considering the functionality of s_j for fixed c_j, taking into account that the sum of the r_i cannot be chosen arbitrarily large, we find that in order to maximize s_j the $a_{i,j}$ must be concentrated in a few rows: $a_{i,j} = k$ for c_j/k values of i, and $r_i = k \cdot n^2/c_j$ for these i. In this case $s_j = c_j \cdot k \cdot n^2/(k \cdot n^2 + c_j^2)$. Differentiation shows that the maximum of this function is assumed for $c_j = \sqrt{k} \cdot n$. For this c_j, $s_j = \sqrt{k} \cdot n/2$. Applying Chernoff bounds (as presented e.g. in [4]), the result follows. □

By the randomization the packets are nicely distributed within their row or column at the beginning of phase 4.c. Now, using Lemma 1, we can show that phase 4.c takes at most $\sqrt{k} \cdot n/2 + \mathcal{O}((\sqrt{k} \cdot n \cdot \log n)^{1/2})$ steps. Thus we get

Theorem 1 *On the $n \times n$ mesh, 1-k distributions can be routed in $\sqrt{k} \cdot n/2 + 5 \cdot n + \mathcal{O}((\sqrt{k} \cdot n \cdot \log n)^{1/2})$ steps by a randomized algorithm.*

Notice that RANDROUTE works correctly without knowing k, and that Theorem 1 holds for all values of k.

4 A Deterministic Algorithm

Randomization enables us to formulate RANDROUTE concisely and without loosing many routing steps. However, both steps involving randomization can be replaced by deterministic steps with the same effect.

At first glance coloring the packets deterministically appears to be difficult. However, this can be achieved by sorting the packets on their destination PU and coloring for every destination regularly interspaced packets white and black. The sorting can be performed in $2 \cdot n + o(n)$ steps. By rounding errors at most $n/2$ extra packets may move through any row or column during the last routing phase (compared to the $\mathcal{O}((\sqrt{k} \cdot n \cdot \log n)^{1/2})$ of the randomized algorithm).

The randomization of the packets can be replaced by sorting the packets that are going to be routed row-first (the white packets) in column-major order, and the packets that are going to be routed column-first (the black packets) in row-major order. This idea goes back on Kunde [10].

An additional idea, by which the routing time can be reduced, is to divide the mesh regularly in $n' \times n'$ submeshes and to route the packets first to any destination within their destination squares. From there the packets are routed to their destinations. n' rows (columns) spanning n/n' submeshes are called a **bundle** of rows (columns).

4.1 Algorithm

In the algorithm we take care that the lengths of the queues never exceed $\mathcal{O}(k)$. For the sake of a simple exposition we choose $n' = \sqrt{n}$.

Algorithm DETROUTE

1. Count in every submesh how many packets are going to any bundle of rows (columns), and store the number going to row (column) bundle i in the PU at position (i, i). Perform a routing along row (column) and column (row) in order to obtain the total number r_i (c_i) of packets going to row (column) bundle i, for all $0 \leq i \leq \sqrt{n} - 1$, in position (i, i) of all submeshes.

2. Sort the packets on the indices of their destination submeshes.

3. The numbers r_i and c_j are broadcast within every $\sqrt{n} \times \sqrt{n}$ submesh. A PU P_l holding a packet p with destination in submesh (i, j) picks r_i and c_j out of the stream. Let $\alpha_{i,j} = r_i/(r_i + c_j)$. If $l \bmod (1/\alpha_{i,j}) < 1$, then p is colored white, else p is colored black.

4. Let $m = n/\sqrt{k}$. Divide the mesh in $m \times m$ squares. Sort in each square the white (black) packets in column-major (row-major) order on their destination column (row) bundles.

5. Route the white (black) packets along the row (column) to the first PU in their destination column (row) bundle holding less than $k + 1$ white (black) packets.

6. In every submesh, sort the white (black) packets in row-major (column-major) order on their row (column) bundle.

7. Route the white (black) packets along the column (row) to the first PU in their destination submesh holding less than $k + 1$ white (black) packets.

8. Route the packets within the submeshes to their destinations.

If the value of k is unknown, then in Step 4 we should use some lower estimate of it, e.g. obtained from the maximum of the r_i and the c_j.

4.2 Analysis

We analyze the correctness and the routing time of DETROUTE. Step 1 takes $2 \cdot n + \mathcal{O}(\sqrt{n})$ steps. After Step 1, the r_i and c_j are locally available in all submeshes. Step 2 can be overlapped almost perfectly:

Lemma 4 *Step 1 and Step 2, can be performed in $2 \cdot n + \mathcal{O}(n^{2/3})$ steps.*

Proof: During the routing of Step 1 only one out of every $\sqrt{n}$ connections is used. This means that most connections can be used for Step 2. For the sorting we use the near-optimal deterministic algorithm of [9]. □

The sorting in Step 2 is for rearranging the packets such that the packets going to the same submesh stand in positions with consecutive indices. Then, in Step 3 the correct fractions of them can be selected. Notice that Step 3 works in a distributed fashion without knowing where the packets going to a certain submesh reside or how many packets are going to a submesh. It seems hard to compute these numbers. Step 4 is performed in order to bound the queues at the end of Step 5.

Lemma 5 *Step 3 takes $\mathcal{O}(\sqrt{n})$ steps; Step 4 takes $\mathcal{O}(n/\sqrt{k})$ steps; and Step 5 takes n steps.*

The most important point in the analysis is the proof of the following analogue of Lemma 3:

Lemma 6 *Step 5 can be performed correctly, i.e. no PU holds more than k white (black) packets afterwards. Step 5 takes $\mathcal{O}(k \cdot \sqrt{n})$ steps. After Step 6 no column (row) holds more than $\sqrt{k} \cdot n/2 + \mathcal{O}(\sqrt{n})$ white (black) packets.*

Proof: Denote the number of packets going to submesh (i, j) by $a_{i,j}$. For S_j, the number of packets that will move through column bundle j, we now get $S_j \leq \sum_i \lceil a_{i,j} \cdot r_i/(r_i + c_j) \rceil < \sum_i a_{i,j} \cdot r_i/(r_i + c_j) + \sqrt{n}$. As in the proof of Lemma 3 this gives that $S_j < \sqrt{k} \cdot n^{3/2}/2 + \sqrt{n}$. These packets are distributed over at least $m \cdot \sqrt{n}$ PUs. Hence at most $\lceil k + \sqrt{k}/n \rceil = k + 1$ packets have to be stored in any PU. After the sorting in Step 5 the S_j packets are almost perfectly distributed over the $\sqrt{n}$ columns of the bundle: at most $\sqrt{k} \cdot n/2 + \mathcal{O}(\sqrt{n})$ packets end in every column. □

Comparable to Lemma 6 but easier is

Lemma 7 *Step 7 can be performed correctly in $\sqrt{k} \cdot n/2 + n - n/\sqrt{k} + \mathcal{O}(\sqrt{n})$ steps. I.e. afterwards no PU holds more than $k+1$ white (black) packets. Step 8 takes $\mathcal{O}(k \cdot \sqrt{n})$ steps.*

Proof: If S_j assumes its maximum then the packets in the column bundle j (that may have to go all to the highest or lowest rows) are spread out over at least m rows. Now Lemma 1 gives the maximum routing time. By the sorting in Step 7 the packets going to a submesh are well distributed over the columns. By 'rounding errors' there may be in a column at most $\sqrt{n}$ packets too much. They can be spread out over the $\sqrt{n}$ rows of the destination submesh. Step 8 is a $(2 \cdot k + 2)$-k routing in $\sqrt{n} \times \sqrt{n}$ meshes. The algorithm of [9] can be used. □

Summing all routing times we find

Theorem 2 DETROUTE *takes $\sqrt{k} \cdot n/2 + 4 \cdot n - n/\sqrt{k} + \mathcal{O}(k \cdot \sqrt{n})$ steps. The size of the queues is bounded by $2 \cdot k + 2$.*

Corollary 2 *For $k = o(n^{1/3})$,* DETROUTE *takes less than $\sqrt{k} \cdot n/2 + 4 \cdot n$ steps.*

4.3 Large k

Corollary 2 expresses that our choice $n' = \sqrt{n}$ is particularly good for $k = o(n^{1/3})$. For $k = n^{\alpha}$, $1/3 \leq \alpha < 2$, the additive term can be bounded to $o(n)$ by taking $n' = n^{1/2-\alpha/4}$. However, for these k we are willing to spend $\mathcal{O}(n)$ extra steps in order to obtain a much simpler algorithm:

Algorithm LARGEKROUTE

1. Sort the packets on the indices of their destinations.

2. Make two copies of the routing information. Send one copy to the destination row and one to the destination column. Count the copies within rows and columns. Broadcast the numbers r_i and c_j back. A PU P_h with a packet going to $P_{i,j}$ picks out r_i and c_j. Let $\alpha_{i,j} = r_i/(r_i + c_j)$. If $h \bmod (1/\alpha_{i,j}) < 1$, then p is colored white, else p is colored black.

3. Sort the white (black) packets in column-major (row-major) order on their destination columns (rows).

4. Route the white (black) packets to their destination columns (rows).

5. Route the white (black) packets along the columns (rows) to their destinations,

Analogously to the lemmas of the previous section we can prove

Lemma 8 *Step 1 takes less than $3 \cdot n$ steps; Step 2 takes $3 \cdot n$ steps; Step 3 takes $6 \cdot n$ steps; Step 4 takes n steps; Step 5 takes $\sqrt{k} \cdot n/2 + n$ steps.*

Theorem 3 LARGEKROUTE *routes 1-k permutations in $\sqrt{k} \cdot n/2 + 14 \cdot n$, for all k. The size of the queues is bounded by $2 \cdot k + 2$.*

We believe that by its simplicity and its generality LARGEKROUTE is a very practical algorithm.

4.4 l-k Routing

We show that LARGEKROUTE is also suited for routing l-k distributions for values of l that are small compared to k.

Now Step 1, ..., Step 4 are l-l routing and sorting procedures:

Lemma 9 *Step 1, ..., Step 4, are performed in $\mathcal{O}(l \cdot n)$ steps.*

More interesting is the following analogue of Lemma 6 and Lemma 3:

Lemma 10 *During Step 5 less than $\sqrt{l \cdot k} \cdot n/2 + n$ packets move through any row or column.*

Proof: Without loss of generality we concentrate on S_j, the number of the number of packets moving through column j. Let $a_{i,j}$ be the number of packets with destination in $P_{i,j}$. $0 \leq a_{i,j} \leq k$, $0 \leq r_i \leq k \cdot n$, $\sum_{i=0}^{n-1} a_{i,j} = c_j$, and $\sum_{i=0}^{n-1} r_i \leq l \cdot n^2$. Consider S_j for fixed c_j. Analogously to the proof of Lemma 3, S_j is maximized if $a_{i,j} = k$ for c_j/k values of i, and $r_i = l \cdot k \cdot n^2/c_j$ for these i. In this case $S_j = c_j \cdot l \cdot k \cdot n^2/(l \cdot k \cdot n^2 + c_j^2)$. For $c_j = \sqrt{l \cdot k} \cdot n$ the maximum value equals $S_j = \sqrt{l \cdot k} \cdot n/2$. Additional n packets may come from rounding errors. □

Theorem 4 *Routing l-k distributions by* LARGEKROUTE *takes $\sqrt{l \cdot k}/2 \cdot n + \mathcal{O}(l \cdot n)$ steps. The size of the queues is bounded by $2 \cdot k + 2$.*

Combining Corollary 1 and Theorem 4, gives

Corollary 3 LARGEKROUTE *is near-optimal for routing l-k distributions with $l = o(k)$.*

The choice of an algorithm should be tuned in accordance with the ratio between l and k: for l that are smaller than k LARGEKROUTE is taken; for l that are comparable to k, the algorithm of [9] can be applied. And for l that are larger than k a 'reversal' of LARGEKROUTE should be applied with a coloring of the packets that depends on the initial distribution of the packets.

5 Routing in Many Stages

We apply ideas of Section 4.4 to show that under certain, often satisfied, conditions l-k routing can be performed in $\mathcal{O}(l \cdot n)$ steps, even for k that are much larger than l. Throughout this section $l = o(k)$.

Assume that the destinations of the packets are distributed such that the density of the destinations increases inversely proportional to the size of the considered submesh. Formally,

Definition 2 *Consider an l-k routing problem on an $n \times n$ mesh. Let $\alpha = (\log k - \log l)/logn$. The problem has continuously increasing density with parameter α, if for any $n/x \times n/x$ submesh, $1 \leq x \leq n$, there are at most $l \cdot x^\alpha \cdot (n/x)^2$ packets with destination in this submesh.*

This definition reflects a natural situation. E.g., a problem with continuously increasing density arises when all PUs generate packets with a random destination.

The previously derived lower bounds no longer hold. For an l-k routing problem we remain with the trivial bound $\max\{2 \cdot n - 2, l \cdot n/2, k\}$. Routing the packets to their destination with LARGEKROUTE of Section 4.4 does not exploit the particular properties of the distribution: we would get a routing time of $(1 + o(1)) \cdot \sqrt{l \cdot k}/2 \cdot n$. Routing the packets in $\log n$ stages through smaller and smaller submeshes to their destination requires only $\mathcal{O}(l \cdot n)$ steps. This is the optimal time order. The algorithm is simple:

Algorithm STAGEROUTE;
for $i := 1$ **to** $\log n$ **do**
1. divide the mesh regularly in $n/2^i \times n/2^i$ submeshes;
2. route all packets to their destination submeshes;
3. redistribute the packets within the submeshes.

STAGEROUTE runs correctly without knowing α.

An important point is the implementation of the routing within the stages. An elegant algorithm guarantees an optimal distribution of the packets within the submeshes. We give a slightly more general description: within an $n_1 \times n_1$ mesh in which a PU holds at most k_1 packets, the packets have to be routed to the $n_2 \times n_2$ submeshes of their destinations, and at most $k_2 \cdot n_2^2$ packets have destination in every submesh. The submeshes are indexed from 0 through $n_1^2/n_2^2 - 1$, the PUs in every submesh from 0 through $n_2^2 - 1$. We perform:

1. Sort all packets in the $n_1 \times n_1$ mesh on the numbers of their destination submeshes.

2. Route the packet with rank i to the PU in its destination submesh which has index $i \bmod (n_1^2/n_2^2)$.

Clearly Step 1 is a k_1-k_1 sorting; Step 2 is a k_1-k_2 routing. Thus, the routing in the stages can be performed as fast as a k_1-k_2 routing, in $\sqrt{k_1 \cdot k_2}/2 \cdot n_1 + \mathcal{O}(k_1 \cdot n_1)$ steps.

Theorem 5 *If $\alpha < 1$, then* STAGEROUTE *solves an l-k routing problem with continuously increasing density and parameter α, in $\mathcal{O}(l \cdot n)$ routing steps.*

Proof: In phase i the packets are routed within $n/2^{i-1} \times n/2^{i-1}$ submeshes to the $n/2^i \times n/2^i$ submeshes in which their destination lies. Because we consider a routing problem with continuously increasing density with parameter α, the initial density is at most $l \cdot 2^{(i-1) \cdot \alpha}$, and the final density is at most $l \cdot 2^{i \cdot \alpha}$. Hence, phase i can be performed in $(l \cdot 2^{(i-1/2) \cdot \alpha}/2 + \beta \cdot l \cdot 2^{(i-1) \cdot \alpha}) \cdot n/2^{i-1} \leq (\beta+1) \cdot l \cdot n/2^{(i-1) \cdot (1-\alpha)}$, for some positive constant β. Thus, all phases together take at most $\sum_{i=1}^{\log n} (\beta+1) \cdot l \cdot n/2^{(i-1) \cdot (1-\alpha)} < (\beta+1)/(1-2^{1-\alpha}) \cdot l \cdot n$. □

6 Working Queue Size 3

Apart from its own importance, the 1-k routing problem is interesting in view of Corollary 2.3 from [15]:

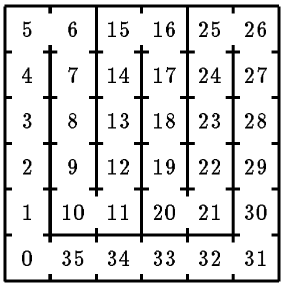
Theorem 6 *If there is a* 1-k *routing algorithm for the* $n \times n$ *mesh in which the routing decisions are made locally, which requires* $T(k,n)$ *steps and has maximal working queue size at most four, then there is a hot-potato worm-hole routing algorithm for worms of maximal length* $k \leq n^2/5$ *requiring* $\mathcal{O}(k^2 \cdot T(k, n/\sqrt{k}))$ *steps.*

DETROUTE of Section 4.1 cannot be used because the size of the working queues is too large. We would like to have a 1-k routing algorithm with $T(k,n) = \mathcal{O}(\sqrt{k} \cdot n)$, and $Q \leq 4$. Once we have shown that this can be achieved, we get

Theorem 7 *Hot-potato worm-hole routing for worms of maximal length* $k \leq n^2/5$ *can be performed in* $\mathcal{O}(k^2 \cdot n)$ *steps.*

6.1 Algorithm

In contrast to the algorithm of the previous section we do not strive for near-optimal performance, we are mainly interested in achieving the right order of magnitude of the routing time. We assume that the PUs of the mesh are indexed in some suitable way from 0 through $n^2 - 1$, for example, in the snake-like row-major order. Furthermore, the mesh is divided in k squares of size $n/\sqrt{k} \times n/\sqrt{k}$. Each such square is indexed from 0 through $n^2/k - 1$. The squares lie on a Hamiltonian cycle. For example, when $k = 36$ as follows:



The algorithm proceeds as follows:

Algorithm SHORTQROUTE(k, n);
1. Sort the packets with respect to the index of their destination PU;
2. route the packet in the PU with index i to the PU with index i **div** k in square i **mod** k;
3. **for** $s := 1$ **to** k **do**
   a. the packets of a square that have not yet reached their destination are routed as a block to the next square of the Hamiltonian cycle;
   b. route the packets that reached their destination square to their final destination.

The essential step of SHORTQROUTE is Step 2. In this step, the packets are distributed over the squares such that there is at most one packet with a certain destination in each square. This implies that after each iteration of Step 3.a, at most one packet has to be routed to a destination in Step 3.b. Hence, the routing in Step 3.b is a partial 1-1 routing within the squares.

For the routing and sorting operations in SHORTQROUTE we apply the algorithm of Schnorr and Shamir [19]. By this algorithm, the packets of an $n \times n$ are sorted in $3 \cdot n + \mathcal{O}(n^{3/4})$ steps in snake like order. This result does not require that the connections act as comparators when we accept queue size two. Partial permutation routing with short queues is slightly harder:

Lemma 11 *On an* $n \times n$ *mesh partial* 1-1 *routing can be performed in* $4 \cdot n + \mathcal{O}(n^{3/4})$ *steps and with working queue size two.*

Proof: We use a variant of the algorithm of Kunde [10]. The mesh is divided in $n/2 \times n/2$ submeshes. In every submesh the packets are sorted in snake-like column-major order on their destination columns with the algorithm of [19]. This takes $3 \cdot n/2 + \mathcal{O}(n^{3/4})$ steps. In $n/2$ steps this is turned into a column-major order. Subsequently the packets are routed row-first in $2 \cdot n$ steps to their destinations. □

Theorem 8 SHORTQROUTE *routes* 1-k *permutations in* $5 \cdot \sqrt{k} \cdot n + 6 \cdot n + \mathcal{O}(k^{5/8} \cdot n^{3/4})$ *steps and with working queue size three.*

Proof: Step 3.a takes $n/\sqrt{k}$ steps, Step 3.b takes $4 \cdot n/\sqrt{k} + \mathcal{O}((n/\sqrt{k})^{3/4})$ steps. These steps are performed k times. Step 1 and Step 2 take $3 \cdot n + \mathcal{O}(n^{3/4})$ each. So, the whole algorithm requires $6 \cdot n + \mathcal{O}(n^{3/4}) + k \cdot (n/\sqrt{k} + 4 \cdot n/\sqrt{k} + \mathcal{O}(n^{3/4}/k^{3/8}))$ steps. The working queue size of the routing in the single steps is two, but during Step 3.b a PU may hold in addition a packet that is waiting for being routed to the next square. □

We conclude that for all $k = \mathcal{O}(n^2)$ the time for routing 1-k distributions is bounded by $\mathcal{O}(\sqrt{k} \cdot n)$. Hence Theorem 7 holds.

Notice that in the algorithm we assumed that k was known. This assumption is not essential: after Step 1 the value of k can be determined and broadcast to all PUs in $\mathcal{O}(n)$ steps.

6.2 Improvement

A routing or sorting algorithm is called **uni-axial**, if in each routing step either only horizontal or only vertical connections are used. If a routing or sorting algorithm is uni-axial two of these algorithms can be run in parallel without interference: one for 'white' packets and one for 'black' packets, which are routed orthogonally at all times. This idea goes back on [12].

The sorting algorithm of Schnorr and Shamir [19] which is used in SHORTQROUTE can be made uni-axial with a loss of $\mathcal{O}(n^{3/4})$ steps. This opens the way to an interesting reduction of the routing time: the mesh is divided into $k/2$ squares of side length $\sqrt{2/k} \cdot n$, and a Hamiltonian cycle through these $k/2$ squares is chosen. Step 2 and 3 of the algorithm are replaced by

2. Color the packet p in the PU with index i white if i **mod** $k < k/2$, black otherwise; route p to the PU with index i **div** $k/2$ in square i **mod** $k/2$;
3. **for** $s := 1$ **to** $k/2$ **do**
   a. the packets of a square that have not yet reached their destination are routed as a block to the next square of the Hamiltonian cycle;
   b. route the packets that reached their destination square to their final destination, white packets orthogonally to black packets.

The modified algorithm will be referred to as SHORTQROUTE$'$.

For the routing in Step 3.b we need the following analogue of Lemma 11:

Lemma 12 *On an $n \times n$ mesh two partial 1-1 routings can be performed in $5 \cdot n + \mathcal{O}(n^{3/4})$ steps and with working queue size two.*

Proof: The white packets are sorted in $3 \cdot n + \mathcal{O}(n^{3/4})$ steps in snake-like column-major order. Then they are routed row-first. The black packets are routed orthogonally. Without further attention it might happen that during the sorting two white and two black packets reside in a single PU at the same time. However, there are two packets in a single PU only during the phases of odd-even transposition sort. Hence, it is easy to arrange that two white packets reside in the PUs $P_{i,j}$ with $i + j$ odd, only when two black packets reside in the PUs with $i + j$ even. □

Theorem 9 SHORTQROUTE$'$ *routes 1-k permutations in $6/\sqrt{2} \cdot \sqrt{k} \cdot n + 6 \cdot n + \mathcal{O}(k^{5/8} \cdot n^{3/4})$ steps and with working queue size three.*

Proof: There are $k/2$ passes of the loop which take $(1 + 5) \cdot \sqrt{2/k} \cdot n + \mathcal{O}((n/\sqrt{k})^{3/4})$ each. □

SHORTQROUTE$'$ is slower than optimal by a factor $6/\sqrt{2} \simeq 8.5$. For higher-dimensional meshes exploiting uni-axiality gives a larger gain (see Section 7.1).

7 Higher Dimensional Meshes

The algorithms LARGEKROUTE of Section 4.4 and SHORTQROUTE$'$ of Section 6.2 are based on ideas that are suited for generalization to other machines that have some similarity with meshes. Also these ideas might be applied successfully to related problems.

In this section we concentrate on the 1-k routing problem on d-dimensional $n \times \cdots \times n$ meshes, for which the generalizations can be given most easily. Analogously to Lemma 2 we can prove that this problem requires at least $\max\{d \cdot (n-1), k^{1-1/d} \cdot n/d\}$ steps.

7.1 Generalizing SHORTQROUTE$'$

For the algorithm SHORTQROUTE$'$ of Section 6.2 there is a natural generalization: the mesh is subdivided in k/d 'cubes' of size $(d/k)^{1/d} \cdot n \times \cdots \times (d/k)^{1/d} \cdot n$. Through these cubes a Hamiltonian cycle is laid out. Slight modification of SHORTQROUTE$'$ suffices:

Algorithm HIGHDIMROUTE(k, n, d);
1. Sort the packets with respect to the index of their destination PU;
2. route a packet p in the PU with index i to the PU with index i **div** k/d in cube i **mod** k/d;
3. **for** $s := 1$ **to** k/d **do**
   a. the packets of a cube that have not yet reached their destination are routed as a block to the next cube of the Hamiltonian cycle (from cube $k/d - 1$ to cube 0);
   b. route the packets that reached their destination cube to their destination.

For an efficient implementation of HIGHDIMROUTE it is important to have

Lemma 13 *For d-dimensional meshes a k-k sorting or routing problem with $k \leq d$ can be solved in $\mathcal{O}(d \cdot n)$ steps with maximal queue size $\max\{2, k\}$.*

Proof: Particularly suited is the k-k sorting algorithm of [9]. In this algorithm regular routing operations are alternated with sorting operations on submeshes. The algorithm is recursive. By the time the submeshes consist of less than n PUs, the sorting can be finished by an odd-even transposition sort of an embedded linear array. □

Theorem 10 HIGHDIMROUTE *routes 1-k distributions on a d-dimensional mesh in $\mathcal{O}(d \cdot n + k^{1-1/d} \cdot n)$ steps with working queue size $d + 1$.*

Proof: Step 1 and 2 take $\mathcal{O}(d \cdot n)$ steps. Then we have k/d passes of the loop in Step 3. Each pass involves a routing part, requiring $(d/k)^{1/d} \cdot n$ steps, and the solution of a 1-d routing problem in a d-dimensional mesh with side length $(d/k)^{1/d} \cdot n$. Using Lemma 13, we find that this takes $\mathcal{O}(d \cdot (d/k)^{1/d} \cdot n)$ steps. Summing and noticing that $d^{1/d} < 2$, we find the stated result. The queue size equals one plus the d from Lemma 13. □

The routing time of HIGHDIMROUTE is a factor d larger than the lower bound. For application of a 1-k routing algorithm as subroutine of a worm-hole routing algorithm we must have $Q \leq 2 \cdot d$. This requirement is met.

7.2 Generalizing LARGEKROUTE

We give a generalization of LARGEKROUTE of Section 4.4 for routing on d-dimensional meshes. The algorithm consists of d phases, Phase 0 through Phase $d-1$. During Phase f the routing is performed within submeshes of dimension $d - f$. After phase 0 we can no longer assume that all PUs initially hold only one packet. Therefore, and for the sake of generality, it is better to consider the l-k routing problem from the start.

For routing l-k distributions on d-dimensional meshes $(l/(l+k))^{1/d} \cdot k \cdot n/d$ is a lower bound. This can be seen as in the proof of Lemma 2. This gives us an analogue of Corollary 1:

Corollary 4 *When $l = o(k)$, then routing l-k distributions on d-dimensional meshes requires at least $(1 - o(1)) \cdot l^{1/d} \cdot k^{1-1/d} \cdot n/d$ steps.*

We describe Phase 0 of the algorithm (corresponding to Step 1, ..., Step 4 of LARGEKROUTE):

1. Sort the packets on the indices of their destinations.

2. Make d copies of the routing information of each packet. Give color 0 through $d-1$ to the copies. For a packet with destination $P_{j_0,\ldots,j_{d-1}}$, the copy with color x, $0 \leq x \leq d-1$, is routed along axis x to the $(d-1)$-dimensional hyperplane of its destination.

3. For all $0 \leq x \leq d-1$, $0 \leq l \leq n-1$, determine the number $r_l^{(x)}$ of packets of color x in the hyperplane $(*, \ldots, *, l, *, \ldots, *)$, normal to axis x, and broadcast it within the hyperplane.

4. For all $0 \leq x \leq d-1$, $0 \leq l \leq n-1$, send $r_l^{(x)}$ along axis x. A PU holding a packet with destination $P_{j_0,\ldots,j_{d-1}}$, picks out $r_{j_0}^{(0)}, \ldots, r_{j_{d-1}}^{(d-1)}$.

5. A packet p with destination $P_{j_0,\ldots,j_{d-1}}$ resides in P_m, $0 \leq m \leq n^d - 1$. For $0 \leq x \leq d-1$, let $\alpha^{(x)} = (1 - r_{j_x}^{(x)} / \sum_{y=0}^{d-1} r_{j_y}^{(y)})/(d-1)$. Let x be the smallest number such that $m \bmod (1/\sum_{y=0}^{x} \alpha^{(x)}) < 1$. Give p color x.

6. Sort the packets of each color. The packets with color x are sorted on j_x with respect to an axis x major indexing scheme.

7. Route the packets of color x along axis x to the $d-1$-dimensional hyperplanes of their destinations.

Step 1 and Step 5 together constitute a deterministic equivalent of the randomization statement: "give color x to p with probability α_x". Step 6 is the translation of randomizing packets with color x in the hyperplane normal to axis x. The subsequent phases are performed analogously within submeshes of decreasing dimension.

Three-Dimensional Meshes. We analyze the algorithm for routing l-k distributions on three-dimensional meshes. And prove an analogue of Lemma 10:

Lemma 14 *After Step 6 there are less than $(4^{2/3}/6 \cdot l^{2/3} \cdot k^{1/3} + 1) \cdot n^2$ packets, that have to move in any two-dimensional hyperplane.*

Proof: Consider $S_0^{(2)}$, the number of packets that has to move through $(*, *, 0)$. $S_0^{(2)} \leq \sum_{i,j} \lceil a_{i,j,0} \cdot \alpha_{i,j,0}^{(2)} \rceil < n^2 + \sum_{i,j} a_{i,j,0} \cdot (r_i^{(0)} + r_j^{(1)})/(r_i^{(0)} + r_j^{(1)} + r_0^{(2)})/2$. Let $a_{i,j,0}$ be the number of packets with destination in $P_{i,j,0}$. $a_{i,j,0} \leq k$, $r_i^{(0)}, r_j^{(1)} \leq k \cdot n^2$, $\sum_{i,j} a_{i,j,0} = r_0^{(2)}$, and $\sum_i r_i^{(0)}, \sum_j r_j^{(1)} \leq l \cdot n^3$. Consider $S_0^{(2)}$ for fixed $r_0^{(2)}$. Analogously to the proof of Lemma 3, S_j is maximized if $a_{i,j,0} = k$ for $r_l^{(2)}/k$ pairs (i,j) forming a square, and $r_i^{(0)} = r_j^{(1)} = l \cdot (k/r_0^{(2)})^{1/2} \cdot n^3$ for these i and j. This gives

$$\begin{aligned} S_0^{(2)} &< n^2 + \frac{r_0^{(2)}}{k} \cdot \frac{k}{2} \cdot \frac{2 \cdot l \cdot (k/r_0^{(2)})^{1/2} \cdot n^3}{r_0^{(2)} + 2 \cdot l \cdot (k/r_0^{(2)})^{1/2} \cdot n^3} \\ &= n^2 + \frac{l \cdot k^{1/2} \cdot n^3 \cdot r_0^{(2)}}{2 \cdot l \cdot k^{1/2} \cdot n^3 + {r_0^{(2)}}^{3/2}}. \end{aligned}$$

Differentiation shows that the maximum is assumed for $r_0^{(2)} = 4^{2/3} \cdot l^{2/3} \cdot k^{1/3} \cdot n^2$. For this $r_0^{(2)}$ we find $S_0^{(2)} < n^2 + 4^{2/3}/6 \cdot l^{2/3} \cdot k^{1/3} \cdot n^2$. □

So, after redistributing the packets within the two-dimensional submeshes, we may assume that a PU holds at most $4^{2/3}/6 \cdot l^{2/3} \cdot k^{1/3} + 1$ packets. For the resulting l'-k routing problem we apply Lemma 10. This gives

Theorem 11 *l-k distributions can be routed on a three-dimensional $n \times n \times n$ mesh in $4^{1/3}/6^{1/2} \cdot l^{1/3} \cdot k^{2/3} \cdot n + \mathcal{O}(l^{2/3} \cdot k^{1/3} \cdot n)$ steps.*

Proof: A one-dimensional hyperplane lies on the intersection of two two-dimensional hyperplanes. Applying Lemma 10 with $l' = 4^{2/3}/6 \cdot l^{2/3} \cdot k^{1/3}$, we find that from each of them at most $(4^{2/3}/6 \cdot l^{2/3} \cdot k^{1/3} + 1)^{1/2} \cdot k^{1/2} \cdot n/2$ packets are routed to the one-dimensional hyperplane. □

$4^{1/3}/6^{1/2} \simeq 0.65$. Hence, for l small in comparison to k, the algorithm takes less than twice as much steps as given by the lower bound of Corollary 4.

The given analysis is not tight: the maximum was computed for each phase independently, requiring different values of $r_i^{(0)}$ and different distributions of destinations to be achieved. Furthermore, in the final result we simply doubled the number of packets moving in a one-dimensional hyperplane because it receives contributions from both planes on which it lies. The algorithm can easily be modified such that the contributions from both planes are taken into account. This gives a reduction of the constant of the routing time. We believe that after this modification a precise analysis will demonstrate that the algorithm is near-optimal.

The Case $d > 3$. For the analysis of the algorithm for higher dimensions we must develop step-by-step Lemma 14 and Theorem 11 for higher and higher dimensions. The case $d = 3$ encompasses all necessary details, for larger d there are no special problems.

8 Short Queue k-k Routing

The k-k routing problem has been solved near-optimally [11, 9] but these algorithms require working queues of size $\Omega(k)$. Considering application as a subroutine for hot-potato worm-hole routing, we would like to route k-k distributions with $Q \leq 4$. Namely, next to Theorem 6, corollary 2.3 of [15] opens the way to even faster hot-potato worm-hole routing:

Theorem 12 *If there is a k-k routing algorithm for the $n \times n$ mesh in which the routing decisions are made locally, which requires $T(k, n)$ steps and has maximal working queue size at most four, then there is a hot-potato worm-hole routing algorithm for worms of maximal length $k \leq n^2/5$ requiring $\mathcal{O}(k \cdot T(k, n/\sqrt{k}))$ steps.*

When every PU is the origin and destination of at most k packets, then the bipartite graph of all source/destination pairs has maximal degree k. As a corollary to Hall's matching theorem [14, Th. 1.17], this implies that the graph has an edge-coloring with k colors. The source/destination pairs corresponding to each of these colors constitute a (partial) permutation. And a permutation can be routed in $\mathcal{O}(n)$ steps and with working queue size two. Hence, if such a k-coloring would be given, then it is trivial to route k-k distributions with $T = \mathcal{O}(k \cdot n)$ and working queue size two. Particularly this applies to the situation in which every PU holds k packets with the same destination. So, the problem is to split the set of packets in k subsets for which the source/destination pairs approximate 1-1 distributions *on-line*. A bad splitting might result in k 1-k routing problems, taking $k^{3/2} \cdot n$ steps in total. On the other hand, for a $\mathcal{O}(k \cdot n)$ routing time, we do not need a perfect splitting at all: we will show that it suffices to have a splitting into $\mathcal{O}(k)$ slices, which are defined as follows:

Definition 3 *The mesh is divided regularly in $n/\sqrt{k} \times n/\sqrt{k}$ submeshes, $\mathcal{M}_0, \ldots, \mathcal{M}_{k-1}$. A subset of the packets $\mathcal{S}$ is a* **slice** *if $\#\{p \in \mathcal{S}|$ destination p in $\mathcal{M}_s\} \leq n^2/k$, for all $0 \leq s \leq k - 1$.*

The packets of a slice can be routed in $\mathcal{O}(n)$ steps to their destinations as follows:

Algorithm SLICEROUTE

1. Sort all packets on the indices of their destination submeshes.

2. Route the packet with rank r to the PU in its destination submesh with index $r \bmod (n^2/k)$.

3. Route the packets within the submeshes to their destinations.

Lemma 15 *For $k = \mathcal{O}(n)$,* SLICEROUTE *can be performed in $\mathcal{O}(n)$ steps with working queue size three.*

Proof: Applying the algorithm of [19], Step 1 is performed in $3 \cdot n + o(n)$ steps and with working queue size two. Step 2 is a partial permutation routing because of the definition of a slice. By Lemma 11 it can be performed in $4 \cdot n + o(n)$ steps and with working queue size two. Step 3 is a (partial) 1-k routing in $n/\sqrt{k} \times n/\sqrt{k}$ meshes. By Theorem 9 it can be performed in $6/\sqrt{2} \cdot n + 6 \cdot n/\sqrt{k} + \mathcal{O}(k^{1/4} \cdot n^{3/4})$ steps with working queue size three. □

8.1 A Randomized Algorithm

A k-k distribution can easily be split into $2 \cdot k$ slices by the following randomized algorithm:

Algorithm RANDCOLOR;
for all PUs **pardo**
generate a random injection
$I : \{0, \ldots, k-1\} \rightarrow \{0, \ldots, 2 \cdot k - 1\}$;
for all $0 \leq h \leq k - 1$ **do** give color $I(h)$ to packet h.

RANDCOLOR does not take a single routing step.

Lemma 16 *The packets with color c, $0 \leq c \leq 2 \cdot k - 1$, constitute a slice with high probability.*

Proof: The expected number of packets with color c with destination in $\mathcal{M}_s$ is $n^2/(2 \cdot k)$. With Chernoff bounds it is easy to show that then the actual number of these packets is smaller than n^2/k with high probability. □

Now k-k distributions can be routed by first applying RANDCOLOR and then $2 \cdot k$ times SLICEROUTE.

Theorem 13 *By a randomized algorithm k-k routing can be performed in $\mathcal{O}(k \cdot n)$ time and with working queue size three.*

Combining Theorem 12 and Theorem 13 gives

Theorem 14 *By a randomized algorithm, hot-potato worm-hole routing for worms of maximal length k can be performed in $\mathcal{O}(k^{3/2} \cdot n)$ steps. The result holds with high probability, and for all $k \leq n^2/5$.*

8.2 A Deterministic Algorithm

It appears impossible to perform a deterministical coloring without spreading information. Therefore, in this section we slightly extend the assumptions of the model:

Assumption 1 *For packets in the working queues*

- *the keys can be checked and compared;*
- *computational operations can be performed on a data field;*
- *the information in the data field can be 'read'.*

The last two points are knew. They imply that a packet can be used to gather information about the distribution of packets.

Under Assumption 1, the packets within row i, $0 \leq i \leq n-1$, with destination in one of the $n/\sqrt{k} \times n/\sqrt{k}$ submeshes $\mathcal{M}_s$, $0 \leq s \leq k-1$, can be colored by the algorithm hereafter. The PUs in this row are denoted P_j, $0 \leq j \leq n-1$. Let x_j be the number of packets in P_j with destination in $\mathcal{M}_s$, and let $X_j = \sum_{h=0}^{j} x_h$ be their prefix sum.

Algorithm DETCOLOR(j)

1. Release from P_0 a packet p with key value 0 and with destination P_{n-1}. In p we monitor X_j.

2. A packet that has rank r among the packets in P_j going to $\mathcal{M}_s$ is attributed color $(X_{j-1}+r) \bmod (2\cdot k)$.

3. Discard p in P_{n-1}.

For coloring all packets in a row, we can start k copies of DETCOLOR one after the other. Thus, all packets can be colored in $k+n-1$ steps, with working queue size one.

Lemma 17 *If $k \leq n/2$, then the packets with color c, $0 \leq c \leq 2\cdot k-1$, constitute a slice.*

Proof: Let $X(i,s) = X_{n-1}(i,s)$ be the number of packets in row i with destination in $\mathcal{M}_s$. In total at most $\sum_{i=0}^{n-1} \lceil X(i,s)/(2\cdot k)\rceil < n + \sum X(i,s)/(2\cdot k) = n + n^2/(2\cdot k)$ of these packets get color c, for any c. If $k \leq n/2$, then $n + n^2/(2\cdot k) \leq n^2/k$. □

A small problem with the constructed coloring is that a PU may hold several packets with the same color. However, this is not serious:

Lemma 18 *If $k \leq n/2$, then there are at most n packets with color c, $0 \leq c \leq 2\cdot k-1$, in any row.*

Proof: In total at most $\sum_{s=0}^{k-1} \lceil X(i,s)/(2\cdot k)\rceil < k + \sum X(i,s)/(2\cdot k) = k + k\cdot n/(2\cdot k) = k + n/2$ of the packets in row i get color c, for any c. □

Corollary 5 *The packets with color c, $0 \leq c \leq 2\cdot k-1$, can be redistributed within their rows such that every PU holds at most one packet of color c. This takes $2\cdot n$ steps and working queue size one.*

Proof: By Assumption 1, within every row the rank of the packets with color c can be determined in $n-1$ steps. Then the packet with rank r moves to P_r. If it has to travel d steps, it starts to move after $2\cdot n-d$ steps. □

Now k-k distributions can be routed by first applying DETCOLOR and then $2\cdot k$ times SLICEROUTE preceded by the redistribution step.

Theorem 15 *Under Assumption 1, k-k routing can be performed by a deterministic algorithm in $\mathcal{O}(k\cdot n)$ time and with working queue size three, for all $k \leq n/2$.*

Proof: Use Lemma 15, Lemma 17 and Corollary 5. □

Combining Theorem 12 and Theorem 15 gives

Theorem 16 *Under Assumption 1, hot-potato worm-hole routing for worms of maximal length $k \leq n/2$, can be performed in $\mathcal{O}(k^{3/2}\cdot n)$ steps.*

Acknowledgement

We thank Assaf Schuster for triggering our interest in the 1-k routing problem. The idea to route in stages (Section 5) originated from a shared work together with Andrea Pietracaprina and Geppino Pucci.

9 Conclusion

We analyzed the 1-k routing problem, and presented a deterministic near-optimal algorithm for it. We also presented an algorithm with very short working queue size, which is useful as a subroutine for hot-potato worm-hole routing. The results were extended to l-k routing and for routing on higher dimensional meshes. Finally we considered k-k routing with short working queues.

We developed several ideas that may be further exploited. The most important ideas are: (1) the information gathering, which can be used for coloring the packets in an intelligent way; (2) a deterministic selection procedure based on local sorting; (3) routing packets along a Hamiltonian cycle, by which a 1-k problem can be reduced to repeatedly routing permutations.

Future research might consider dynamic variants of the problem and the algorithms. Somehow one must assure that the number of active packets remains n^2. This is the case when it is assumed that only a PU that received a packet generates a new packet. Our near-optimal time algorithm can be adapted to this model.

It seems more difficult to handle dynamically generated packets in an algorithm with very short working queues. Another open problem is whether deterministic k-k routing with short working queues can be performed without any assumptions.

References

[1] Athas, W.C., 'Physically Compact, High Performance Multicomputers,' *MIT Conference on Advanced Research in VLSI*, pp. 302-313, 1990.

[2] Chlebus, B.S., M. Kaufmann, J.F. Sibeyn, 'Deterministic Permutation Routing on Meshes,' *Proc. 5th Symp. on Parallel and Distributed Proc.*, IEEE, 1993, to appear.

[3] Dally, W.J., 'Virtual Channel Flow Control,' *17th Symp. on Computer Architecture*, pp. 60-68, ACM, 1990.

[4] Hagerup, T., C. Rüb, 'An Efficient Guided Tour of Chernoff Bounds,' Inf. Proc. Lett. 33, 305-308, 1990.

[5] Feige, U., P. Raghavan, 'Exact Analysis of Hot-Potato Routing,' *Proc. 33rd Symp. on Foundations of Computer Science*, pp. 553-562, IEEE, 1992.

[6] Felperin, S., P. Raghavan, E. Upfal, 'A Theory of Wormhole Routing in Parallel Computers,' *Proc. 33rd Symp. on Foundations of Computer Science*, pp. 563-572, IEEE, 1992.

[7] Hillis, W.D., 'The Connection Machine,' *MIT Press*, 1985.

[8] 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.

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

[10] Kunde, M., 'Routing and Sorting on Mesh Connected Processor Arrays,' *Proc. VLSI Algorithms and Architectures*, Lecture Notes in Computer Science, 319, pp. 423-433, Springer-Verlag, 1988.

[11] Kunde, M., 'Block Gossiping on Grids and Tori: Deterministic Sorting and Routing Match the Bisection Bound,' *Proc. European Symp. on Algorithms*, 1993.

[12] Kunde, M., T. Tensi, 'Multi-Packet Routing on Mesh Connected Processor Arrays,' *Proc. Symposium on Parallel Algorithms and Architectures*, pp. 336-343, ACM, 1989.

[13] Leighton, T., F. Makedon, Y. Tollis, 'A $2n-2$ Step Algorithm for Routing in an $n \times n$ Array with Constant Size Queues,' *Proc. Symposium on Parallel Algorithms and Architectures*, pp. 328-335, ACM, 1989.

[14] Leighton, T., *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan-Kaufmann Publishers, San Mateo, California, 1992.

[15] Newman, I., A. Schuster, 'Hot-Potato Worm Routing as almost as easy as Store-and-Forward Packet Routing,' *Proc. ISTCS*, 1993.

[16] Noakes, M., W.J. Dally, 'System Design of the J-Machine,' *MIT Conference on Advanced Research in VLSI*, pp. 179-194, 1990.

[17] Peleg, D., E. Upfal, 'The Generalized Packet Routing Problem,' *Theoret. Computer Sc.*, 53, pp. 281-293, 1987.

[18] Rajasekaran, S., Th. Tsantilas, 'Optimal Routing Algorithms for Mesh-Connected Processor Arrays', *Algorithmica*, 8, pp. 21-38, 1992.

[19] Schnorr, C.P., A. Shamir, 'An Optimal Sorting Algorithm for Mesh Connected Computers,' *Proc. 18th Symposium on Theory of Computing*, pp. 255-263, ACM, 1986.

[20] Seitz, et al., 'The Architecture and Programming of the Ametek Series 2010 Multicomputer', *3rd Conference on Hypercube Concurrent Computers and Applications*, pp. 33-36, ACM, 1988.

[21] Sibeyn, J.F., *Algorithms for Routing on Meshes*, Ph. D. Thesis, Universiteit Utrecht, Utrecht, 1992.

[22] Smith, B., 'Architecture and Applications of the HEP Multiprocessor Computer System,' *Proc. 4th Real Time Signal Processing*, pp. 241-248, 1981.