Optimal Wormhole Routing in the (n,d)-Torus[*]

Stefan Bock
Department of Economics

Friedhelm Meyer auf der Heide, Christian Scheideler
Department of Mathematics and Computer Science
and Heinz Nixdorf Institute

University of Paderborn, 33095 Paderborn, Germany

Abstract

In this paper we consider wormhole routing in a d-dimensional torus of side length n. In particular, we present an optimal randomized algorithm for routing worms of length up to $O(n/(d\log n)^2)$, one per node, to random destinations. Previous algorithms only work optimally for two dimensions, or are a factor of $\log n$ away from the optimal running time. As a by-product, we develop an algorithm for the 2-dimensional torus that guarantees an optimal runtime for worms of length up to $O(n/(\log n)^2)$ with much higher probability than all previous algorithms.

1. Introduction

Since communication between several processors requires a large portion of the runtime of a parallel algorithm, it is very useful to construct efficient algorithms for sending information in a network.

In this paper we will propose a new *wormhole routing* algorithm for the (n,d)*-torus*. This network is defined as follows:

Definition 1.1 *The (n,d)-torus $T(n,d)$ consists of a set of processors $\mathcal{P} = \{(a_d, a_{d-1}, a_{d-2}, \dots, a_2, a_1) |\ 0 \leq a_i \leq n-1\ \forall i \in \{1, \dots, d\}\}$ and a set of edges $E = \{\{u,v\}|$ $u,v \in \mathcal{P}$, $\exists i$: $\forall j \in \{1, \dots, d\}$ with $j \neq i$: $u_j = v_j$ and $(u_i + 1) \bmod n = v_i$ $\}$.*

Most parallel and distributed systems utilize either *store-and-forward* or *wormhole routing*. In store-and-forward routing, each message can cross a single link in unit time. The store-and-forward model is a standard model which has been widely used to study routing and other problems in parallel computers. In wormhole routing, messages are sent as *worms*, each of which consists of a sequence of fixed size units called *flits*. A flit is defined as the amount of data that can be sent along a link in one time step. The *length* of a worm is the number of flits it contains. The first flit is called the *head* and the remaining flits are called the *body* of the worm. During the routing a worm occupies a contiguous sequence of edges along its path, one flit per edge. This means that worms are not allowed to be seperated into different parts. In addition to this we assume that no processor in the network is able to store parts of the worms during the algorithm. This has the consequence that only the links can hold the flits.

Wormhole routing is an extremly popular strategy for data movement in parallel computers and is used in a variety of machines including the Intel Paragon, CRAY T3D, MIT J-Machine and Stanford DASH. It has several advantages over store-and-forward routing. In store-and-forward routing, if a b-flit message traverses a path of length d, and is never delayed, then it will reach its destination in bd steps (assuming that each channel can transmit one flit in each step). In a wormhole router, however, the first flit does not wait for the rest of the message. It therefore arrives at its destination after d steps, and the last flit of the message arrives after $d+b-1$ steps. The difference in time is due to a better utilization of network links by the wormhole router. In addition to reduced latency, wormhole routing also has the advantage that it can be implemented with small, fast switches, and is a realistic model for optical communication.

Given a function $f : \mathcal{P} \longrightarrow \mathcal{P}$, our (wormhole) routing problem is to design a protocol that routes n^d worms, one from each processor v to its destination $f(v)$. If f is chosen at random we talk about *routing a random function*; if

[*] email:{stbo,fmadh,chrsch}@uni-paderborn.de, Fax: +49-5251-603511/606482. Supported in part by DFG-Sonderforschungsbereich 376 “Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen”, by DFG Leibniz Grant Me872/6-1 and by EU ESPRIT Long Term Research Project 20244 (ALCOM-IT).

f is a permutation we talk about *permutation routing*. Our distributed protocol is *online*, i.e., every processor decides about the next routing step only based on local information. We present the first protocol for wormhole routing on the (n, d)-torus that is optimal also for large, non-constant dimension d.

1.1. The Lower Bound

We can formulate the following lower bound for routing randomly chosen functions:

Remark 1.2 *For any routing protocol, the expected time to route a random function with worms of length L in the (n, d)-torus is in $\Omega((L + d)n)$.*

Proof: At first we ask for the expected distance between the source and the destination for any worm in the network. Consider an arbitrary node $a = (a_d, a_{d-1}, ..., a_1)$. The destination $f(a)$ of the worm starting from a can have a distance of 0, 1, 2, .. , $n/2 - 1$ from a with probability at least $1/n$ in every dimension. This leads to the expected distance $\Omega(n)$ in every dimension. Since we have d dimensions, we get an expected total distance of $\Omega(dn)$.

Furthermore, we can analyse the work that the network has to perform. Since, for a randomly chosen function, there are n^d worms of length L that want to travel an expected distance of $\Omega(dn)$, the expected number of movements of flits along edges to be executed is at least $\Omega(n^d L \cdot dn)$. On the other hand, the network has at most $n^d d$ links, so that there are $\Omega(Ln)$ parallel steps needed to do the work.

These two lower bounds yield the claimed expected routing time. □

1.2. Previous work

Bar-Noy, Raghavan, Schieber and Tamaki [1] present an algorithm for the $(n, 2)$-torus which achieves a runtime of $O(kLn)$ with probability at least $1 - n^{-k}$ for any $k > 0$.
Cypher, Meyer auf der Heide, Scheideler and Vöcking [2] present an algorithm that reaches the running time $O(\frac{Lnd^{1/B}+(nd+L)d\log n}{B})$ for routing n^d worms of length L, one from each processor, in a (n, d)-mesh with bandwidth B. This result is optimal for $L \geq d^2 \log n$ and $B \geq \log d$, but leads to high time loss if B is equal to 1, which is the case in this paper.
Fridetzky [3] shows that it is possible to route a random function f w.h.p. (w.h.p. means "with high probability", i.e., with probability at least $1 - n^{-c}$ for some $c > 0$) in time $O((L + d)n \log n)$ in the (n, d)-mesh.

1.3. New Results

We present the first algorithm for wormhole routing on (n, d)-tori with runtime $O((L + d)n)$. As shown in Remark 1.2 this is optimal. We achieve this result for every d and worms of length L as long as $L = O(\frac{n}{(d\log n)^2})$. We only handle the case of sending worms to random destinations. By using Valiants trick [8] (first send the messages to random intermediate destinations and from there to the final destinations) it can be extended to reach the same time bound also for arbitrary permutations. As a subroutine of our algorithm we develop a new algorithm for routing random functions on the $(n, 2)$-torus. It has the same (optimal) expected runtime as the one in [1], but is much more reliable for moderate sizes of L. It runs in time $O(Ln+(kL\log n)^2)$ with probability at least $1 - n^{-k}$, for any $k > 0$.

2. Description of the new algorithm

In this section we describe our algorithm for the (n, d)-torus. We assume that $L \leq \frac{n}{\beta(d\log n)^2}$ for an arbitrary constant $\beta > 0$.

2.1. The structure of the algorithm

Our algorithm routes every worm to its destination by seperating this problem into d new subproblems. For every dimension we want to adjust the actual position of every worm to the position of its destination within the dimension. In particular, we first want to adjust the actual position of the worm to the position of its destination with regard to the first dimension before we do this for the second dimension and so on. To handle these subproblems we use a new 2-dimensional algorithm that is used in parallel in the (n,2)-subtori along dimension i and $i + 1$ to adjust the position of the worms in this part of the network to the dimension i of their destinations. To enable this, our algorithm works in $6abL$ batches for suitably chosen constants a and b. Every batch consists of $\frac{n^d}{6abL}$ suitably chosen worms such that at most $n/6abL$ worms start in every *cycle*. (In the (n, d)-torus we call a set of n processors with the same values in every dimension except in a particular dimension i a *cycle along dimension i)*. The whole algorithm is seperated into *stages* of length kn for some fixed constant k defined later.

At the first stage, our new 2-dimensional algorithm is used to adjust the positions of the first batch of worms to the positions of their destinations in the first dimension within kn steps, w.h.p., using only links in the first two dimensions. After this the worms of the first batch are in cycles along dimension 2, and their values in the first dimension are now equal to the first dimension of their destinations. In the next stage the same happens for dimensions 2 and 3. So we can assume that after further kn steps every worm of this batch has correct values w.r.t. dimensions 1 and 2 and is layed out in an arbitrary cycle along dimension 3. While the 2-dimensional algorithm starts with the worms of batch 1 in dimensions 3 and 4, the second batch begins with dimensions 1 and 2. Parallel to the algorithm in dimensions 3 and 4, the positions of the worms participating in the second batch are adjusted to the positions of their destinations w.r.t dimension 1 by our 2-dimensional algorithm, using only links in the dimensions 1 and 2. This leads to the basic structure of the algorithm, illustrated in Figure 1.

Step:	Dimension:										
	1	2	3	4	5	6	7	8	9	10	....
0	batch 1	batch 1									
$2kn$	batch 2	batch 2	batch 1	batch 1							
$4kn$	batch 3	batch 3	batch 2	batch 2	batch 1	batch 1					
⋮											
$(12abL+d-3)kn$	All batches have reached their destination										

Figure 1. The mainframe of the algorithm. (a and b are suitably chosen constants)

Note that if every 2-dimensional algorithm can cope with its task in time kn, the worms of different batches never hinder each other. Thus we can conclude that the whole algorithm finishes after $O((d+L)n)$ steps w.h.p.

2.2. The algorithm for (n,2)-subtori

As mentioned above, we use a *new* 2-dimensional algorithm in every stage. Although there are optimal protocols for the wormhole routing problem in the $(n,2)$-torus in the literature, we can not use any of them to cope with our problem. The reason for this is that all known algorithms for the 2-dimensional torus do not guarantee runtime bounds with sufficiently high probability. As our algorithm for the (n,d)-torus uses such a protocol more than n^{d-2} times as a subroutine for the $(n,2)$-subtori, the known algorithms (see Section 1.2) would fail in performing within a constant factor of their expected runtime for several of the applications if d is non-constant. This would destroy our algorithm for the (n,d)-torus for non-constant d. So we design a new algorithm that consists of two phases:

2.2.1 The first phase

The first phase (cf. Figure 3) starts with arranging the worms in *sliding diagonals* along cycles of dimension i. Such a *sliding diagonal* – the concept was first introduced in [1] – is an arbitrary diagonal of a $(n,2)$-subtorus formed by the heads of worms, that moves in direction of a particular dimension. (By "*in direction*" we mean that the value w.r.t. this dimension will be increased in every step. We will use "*against the dimension*" for the opposite direction.) The sliding diagonal technique guarantees that only worms on the same diagonal compete for turning into the next dimension, and if two worms collide, they only collide with their heads. In case of a collision, we will always prefer the worm that has already turned into the new direction, which means that once the head of a worm is successful in turning into the new dimension, this worm can not be hindered anymore.

The analysis will show that there are w.h.p. at most $\frac{n}{abL}$ worms in every cycle along dimension i. So it is possible to distribute them in such a way among n/bL diagonals – which run with distance bL in direction of dimension i – that there are at most n/a worms on every diagonal and at least a empty places on a diagonal between any two worms. This distribution is illustrated in Figure 2.

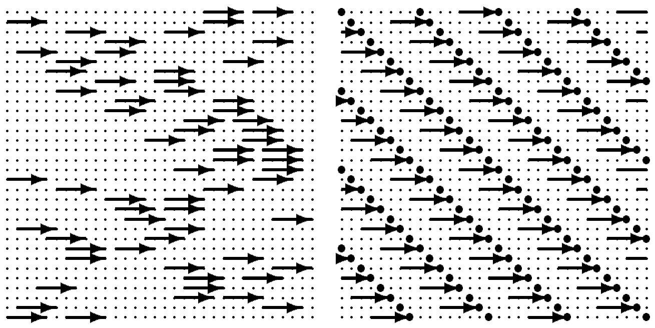

Figure 2. The distribution at the beginning of the first phase. In this example the parameters are $n = 32$ and $a,\ b = 2$.

After the distribution – this takes at most $L+n$ steps – the first phase begins. In this phase we seperate the 2-dimensional torus into blocks of $\alpha d \log n$ columns which we call *coarse destinations*. The coarse destination of a worm is that block of columns its real destination column belongs to. Every worm is supposed to turn only into a column of its coarse destination. To achieve this the sliding diagonals move in direction of dimension i, and every worm on these diagonals that reaches its coarse destination attempts to turn in direction of dimension $i+1$ again and again until it can turn or all $\alpha \log nd$ trials fail.

Note that, as mentioned above, a worm can only be hindered by the *head* of a worm which is placed on the *same* diagonal and is already moving along this column. But if it is able to use the link in direction of dimension $i+1$ it runs in direction of dimension $i+1$ for the rest of this first phase. If all $\alpha d \log n$ trials fail, the worm runs further in direction of dimension i, which means that our protocol fails. The following analysis will show that this is very improbable. But if it happens, and there is a worm that has not turned after those $n+L$ steps, it will achieve this in the next $n+L$ steps. There the same procedure takes place,

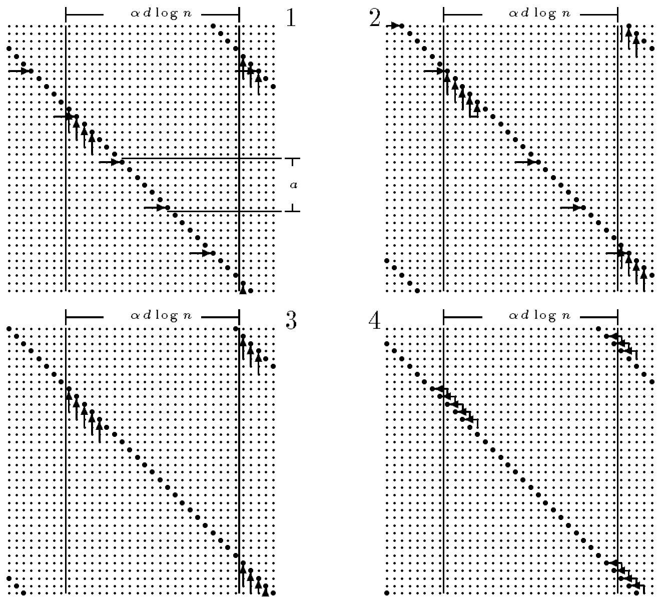

Figure 3. The run of the first phase: One diagonal reaches a column block (1), where a worm turns into the next free column (2). After this the worms that have reached a column of their column block run to the top (3) until the first phase ends, which happens after a fixed number of steps. Then they turn to the left (4) to be subsequently reordered for the second phase.

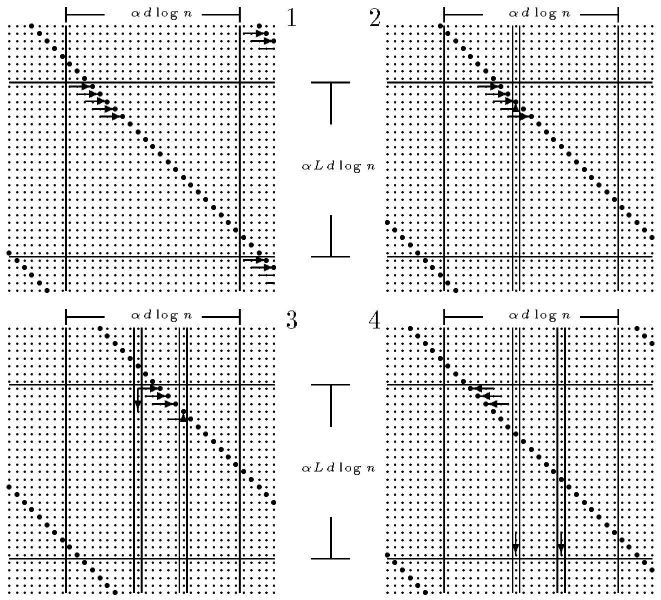

Figure 4. The run of the second phase: The worms run at the beginning of each phase to the right (1). One of them turns into its destination column (2). In this column it runs to the top and turns around at the border of its block to run downwards (3). While it has reached its destination for this phase the other worms on the diagonals come back after $2L \log n$ steps (4).

but now every unsuccessful worm will attempt to turn into an *arbitrary* column. This is certainly successful because of the fact that there are much more columns than worms that are placed on the same diagonal. So after all, every worm runs in direction of dimension $i + 1$ with its diagonal.

Before we can start with the *second phase* of the 2-dimensional algorithm we have to reorder the worms again. For this all worms first turn *against* the direction of dimension i and run in this direction $L + \alpha d \log n$ steps. After this, they turn again *in* direction of dimension i and run in this direction L steps. This means that every worm – we assume the first phase was successful for every worm – stands with the head in direction of dimension i exactly one block in front of its coarse destination. Therefore it is at most $2\alpha d \log n - 1$ and at least one step in front of its real destination column. This reordering is illustrated among other things in Figure 5 (2).

2.2.2 The second phase

In this phase (cf. Figure 4) every worm should turn into its destination column. Therefore we now partition the 2-dimensional subtorus into *blocks* of $\alpha d L \log n$ rows. We define the links *against* the direction of dimension $i + 1$ as a *reservoir* for worms, that is, a place where worms that already reached their correct column can wait until the second phase is over.

To achieve this the second phase consists of $\alpha d \log n$ *rounds*. Every round itself has $4L\alpha d \log n + 2L$ steps. In the first $2L\alpha d \log n$ steps every diagonal runs in direction of dimension i and every worm that is placed on it tries to turn into its destination column. If this is possible it runs in direction of dimension $i + 1$ until it reaches the top of the row block. There it turns to travel in the opposite direction. It continues to travel in this way until it reaches the bottom of this row block or the next link is occupied by an already waiting worm. The worm waits in this position until the end of the second phase and so until the 2-dimensional algorithm is over. Note that this worm has corrected the value of its ith dimension, which is illustrated in Figure 5 (4). On the other hand, if it is not able to turn it continues to travel in direction of the ith dimension. Then after the first $2L\alpha d \log n$ steps the diagonals turn around and travel $2L\alpha d \log n + L$ steps in the opposite direction. During this part no worm attempts to turn into its column. After this the

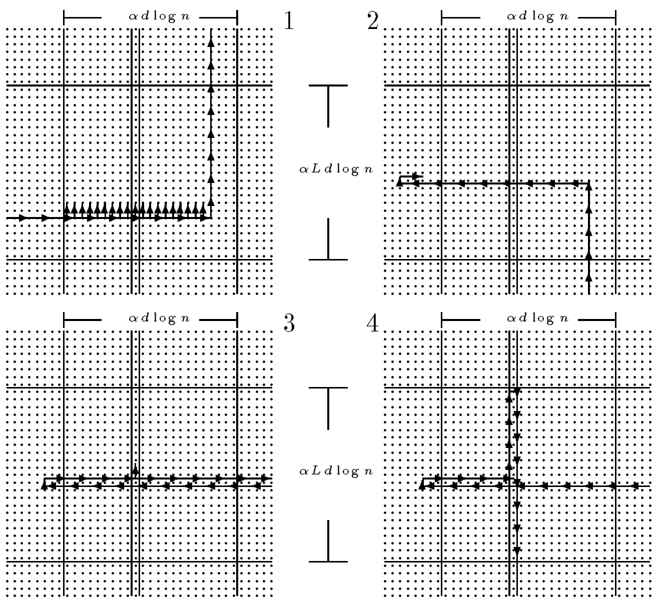

Figure 5. A typical way a worm can take in the algorithm. After several failed attempts it reaches the coarse destination (1). There it runs upwards until the first phase ends. Then it changes the direction (2) to begin the second phase where it fails to turn into the correct destination column (3), which is reached at the end (4).

worms turn again in direction of the dimension i for further L steps. Note that they have reached now the old position at the beginning of the round. At this moment the old round ends and a new begins.

We want to emphasize that during the first half of every round no worm can be blocked by a worm of a different diagonal. This can only happen if a reservoir spills over and therefore no more worm can reach the links against the direction of dimension $i+1$. But the analysis will show that this is very improbable.

3. Analysis of the algorithm

We will prove the following bound for the efficiency of our protocol:

Theorem 3.1 *For $L = O(\frac{n}{(d\log n)^2})$ the protocol described above routes worms of length L to random destinations in the (n,d)-torus within time $O((L+d)n)$, w.h.p.*

Note that this runtime bound is optimal (see Remark 1.2). Further note that this result can be extended to routing arbitrary permutations by using Valiant's trick (see Section 1.3). To prove Theorem 3.1 we have to show that with high probability *every* application of the 2-dimensional algorithm copes with its task in time $O(n)$. For this, we start with upper bounding the expected number of worms that are in the same cycle of the network after a stage. We have to show that this number is bounded by n/abL, because we want to work with at most n/bL diagonals per $(n,2)$-subtorus with at most n/a worms each. This is done in following lemma.

Lemma 3.2 *If $L \le \frac{n}{\beta(d\log n)^2}$ for some constant $\beta > 0$ then, for every constant $\ell > 0$, there exists with probability at least $1 - \frac{1}{n^\ell}$ no cycle along any dimension on which more than $\frac{n}{abL}$ worms of a batch want to be placed after an arbitrary stage.*

Proof: For dimension $i = 1$ there is nothing to show because we only allow $n/6abL$ worms to start in every cycle along dimension 1. So we can assume in the following that i is greater than 1. Let us consider an arbitrary fixed cycle along the ith dimension. Every node in this cycle has the same value for every dimension except the ith. This means that a worm that is able to reach this cycle during the routing has to start in a processor with the same values w.r.t. dimensions $i+1$ to d, because the values of these dimensions are kept unchanged during the first i stages of the algorithm. Hence the maximum number of worms that can reach this cycle is n^i. Since the first $i-1$ dimensions have to become equal to the values of the fixed cycle in order to reach it and the destinations are chosen uniformly at random, the probability for any of these worms for reaching this cycle is $1/n^{i-1}$. This leads to the following estimation:

Prob(There are at least t worms of a *fixed* batch in a *fixed* cycle along dimension i)

$$\le \binom{\frac{n^i}{6abL}}{t}\left(\frac{1}{n^{i-1}}\right)^t \le \left(\frac{e}{6}\right)^{\frac{n}{abL}}$$

$$\le \left(\frac{1}{2}\right)^{\frac{\beta d^2(\log n)^2}{ab}} \le \left(\frac{1}{2}\right)^{\zeta d \log n} = \left(\frac{1}{n^{\zeta d}}\right)$$

for any constant $\zeta > 0$ depending on n if $t = \frac{n}{abL}$ and $L \le \frac{n}{\beta(d\log n)^2}$ for some constant $\beta > 0$.

Since we have $6abL$ batches and there are $n^{d-1}d$ cycles along an arbitrary dimension, we get:

Prob(There exists a cycle on which at least $\frac{n}{abL}$ worms of an arbitrary batch want to be placed during the routing)

$$\le \frac{1}{n^{\zeta d}} n^{d-1} d 6abL \le \frac{1}{n^{\zeta d}} n^d \le \frac{1}{n^\ell}$$

for any constant $\ell > 0$ depending on ζ. □

With this lemma we can now assume that, for every stage and dimension, n/bL diagonals can be formed per $(n,2)$-torus with at most n/a worms each. Additionally we know that there are at least a empty places between two worms of every diagonal. In the following lemma we show that in this case it is very improbable that a worm cannot turn into its coarse destination during the first phase of the algorithm.

Lemma 3.3 *For every constant $\ell > 0$ there is a constant $\alpha > 0$ such that with probability at least $1 - \frac{1}{n^\ell}$ there exists no column block of size $\alpha d \log n$ in which a worm of an arbitrary batch is not able to turn into its coarse destination during the first phase.*

Proof: We mentioned above that only the head of a worm that is placed on the same diagonal and is already moving along a column of its coarse destination can block a worm from turning. This leads to the following observation: If a worm on a particular diagonal in a fixed batch is not able to turn into its coarse destination, there are at least $\alpha d \log n$ worms on the same diagonal that want to turn into the same block of columns. The probability of this event can be estimated by:

Prob(During a *fixed* application of the 2-dimensional algorithm, there are at least t worms on some *fixed* diagonal that want to turn into a particular block of columns)

$$\leq \binom{\frac{n}{a}}{t}\left(\frac{\alpha d \log n}{n}\right)^t \leq \left(\frac{1}{2}\right)^{\alpha d \log n} = \frac{1}{n^{\alpha d}}$$

for $t = \alpha d \log n$.

Again we can estimate the number of these cases. There are $6abL$ batches of worms, each applying the 2-dimensional algorithm $n^{d-2}(d-1)$ times with at most $\frac{n}{bL}$ diagonals within each application. Furthermore, each of these diagonals crosses $\frac{n}{\alpha d \log n}$ blocks of columns. This leads to the total number of $6abL\frac{n}{bL}n^{d-2}(d-1)\frac{n}{\alpha d \log n}$ experiments. So we can estimate:

Prob(There exists a diagonal in an arbitrary stage where at least $\alpha d \log n$ worms want to turn into the same block of columns)

$$\leq \frac{1}{n^{\alpha d}} n^{d-2}(d-1)6abL\frac{n}{bL}\frac{n}{\alpha d \log n} \leq \frac{1}{n^\ell}$$

for any constant $\ell > 0$ depending on α. □

Because of the lemma above we can now assume that every worm has reached its coarse destination after the first phase of the algorithm. So according to the description of the algorithm we know that, before the second phase starts, every worm stands exactly one block of columns in front of its coarse destination. This means that its value w.r.t. dimension i is at least one and at most $2\alpha d \log n - 1$ lower than the value of its destination in this dimension. So we can conclude – if we assume, that no reservoir spills over – that after $\alpha d \log n$ rounds of the second phase every worm can turn into its destination column. This holds, because if no reservoir spills over then there are at most $\alpha d \log n$ worms on every diagonal that want to move to the same reservoir. So we still have to show that with high probability no reservoir spills over.

Lemma 3.4 *Let us assume that the first phase is always successful. Then for every constant $\ell > 0$ there exists a constant $\alpha > 0$ such that there is with probability at least $1 - \frac{1}{n^\ell}$ no reservoir of size $\alpha d L \log n$ into which more than $\alpha d \log n$ worms want to turn during the second phase.*

Proof: Because of the assumption that the first phase has been successful we can conclude that each worm stands exactly one block in front of its coarse destination. So if we ask for the number of worms that want to enter a particular reservoir we can only find candidates in the left neighbouring block of processors consisting of $\alpha d L \log n$ rows and $\alpha d \log n$ columns. In addition to this the diagonals on which the candidates are placed still have a distance of bL, so that we can estimate:

Prob(During the second phase of a fixed application of the 2-dimensional algorithm, there are at least t worms that want to turn into a *particular* reservoir of size $\alpha d \log n$)

$$\leq \binom{(\alpha d \log n)(\alpha d L \log n \frac{1}{bL})}{t}\left(\frac{1}{\alpha d \log n}\right)^t$$

$$\leq \left(\frac{e}{b}\right)^{\alpha d \log n} \leq \frac{1}{n^{\alpha d}}$$

for $t = \alpha d \log n$ and b sufficiently large.

Again we have to count the number of such cases. For every of the $6abL$ batches there are $n^{d-2}(d-1)$ applications of the 2-dimensional algorithm to consider. In addition to this we have to multiply this number with $\frac{n^2}{dL\alpha \log n}$ reservoirs where an overfill can happen. This leads us to the following estimation:

Prob(During the second phase of an arbitrary application of the 2-dimensional algorithm, there exists a reservoir which at least $\alpha d \log n$ worms attempt to enter)

$$\leq \quad \frac{1}{n^{\alpha d}} 6abL \frac{n^2}{\alpha dL \log n} n^{d-2}\,(d-1) \leq \frac{1}{n^{\alpha d}} n^d \leq \frac{1}{n^\ell}$$

for any constant $\ell > 0$ depending on α. □

With the lemmas above we have shown that with high probability every worm of an arbitrary batch can turn into its column during the $\alpha d \log n$ rounds of the second phase. Hence the total number of steps required by every stage w.h.p. is bounded by:

$$\underbrace{n+L}_{\text{starting}} + \underbrace{2n+2L}_{\text{phase 1}} + \underbrace{d\alpha \log n + 2L}_{\text{turning\&correcting}} +$$
$$\underbrace{\alpha d \log n\,(4L\alpha d \log n + 2L)}_{\text{phase 2}} = O(n)$$

if $L \leq \frac{n}{\beta(d \log n)^2}$.

It remains to count the number of stages in the whole algorithm. As mentioned above, at most $12abL + d - 1$ stages are necessary if all applications of the 2-dimensional algorithm are successful. Hence the running time of the whole algorithm is bounded by:

$$\underbrace{(12abL + d - 1)}_{\text{stages}} \cdot \underbrace{(kn)}_{\text{2d-algorithm}} +$$
$$\underbrace{n+L}_{\text{routing to destination}} = O\left((d+L)\,n\right)$$

This proves Theorem 3.1 □.

It is easy to modify our 2-dimensional subroutine for routing random functions in the $(n, 2)$-torus. For this we use abL stages to route n/abL fixed worms to their destination. In every such stage we use our 2-dimensional subroutine above but with new seperations of our network: In the d-dimensional algorithm the subroutines worked on an $(n, 2)$-subtorus seperated into row blocks of size $\alpha dL \log n$ and column blocks of size $\alpha d \log n$. Now we use the same algorithm with row blocks of size $\delta L \log n$ and column blocks of size $\delta \log n$. After correcting the value w.r.t. the first dimension we route the worms within further $n + L$ steps to their destination. It is easy to verify that by exchanging $\alpha d \log n$ with $\delta \log n$ in the proof of Theorem 3.1 the same runtime bound is achieved for a stage with probability $1 - n^{-\Theta(\delta)}$. So within abL stages we can route all n^2 worms to their destination and therefore get the following runtime bound for our algorithm in the $(n, 2)$-torus:

$$\underbrace{abL}_{\text{stages}}\,(\underbrace{n+L}_{\text{starting}} + \underbrace{2n+2L}_{\text{phase 1}} + \underbrace{\delta \log n + 2L}_{\text{turning\&correcting}} +$$
$$\underbrace{\delta \log n\,(4L\delta \log n + 2L)}_{\text{phase 2}} + \underbrace{n+L}_{\text{routing to destination}})$$

$$= O(abL(n + L(\delta \log n)^2)) = O(Ln + (\delta L \log n)^2).$$

This leads to the following observation:

Observation 3.5 *Consider wormhole routing of a random function with worms of length L in the $(n, 2)$-torus. Then the runtime of the 2-dimensional algorithm is bounded by $O(Ln + (\delta L \log n)^2)$ with probability $1 - n^{-\Theta(\delta)}$ for any $\delta > 0$.*

Note that – if L is of moderate size – this new 2-dimensional algorithm guarantees the optimal runtime $O(Ln)$ with much higher probability than the already known algorithms (see Section 1.2). The reason for this is that Ln asymptotically dominates $(\delta L \log n)^2$ for $L \leq \frac{n}{(\delta \log n)^2}$ and so δ does not affect the runtime as much as in the previous algorithms.

References

[1] A. Bar-Noy, P. Raghavan, B. Schieber, H. Tamaki: Fast Deflection Routing for Packets and Worms. In *Proc. of the 12th ACM Symp. on Principles on Distributed Computing*, pp. 75-86, 1993.

[2] R. Cypher, F. Meyer auf der Heide, C. Scheideler, B. Vöcking: Universal Algorithms for Store-and-Forward and Wormhole Routing. In *Proc. of the 28th ACM Symp. on Theory of Computing*, pp. 356-365, 1996.

[3] T. Friedetzky: Wormhole Routing auf mehrdimensionalen Gittern. *Diploma Thesis at University of Paderborn (Research group F. Meyer auf der Heide)*, 1995.

[4] S. Felperin, P. Raghavan, E. Upfal: A Theory of Wormhole Routing in Parallel Computers. In *Proc. of the 33rd IEEE Symp. on Foundations of Computer Science*, 1992.

[5] T. Hagerup, C. Rueb: A guided tour of Chernoff bounds. In *Information Processing Letter 33*, pp. 305-308, 1989/90.

[6] F.T. Leighton: *Intodruction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[7] F.T. Leighton, B.M. Maggs, S.B. Rao: Universal packet routing algorithms (extended abstract). In *Proc. of the 29th Annual Symp. on Foundations of Computer Science* pp. 256-271, 1988.

[8] L.G. Valiant: A scheme for fast parallel communication. In *SIAM Journal on Computing 11/2* pp. 550-561, 1982 .