Angewandte Mathematik und Informatik
Universität zu Köln

Report No. 96.218

Optimal Oblivious Permutation Routing in Small Hypercubes

by

Thomas Seifert[1], Ewald Speckenmeyer

1996

Universität zu Köln
Institut für Informatik / Lehrstuhl III
Pohligstr. 1
D-50969 Köln

[1]This research was supported by the Deutsche Forschungsgemeinschaft under reference number Sp 317/2-1.

1991 Mathematics Subject Classification: 94C99
Keywords: Routing, Oblivious Routing, Hypercubes, Permutations

Optimal Oblivious Permutation Routing in Small Hypercubes

Thomas Seifert[†], Ewald Speckenmeyer
Universität zu Köln
Institut für Informatik / Lehrstuhl III
Pohligstr. 1, D-50969 Köln
e-mail: {seifert,esp}@informatik.uni-koeln.de

24th January 1996

Abstract

For each $d \leq 8$ we provide an oblivious algorithm for routing any permutation on the d-dimensional hypercube in at most d communication steps. To prove our result we show that any 1-to-$2^{d'}$-routing problem and any $2^{d'}$-to-1-routing problem can be solved in at most d' ($d' \leq 4$) coummunication steps on a d'-dimensional hypercube. Furthermore we present a class of efficiently working routing algorithms which allows us to make an improved statement about the complexity of some of the provided algorithms.

1 Introduction

Many different routing problems on the most popular networks have been examined in the past. In this paper we consider the d-dimensional binary hypercube and the problem of routing permutations with an oblivious routing algorithm. It is well known that any deterministic oblivious permutation routing algorithm working on a n-node degree-d network, will need at least $\sqrt{n}/(2d)$ parallel communication steps in the worst case [KKT90]. Applying this result to d-dimensional hypercubes we can conclude that there is no oblivious algorithm for routing permutations using at most d communication steps, if $d \geq 19$. The best known oblivious routing algorithm for a d-dimensional hypercube with $n = 2^d$ vertices works in $\frac{2\sqrt{n}}{\log n - 2\log\log n + 2} + \frac{\log n}{2}$ communication steps [KKT90]. Table 1, column 4 shows the values of this function for $1 \leq d \leq 20$. For each d the cited algorithm requires more than d communication steps. This is not optimal. In this paper we provide optimal algorithms for binary hypercubes with up to 256 processors (i.e. $d \leq 8$).
Anyway, this result is interesting since there is no online algorithm for the d-dimensional hypercube, deterministically routing any permutation in time $O(d)$. It is however

possible to route a fixed permutation in $O(d)$ parallel communication steps, i.e., there is an offline algorithm for efficiently routing permutations. In addition, there are routing algorithms based on fast sorting algorithms [CP90], which run in time $O(d \log d)$. Also there are algorithms which work with high probability im time $O(d)$ [LMRR94].
This paper is organized as follows: In section 2 we provide some basics and definitions. Section 3 describes a class of efficiently working routing algorithms. Immediately, we get an improved statement about the complexity of the well known greedy algorithm on 2-dimensional grids. In section 4 we provide the routing algorithm for hypercubes and prove its complexity. Finally we summarize our results in section 5 and list several open problems.

$d = \log n$	n	$\approx \frac{\sqrt{n}}{2 \cdot \log_2(n)}$	$\approx \frac{2\sqrt{n}}{\log n - 2\log\log n + 2} + \frac{\log n}{2}$
1	2	0,707107	1,4428
2	4	0,5	3,0
3	8	0,471405	4,5911
4	16	0,5	6,0
5	32	0,565685	7,3017
6	64	0,666667	8,6536
7	128	0,808122	10,184
8	256	1,0	12,0
9	512	1,25708	14,211
10	1024	1,6	16,9489
11	2048	2,05704	20,3837
12	4096	2,66667	24,7406
13	8192	3,48114	30,3211
14	16384	4,57143	37,5297
15	32768	6,03398	46,9111
16	65536	8,0	59,2
17	131072	10,6482	75,3889
18	262144	14,2222	96,8205
19	524288	19,0547	125,314
20	1048576	25,6	163,338

Table 1: column 1: dimension d; column 2: number of processors n; column 3: lower bound for oblivious routing algorithms on hypercubes; column 4: number of communication steps used by the best known oblivious routing algorithm on hypercubes.

2 Preliminaries

A *network* is a simple, undirected graph $G = (V, E)$. Its vertices are also called *processors*. A *packet routing problem* can be described by a finite number of data packets, each to be sent from a source processor to a target processor. To achieve this a packet is only allowed to use the edges of the network, which are also called links. A network is assumed to work in discrete time steps and in each time step each link can be used to transport at most one packet. This is also called simultaneous full-duplex routing. We exclusively consider *static* routing problems, i.e. all packets to be routed are present in the network when the routing commences and cannot be generated dynamically. Packets are considered to be atomic with a constant size which is independent from the number of processors. A routing problem is said to be a k-to-l routing problem if each vertex is the source of at most k packets and each vertex is the destination of at most l packets. A *routing algorithm* is a strategy to solve a class of routing problems on a fixed network. A routing algorithm is called to be *oblivious* if the path to be travelled by a packet depends on the origin and destination of the packet only. A routing algorithm works *online* if at each time step the link to be used by a packet can be locally determined within the packet's processor. The *cross product* of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a graph $G = (V, E)$ with $V = \{(v_1, v_2) \mid v_1 \in V_1, v_2 \in V_2\}$ and $E = \{((u, x), (v, x)) \mid (u, v) \in E_1, x \in V_2\} \cup \{((x, u), (x, v)) \mid (u, v) \in E_2, x \in V_1\}$. [Lei92, section 3.1.2] A *d-dimensional hypercube* is a network $HC_d = (V, E)$ with $n = 2^d$ processors, where a communication link between two processors v and w exists if and only if the binary representations of v and w differ in exactly one bitposition. If q and z are two processors of a d-dimensional hypercube, then $dist(q, z) := num1bits(q \oplus_{\text{bin}} z)$ is called the distance between q and z, i.e. the number of bitpositions where the binary representations of q and z are different. A *1-dimensional grid* (linear array) is a network with a set of processors $V = \{1, \ldots, n\}$, where processor $i \in \{2, \ldots, n-1\}$ is linked together with processors $i-1$ and $i+1$ and no other edges exist. A *2-dimensional grid* (rectangular array) is a cross product of 2 linear arrays.

3 An efficient solvable class of routingproblems

Let $G = (V, E)$ be a network and R a routing algorithm, which solves a class of static routing problems on G. Furthermore let $P = ((q_1, z_1), \ldots, (q_m, z_m))$ be a routing problem from this class, i.e. $(q_i, z_i) \in V^2$ is a pair of source-/destination processors of the i-th packet. The procedure how R solves the routing problem P can be described by

- a set of pathes $W_1, \ldots, W_m$, where $W_i = (q_i = v_{i1}, v_{i2}, \ldots, z_i = v_{il_i})$ is the path which the i-th packet has to travel, and
- a contention-resolution protocol, which determines which packet is sent first when there are many packets competing with each other for using a single link.

Let p be a packet, $t \geq 0$ a time step and $v \in V$ a processor. Then let $\varrho(p, t)$ be the length of the path which is used by p in accordance with R beginning with the vertex where p resides at time step t. $\varrho(p, t)$ is also called *remaining path length* of p at time step t. If p has not arrived at its destination in time step t, let $\eta(p, t)$ denote the directed edge which is to be used by p in time step t. Furthermore let $\mathcal{P}(v, t)$ be a set of packets on processor v at time step t with a destination different from v. Now let $p, q \in \mathcal{P}(v, t)$. A *link contention* at time step t between q and p is given if $\eta(p, t) = \eta(q, t)$. If there is a *link contention* between q and p, the remaining path lengths of the two packets are different ($\varrho(p, t) \neq \varrho(q, t)$) and if the packet with the greater remaining path length is sent first, then this contention-resolution protocol is called *farthest-first-strategy.* We call the farthest-first-strategy *applicable* if $\eta(p, t) = \eta(q, t)$ and $\varrho(p, t) \neq \varrho(q, t)$.

In what follows we consider routing algorithms which always permit to apply the farthest-first-strategy, i.e. routing algorithms with the following property:

$$\forall v \in V : \forall t \geq 0 : p, q \in \mathcal{P}(v,t) \Rightarrow \varrho(p,t) \neq \varrho(q,t) \text{ oder } \eta(p,t) \neq \eta(q,t). \tag{1}$$

I.e., in each time step there is no link contention between packets on a processor or the remaining path lengths are different.
Apart from this we assume that all considered routing algorithms effectivly apply the farthest-first-strategy, if it is applicable.
The main result of this section is the following theorem:

Theorem 3.1 *Let R be a routing algorithm. We assume that the farthest-first-strategy is always applicable. Let P be a routing problem which is solved by R. Let m be the maximal total path length which appears when P is solved with R. Then P is solved by R in at most m time steps.*

Proof: By applying property (1) we observe that in the first communication step all packets with remaining path length m can move toward their target processors. After this only packets with remaining path length less or equal $m-1$ exist. Inductively, it follows that with completing the i-th communication step only packets with remaining path length less or equal $m-i$ exist, for all $i \in \{1, \ldots, m\}$. Therefore, with completing the m-th communication step all packets have reached their target. $\square$

In what follows we present a simple application of this theorem: It is well known that each 1-to-1-routing problem on the 2-dimensional grid can be solved online with the greedy algorithm in at most $n+m-2$ communication steps [Lei92, section 1.7.1]. The greedy algorithm works as follows: Each packet is sent within its source processor's row using the shortest path to the destination processor's column. Within this column each packet is sent using the shortest path to its destination processor.
Using theorem 3.1 we can tighten the preceding statement about the complexity of the greedy algorithm as follows.

Theorem 3.2 *Let P be a 1-to-1-packet routing problem on the 2-dimensional grid solved by the greedy algorithm. Let l be the maximum of the lengths of all shortest pathes the packets have to be routed in accordance with P. Then P is solved in exactly l time steps.*

Proof: We only have to test the preconditions to be able to apply theorem 3.1. Using the greedy algorithm each packet can be transported within the row of its source processor without any link contention. Since we have a 1-to-1-routing problem, the remaining path lengths of the contentious packets are different. Therefore the farthest-first-startegy is always applicable and the claim is proven. $\square$

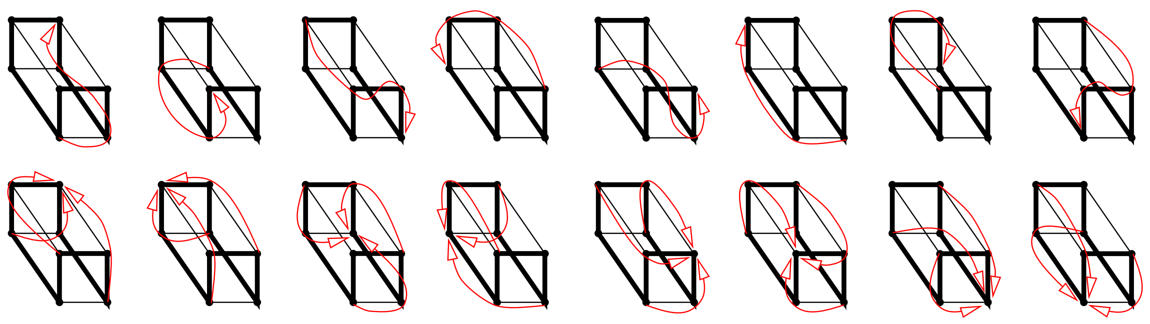

Figure 1: For all pairs of source and destination processors (q, z) the path $P_{q,z} \in \mathcal{P}_1$ mit $dist(q, z) \geq 2$ is shown.

4 Routing algorithms on the hypercube

In what follows we present a simple algorithm to optimally solve 1-to-1-routing problems on the d-dimensional hypercube if $d \leq 8$. The routing of permutations is a special case of 1-to-1-routing.
The algorithm works as follows: the d-dimensional hypercube is considered as cross product of two hypercubes with dimensions $d_1 := \left\lceil \frac{d}{2} \right\rceil$ and $d_2 := \left\lfloor \frac{d}{2} \right\rfloor$, respectively. We will prove that on the d'-dimensional hypercube every 1-to-$2^{d'}$-routing problem and every $2^{d'}$-to-1-routing problem can be solved in at most d' communication steps. If $d' \leq 3$ the presented algorithms use at most $m \leq d'$ time steps, where m denotes the maximum path length of all packets.
First of all we define two path systems $\mathcal{P}_1$ and $\mathcal{P}_2$, each consisting of pathes $P_{q,z}$ for each ordered pair of source and destination processors (q, z) in the 3-dimensional hypercube HC_3. The pathes $P_{q,z}$ in $\mathcal{P}_1$ are defined as follows: If $dist(q, z) = 0$, then is $q = z$ and $P_{q,z}$ is defined to be the path of length 0. If $dist(q, z) = 1$, then $P_{q,z}$ is defined as the unique path of length 1 from q to z in HC_3. For all pairs (q, z) with $dist(q, z) \in \{2, 3\}$ the corresponding pathes are shown in figure 1. The path system $\mathcal{P}_2$ is defined to include the pathes which are reverse to the pathes in $\mathcal{P}_1$, i.e. $\mathcal{P}_2 := \{(v_1, v_2, \ldots, v_k) \mid (v_k, \ldots, v_2, v_1) \text{ is path in } \mathcal{P}_1\}$. Now we are able to formulate and prove the following lemma:

Lemma 4.1 *Let $d \leq 3$. There is an oblivious routing algorithm which solves any 1-to-2^d-routing problem on the d-dimensional hypercube in at most d time steps.*

Proof: First of all we assume $d = 3$. The oblivious routing algorithm we use is based on the path system $\mathcal{P}_1$. In what follows, we will show that the farthest-first-strategy is always applicable. Then the assertion for $d = 3$ follows by applying theorem 3.1 and using the fact that the longest path has length 3. Now let p_1 and p_2 be any two packets with source processors q_1 and q_2 and destination processors z_1 and z_2, respectively. We make the following observations:

- Each packet with total path length 1 arrives at its target in the first communication step without any link contention, since each processor is the source of at most one packet. I.e., a packet with total path length 1 can never content with any other packet.

- If both p_1 and p_2 have total path length 3, then p_1 and p_2 can never have a link contention, since any two (directed) edges belonging to two different pathes of total length 3 are different from each other. This can easily be verified by looking up at figure 1 (upper row).
- If both p_1 and p_2 have total path length 2, then there cannot be any link contention concerning these packets. This can be seen as follows: Both p_1 and p_2 use the first link without contention. If there would be a contention between p_1 and p_2 when using the second edge of their pathes, then p_1 and p_2 would have the same target. But as can be seen in figure 1 (lower row), two packets having total path length 2, the same target and different sources, are always moved along different links towards their targets.
- Let us assume that p_1 has total path length 2 and p_2 has total path length 3. If p_1 and p_2 have a link contention, then they have also different remaining path lengths. This can be seen as follows: Both p_1 and p_2 use the first edge of their pathes without link contention. Therefore, if there is a link contention, then with regard to the second edge of the path of p_1. If there occurs no contention in the second communication step, then there will be never a link contention between these two packets since p_1 arrives within the second communication step at its destination processor. But in this case p_1 and p_2 have different remaining path lengths since both packets have moved exactly one edge. On top of that p_1 can only content with at most one packet with total path length 3, since for each (directed) edge there is at most one path of total length 3 containing this edge.

Altogether we have shown that either there is no link contention between p_1 and p_2 or their remaining path lengths are different.
We now consider the case $d \in \{0, 1, 2\}$. Each 1-to-2^d-routing problem on the d-dimensional hypercube can be extented to a 1-to-8-routing problem on a 3-dimensional hypercube, so that the greatest total path length is d. The routing algorithm we use is obtained by restriction of the routing algorithm on the 3-dimensional hypercube to the considered hypercube. In accordance with the explanations in the first part of this proof the farthest-first-strategy is always applicable, so that by using theorem 3.1 the claim also follows for $d \in \{0, 1, 2\}$. □

Lemma 4.2 *Let $d \leq 3$. There is an oblivious routing algorithm which solves any 2^d-to-1-routing problem on the d-dimensional hypercube in at most d time steps.*

Proof: This proof is similar to the proof of lemma 4.1. Again we show the claim for the case $d = 3$. The used oblivious routing algorithm is based on the pathes in path system $\mathcal{P}_2$. In what follows, we will show that the farthest-first-strategy is always applicable. Then the assertion for $d = 3$ follows by applying theorem 3.1 and using the fact that the longest path has length 3.
Now let p_1 and p_2 be any two packets with source processors q_1 and q_2 and destination processors z_1 and z_2, respectively. We make the following observations:

- If there is a packet with total path length 1 which does not reach its destination in the first or a following communication step, then it contents with a packet with total path length greater than 1. Otherwise there would be a processor which is the destination of at least two packets. Since the algorithm works on a 8-to-1-routing problem this is not possible.

- If both p_1 and p_2 have total path length 3, then p_1 and p_2 can never have a link contention, since any two (directed) edges belonging to two different pathes of total length 3 are different from each other. This can easily be verified by looking at figure 1 (upper row). The pathes of p_1 and p_2 are different since otherwise they would have the same destination.
- If both p_1 and p_2 have total path length 2, then we consider two cases: **Case 1:** p_1 and p_2 have the same source processor. Then p_1 and p_2 have different destinations since we are working on a 8-to-1-routing problem. Figure 1 (lower row) shows that two pathes both having total path length 2, the same source, and different destinations, are edge disjoint. Therefore there cannot be a link congestion. **Case 2:** p_1 and p_2 have different source processors. Then the first edge of their pathes are different, since the sources are different. The same holds for the second edges of their pathes, since p_1 and p_2 have different destinations. Therefore, if there is a link contention between p_1 and p_2, then with regard to an edge which is the first edge of the path of p_1 and the second edge of the path of p_2, or the other way round. But this means that the remaining path lengths are different.
- Let us assume that p_1 has total path length 2 and p_2 has total path length 3. Again we consider two cases: **Case 1:** p_1 and p_2 have the same source processor. If the pathes of p_1 and p_2 have a common edge, then the remaining path lengths are different. **Case 2:** p_1 and p_2 have different source processors. Then the first edges of their pathes are different, since the sources are different. Let e be the second edge of the path of p_1. Then e cannot be the third edge of the path of p_2, since the packets would have the same destinations. Therefore the remaining path lengths are different, if there is a link contention with regard to edge e.

Altogether we have shown that either there is no link contention between p_1 and p_2 or their remaining path lengths are different.
We now consider the case $d \in \{0, 1, 2\}$. As in the proof of lemma 4.1 we observe: Each 2^d-to-1-routing problem on the d-dimensional hypercube can be extented to a 8-to-1-routing problem on a 3-dimensional hypercube, so that the greatest total path length is d. The routing algorithm we used is obtained by restriction of the routing algorithm on the 3-dimensional hypercube to the considered hypercube. In accordance with the explanations in the first part of this proof the farthest-first-strategy is always applicable, so that by using theorem 3.1 the claim also follows for $d \in \{0, 1, 2\}$. $\square$

In what follows we define the path systems $\mathcal{P}_3$ and $\mathcal{P}_4$ in the 4-dimensional hypercube. We consider figure 2, the left hand graph. It is easy to verify that the graph represents a 4-dimensional hypercube. Let us denote the red drawn circuits with C and $\bar{C}$. Furthermore we define $\mathcal{A}$ to be the orientation *counterclockwise* and $\bar{\mathcal{A}}$ to be the orientation *clockwise*. Let $a, b \in C$ and $\mathcal{O} \in \{\mathcal{A}, \bar{\mathcal{A}}\}$. With $dist_{\mathcal{O}}(a, b)$ we denote the minimal number of edges on C, which must be passed when travelling from a using orientation $\mathcal{O}$ to b.
For each ordered pair of processors (q, z) let $\mathcal{P}_3$ contain a directed path $P_{q,z}$ defined as follows:

- If both q and z are on C and $dist_{\mathcal{A}}(q, z) \leq 4$, then $P_{q,z}$ starts at q and runs along C using orientation $\mathcal{A}$ to q.

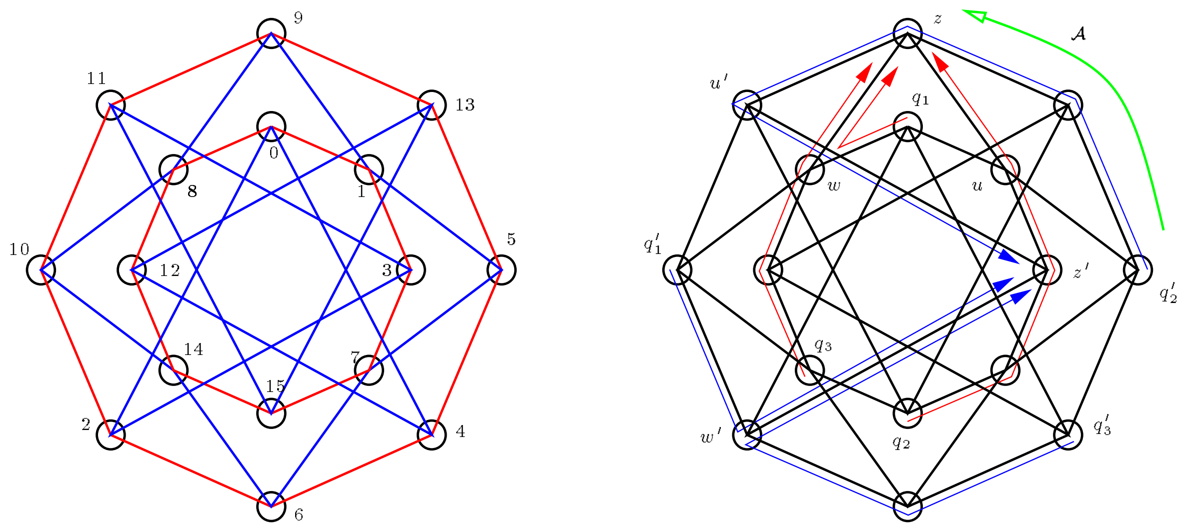

Figure 2: Representation of a 4-dimensional hypercube.

- If both q and z are on C and $dist_{\mathcal{A}}(q,z) > 4$, then $P_{q,z}$ starts at q and runs along C using orientation $\bar{\mathcal{A}}$ to q.
- If $q \in C$ and $z \in \bar{C}$, let $u, w \in C$ be the unique vertices incident with z and $dist_{\mathcal{A}}(u,w) = 2$. If $dist_{\mathcal{A}}(q,u) \leq 3$, then $P_{q,z}$ starts at q and runs along C using orientation $\mathcal{A}$ to u and then uses link (u,z). Edge (u,z) is called a type-1-link. If $dist_{\mathcal{A}}(q,u) = 7$, then $P_{q,z}$ starts at q and runs along C using orientation $\mathcal{A}$ to w and then uses link (w,z). If $dist_{\mathcal{A}}(q,u) \in \{4,5,6\}$, then $P_{q,z}$ starts at q and runs along C using orientation $\bar{\mathcal{A}}$ to w and then uses link (w,z). Edge (w,z) is called a type-2-link.

In figure 2, the right hand graph, the pathes $P_{q_i,z}$ (red) and $P_{q'_i,z'}$ (blue) are shown for $i \in \{1,2,3\}$.
Let path system $\mathcal{P}_4$ contain all reversed pathes of the pathes in $\mathcal{P}_3$.

Lemma 4.3 *There is an oblivious routing algorithm which solves any 1-to-16-routing problem on the 4-dimensional hypercube in at most 4 time steps.*

Proof: A packet with source processor q and target processor z is moved along the path $P_{q,z} \in \mathcal{P}_3$. First of all we observe that any packet is able to move along a circuit without any link contention, since each processor is source of at most one packet. Every blue edge can only occur as last edge of a path $P_{q,z}$ and only with respect to a blue edge a link contention may occur. Now let $z \in \bar{C}$ a destination and $u, w \in C$ the unique vertices incident with z such that $dist_{\mathcal{A}}(u,w) = 2$. The packets moving along link (u,z) to z use the edge (u,z) within the communication steps 1–4 without any contention. With respect to link (w,z) there may be a link contention between the packet with source q' ($dist_{\mathcal{A}}(q',w) = 1$) and at most three other packets moving along (w,z) to z. We define the packet with source q' to use the edge (w,z) in the fourth communication step, while the other three packets use link (w,z) in the communication steps 1–3. Since each path in $\mathcal{P}_3$ has maximum path length 4 it follows with the previous explanations that each packet arrives within 4 communication steps at its destination. □

The reader may recognize that this routing algorithm works with any contention-resolution protocol as far as a packet is moved as soon as possible.

Lemma 4.4 *There is an oblivious routing algorithm which solves any 16-to-1-routing problem on the 4-dimensional hypercube in at most 4 time steps.*

Proof: We use the routing algorithm defined by the path system $\mathcal{P}_4$. As far as possible arising link contentions are resolved using the farthest first strategy. If other link contentions occur, i.e. if packets with identical remaining path length are contenting with each other, an arbitrarily choosen packet among all involved packets is moved first.
First of all we realize that each packet which has to change the circuit in order to reach its destination, either is moved exactly once across a type-1-edge or a type-2-edge. A packet with total path length (tpl) $5 - i$ (which has to move across a type-1-edge) at the latest moves across a type-1-edge in the i-th communication step (cs). A packet with tpl 3 is moved in the first cs, a packet with tpl 2 is at the latest moved in the third cs and a packet with tpl 1 is at the latest moved in the fourth cs across a type-2-edge. Anyway, the following statement $B(i)$ holds for each $i \in \{1,2,3,4\}$: After the i-th cs all packets with remaining path length $4 - i$ have reached the circuit of its destination. Furthermore we observe that each packet has a remaining path length (rpl) of at most 3 after the first cs.
We consider the following chain of implications for $i \in \{1,2,3\}$: if after the i-th cs only packets with rpl less or equal $4 - i$ are left and if after the i-th cs all packets with rpl $4 - i$ have reached the circuit of its destination processor ($B(i)$), then any two packets with rpl $4 - i$ cannot content with each other, since we have a 16-to-1-routing problem. Therefore all packets with rpl $4 - i$ can move in the $(i+1)$-th cs.
Since each packet with rpl 4 can move in the first cs (on each node there are at most two packets with tpl 4, which surely have to move across two different links), there are only packets with a rpl less or equal 3 left. Using this chain of implications we inductively conclude that each packet arrives at its destination after 4 cs. $\square$

Theorem 4.1 *Let $d \leq 8$. There is an oblivious routing algorithm which solves any 1-to-1-routing problem on the d-dimensional hypercube in at most d time steps.*

Proof: If $d \leq 3$ the assertion is proved by lemma 4.1, since any 1-to-1-routing problem is a special case of a 1-to-2^d-routing problem.
If $d \in \{4,5,6,7,8\}$ the given hypercube is considered as cross product of a d_1-dimensional hypercube h_1 and a d_2-dimensional hypercube h_2 with $d_1 := \left\lceil \frac{d}{2} \right\rceil$ and $d_2 := \left\lfloor \frac{d}{2} \right\rfloor$. The routing problem on the d-dimensional hypercube is solved by first solving 1-to-2^{d_1}-routing problems in 2^{d_2} copies of h_1 and then solving 2^{d_2}-to-1-routing problems in 2^{d_1} copies of h_2. A packet with source processor $(q_d, \ldots, q_1)$ is first sent to vertex $(z_d, \ldots, z_{d-d_1+1}, q_{d-d_1}, \ldots, q_1)$ (problems A) and afterwards to processor $(z_d, \ldots, z_1)$ (problems B).
Since each processor is the source of at most one packet, it is guaranteed that each problem A is a 1-to-2^{d_1}-routing problem which can be solved in at most d_1 communication steps by applying lemma 4.1 and lemma 4.3, respectivley. Accordingly, since each processor is the destination of at most one packet, it is guaranteed that each problem B

is a 2^{d_1}-to-1-routing problem which can be solved in at most d_2 communication steps by applying lemma 4.2 and lemma 4.4, respectivley. Since all partial problems are solved using a oblivious routing algorithm, the whole algorithm works oblivious, too. Finally, it is obvious that any packet uses at most d communication steps to reach its destination. $\square$

5 Final remarks

We have shown that any 1-to-1-routing problem on the d-dimensional hypercube can be solved using at most d communication steps with an oblivious routing algorithm, if $d \leq 8$. It is not known, whether this is also possible for $d \in \{9, \ldots, 18\}$. We surely know that it is impossible, if $d \geq 19$ because of the cited lower bound.
Our approach cannot be extented to higher dimensional hypercubes ($d \geq 9$). In the case $d = 9$ we would have to solve a 1-to-16-routing problem (or a 16-to-1-routing problem) on a 5-dimensional hypercube in at most 5 time steps. However, this is not possible, since there are 16 vertices with distance greater or equal 3 from a given processor v. Consider the routing problem where 16 packets starting on these processors have to be moved to v. These packages would have to use the edges incident with v in the communication steps 3, 4 and 5. However, at most 15 packets can be moved within 3 time steps on v.
In the presented algorithms for 1-to-16-routing and 16-to-1-routing, there are packets with distance 2 (between source and destination) which move using pathes of length 4. It is an open problem if there is an oblivious routing algorithm, such that each packet with source q and destination z uses a path of length $dist(q, z)$. It is also an open problem whether such an algorithm can guarantee that each packet can reach its destination within m time steps, where m denotes the maximal total path length in the given problem.

References

[CP90] R. Cypher and C.G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (Baltimore, Maryland, May 14–16, 1990)*, pages 193–203, New York, 1990. ACM SIGACT, ACM Press.

[KKT90] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures SPAA '90 (Island of Crete, Greece, July 2-6, 1990)*, pages 31–36, New York, 1990. ACM SIGACT, ACM SIGARCH, ACM Press.

[Lei92] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: Arrays, trees, hypercubes.* Morgan Kaufmann Publishers Inc., San Mateo, CA, 1992.

[LMRR94] F. Thomson Leighton, Bruce M. Maggs, Abhiram G. Ranade, and Satish B. Rao. Randomized routing and sorting on fixed-connection networks. *J. Algorithms*, 17:157–205, 1994.