Hot-Potato Routing Algorithms for Sparse Optical Torus

Risto Honkanen[1], Ville Leppänen[2], and Martti Penttonen[1]

[1] Department of Computer Science and Applied Mathematics
University of Kuopio,
P.O.Box 1627
FIN-70211 Kuopio, FINLAND
E-mail: {honkanen,penttonen}@cs.uku.fi
[2] Department of Computer Science
University of Turku,
Lemminkäisenkatu 14–18, Datacity
FIN-20520 Turku, FINLAND
E-mail: Ville.Leppanen@cs.utu.fi

Abstract. In this work we present an optical network architecture and deflection (or hot potato) routing algorithms supporting efficient communication between n processor nodes. The network consists of an $n \times n$ torus, where processor nodes are situated diagonally, and routing nodes are optical deflection nodes of two inputs and two outputs. A design of optical deflection node is presented. Routing algorithms are variations of the greedy routing algorithm, and by experiments and partial theoretical analyses they run efficiently on this architecture.

1 Introduction

Routing algorithms have many applications in data communication. Our work is motivated by the design of an efficient parallel computer, and in particular, by the emulation of shared memory with distributed memory modules. In this application, we have a network connecting a number of processor nodes (consisting of a router, a processor, and a memory module), and router nodes. In each clock cycle, a router forwards all packets that arrived at it. A processor node acts like a router if a packet was not targeted to it, but it absorbs a packet targeted to it. In addition to the packets that it received to be routed forward, it also may send packets created by itself. However, own packets can be sent only when there are less than two incoming packets to be routed, because the number of incoming and outgoing packets is bounded by the properties of the network — in our case there are only two input lines and two output lines.

Our task is very demanding because every clock cycle a routing decision must be made in each routing node, and successive nodes independent of each others. Actually, indepence is not quite true. Depending on the routing algorithm, in the long run there may be some kind of accumulation of packets, which is usually harmful and difficult to manage.

By results in [18], if a parallel computation has enough parallel *slackness*, the implementation of shared memory can be reduced to efficient routing of an h-relation. By definition, in an *h-relation* each (processor) node has at most h packets to send, and it is the target of at most h packets. An implementation of an h-relation is *work-optimal* at *cost c*, if all packets arrive at their target in time ch. Obviously, if a processor has two output lines and two input lines, $c \geq 0.5$ at best, and $c = 1$ means that no time is lost for waiting even if a packet is send every time unit.

A precondition of work-optimal routing on n-processor architecture is that h is greater than the diameter ϕ of the network and the network can move $\Omega(n\phi)$ packets in each step, since otherwise slackness cannot be used to "hide" diameter influenced latency. Many kinds of architectural solutions satisfying the above have been derived; see e.g. [7, 11, 16, 18].

We favor *mesh*-based architectures like *torus*, since mesh has a simple, homogeneous structure. A mesh also scales up by adding similar routing elements. (However, the size of the network grows quadratically with respect to the number of processors.) On the other hand, work-optimality of h-relation routing clearly requires a high bandwidth, and *optics* has a huge potential in this respect.

h-relation routing also requires switching of packets at the rate of clock cycle. Optoelectrical conversions might be slow and energy consuming, and thus it would be desirable to keep the packets in optical form as long as possible. The lack of optical memory, on the other hand, leads to *hot-potato* (or *deflection*) routing as data cannot be stored in router nodes. The requirement that the network must be able to move at least $n\phi$ packets at every moment implies that there must be $\Omega(\phi)$ nodes for each processor, i.e. the network must be *sparse*. This is satisfied, if there are n processors in a $n \times n$ torus. For symmetry and equidistance, it is natural that the processors are situated *diagonally*. This reasoning leads us to the concept of sparse optical torus (Section 2) using only simple directed 2×2 optical nodes, and to the search of efficient hot-potato routing algorithms for it (Section 3).

2 Sparse optical torus

We study a 2-dimensional $n \times n$ torus structure of diameter $\Theta(n)$, while many other networks studied in the literature have diameter $\Theta(\log n)$. A small diameter is considered advantageous as the diameter determines a lower bound for the routing time. However, in three-dimensional world the number of nodes can grow only cubically with respect to the physical distance. Hence, the length of connections increases when the system is scaled up [19]. Some logarithmic networks (e.g. the hypercube) also suffer from the fact that the degree of nodes increases as the size of the network is scaled up. Meshes and tori do not suffer node degree and connection length problems (both are constant). With a clever layout [14] all connections of a torus can have a small constant length. We prefer the torus structure, because it is simple, has a constant degree, and connections have a constant length. In this section we present the structure of the sparse optical torus and discuss chances to construct one.

2.1 Structure of the network

A 2-dimensional *sparse optical torus*, $SOT(n)$, consists of p processors $(P_0, P_1, \ldots, P_{n-1})$ and $n(n-1)$ optical deflection nodes. The processors are on a diagonal so that processor P_i resides at location $(i, n-1-i)$ of the underlying torus (see Figure 1). There is a 2×2 optical node $R_{i,j}$ located at each position (i,j) such that $i + j \neq n-1$ and $0 \leq i,j \leq n-1$. The two outputs of deflection node $R_{i,j}$ are connected to the deflection nodes or processors at locations $(i+1 \bmod n, j)$ and $(i, j+1 \bmod n)$ (respectively the inputs to $R_{i,j}$ come from locations $(i-1 \bmod n, j)$ and $(i, j-1 \bmod n)$). Thus, all the connections are unidirectional. We assume that each deflection node and processor is capable of receiving and sending one message per link in one time unit.

Lemma 1. *The distance from processor P_i to processor P_j is n, for all $i \neq j$.*

Proof. A shortest path from location $(i, n-1-i)$ to $(j, n-1-j)$ involves $d_X = j - i \bmod n$ moves along the X-axis and $d_Y = n-1-j-(n-1-i) \bmod n$ moves along the Y-axis. Clearly, $0 < d_X, d_Y < n$, since $i \neq j$. Since $d_X + d_Y \equiv j - i + n - 1 - j - n + 1 + i \equiv 0 (\bmod n)$, it must be that $d_X + d_Y = n$. □

Lemma 1 basically means that in hot-potato routing, the set of packets A_τ sent by the processors at time τ can deflect only packets belonging to A_τ and for each set of packets A_τ, there are n moments of time for such deflections to take place.

2.2 Deflection routing

A routing strategy used to resolve output port contention problem in packet-switched interconnection networks is the *hot-potato* or *deflection* routing strategy. In the hot-potato routing all entering packets must leave at the next step – i.e. packets cannot be buffered like in the store-and-forward routing strategy. In general, in each node the out-degree must be at least the in-degree, and the output link contention must be resolved somehow. If there are multiple packets preferring the same output link, the routing strategy must select at most one for each out-going link.

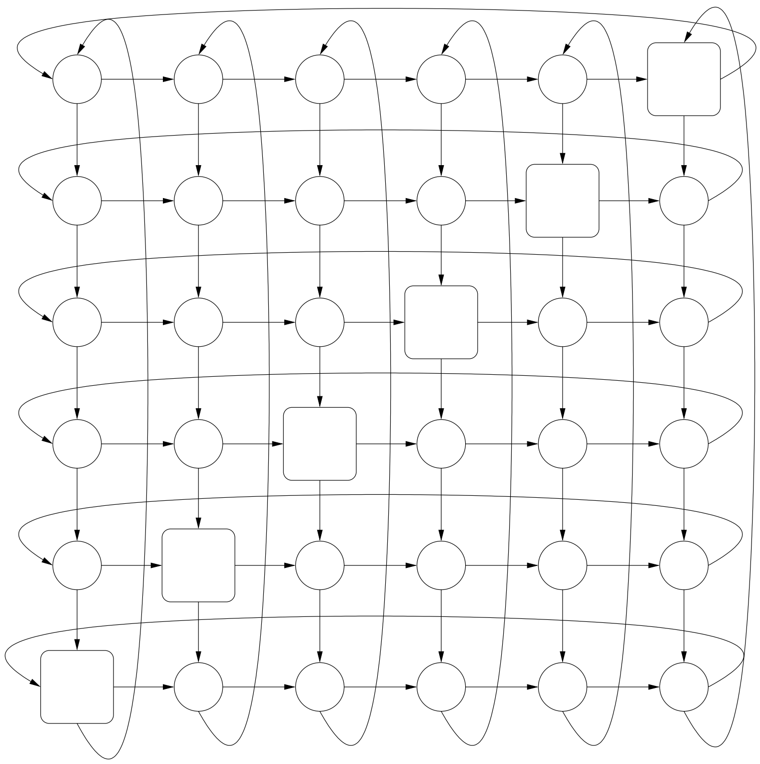

Fig. 1. A 6-processor 6×6 sparse optical torus.

An out-going link is *good* for a packet, if it takes the packet closer to its destination. A link is *bad* for a packet if it is not good for it. Actually, choosing a bad link means that the distance to the destination grows by $n-1$ instead of decreasing by 1. In this case it is also said that the packet is *deflected*. Notice also that several links might be good for a packet. See [15] for definitions and a survey of hot-potato routing techniques and results.

An advantage of the hot-potato routing is that the packets do not need buffering during the transmission and only a small fraction of packets needs an optoelectronic conversion during the transmission. Besides transmitting packets, a hot-potato routed network acts like a memory.

2.3 Implementation of an optical deflection routing node

The bandwidth of the system is divided in time slots, whose length T_p equals to the bypass time of a packet. We call the length of slot T_p the packet cycle. Continuous streams of pulses are transmitted over switches under the global bit clock. A packet consists of a header and data bits so that the overall length of the time slot measured in time units is T_p. A processor may insert a packet into a time slot, if there is an empty slot in a data stream passing the corresponding switch.

An illustration of an optical deflection node has been given by Figure 2 [5, 6]. An optical deflection node operates as follows: The payload of an incoming packet is divided by a passive coupler. The coupling ratio can be chosen to be e.g. 90/10 so that 10 % of the energy of packet is guided to O/E conversion and 90 % is guided to a delay line after amplification. The header is converted into electrical form and sent to the RCP (*Routing Control Processor*) while the other share remains in optical form in a delay line. The RCP makes the routing decision and sets the switches of the routing node into the correct state. If the circuit connected to the node desires to send a packet, it activates the request signal of the routing node. When the node detect that there is an empty incoming slot, it gives a permission to send for the circuit.

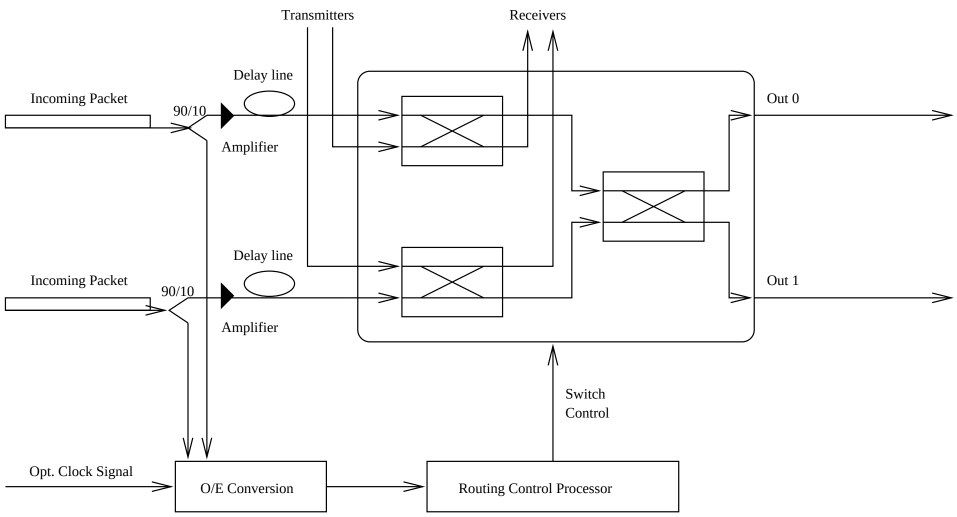

Fig. 2. A schema of an optical deflection node[5].

The basic components of routing nodes, namely, passive splitters, passive combiners, and electrically controlled switches, can be implemented by $LiNbO_3$ technology [9]. The switching time of $LiNbO_3$ switches lies in the range 10 − 15 ps [2].

The length of packet in an optical fiber can be evaluated by equation $l_p = \frac{N_p \times v_c \times n}{B}$, where N_p is the size of the packet in bits, v_c is the speed of light in vacuum, n is the refraction index of fiber, and B is the link bandwidth. Note that the length of time slot T_p can be evaluated by equation $T_p = \frac{N_p}{B}$.

There are three main factors that limit the capacity of a fiber-optic delay-line memory [17]: fiber loss, chromatic dispersion, and synchronization errors. Assuming commercially available 10 Gbit/s link bandwidth and the size of packet 64 bytes ($N_p = 8 \times 64$), the length of packet (fiber-optic delay-line memory) is about 10 meters and the length of time slot is about 50 ns. According to Soukop et al., with a fiber loss of 0.2 dB/km, a maximum of 50 km of fiber could be used for the data delay line [17]. According to Saleh and Teich, it is possible to obtain up to 10 Gbit/s link bandwidth in the range up to $\sim$ 50 km utilizing single-mode fiber at the operating wavelength of 1.3 μm [13]. Temperature changes may cause synchronization errors. If the temperature dependant delay varies 35.5 ps/°Ckm [17], variations of ± 5°C result ± 7.1 ps of error in a 40 meters data delay line. Synchronization error can be, however, removed by synchronization blocks [4].

3 Routing

In our architecture the distance from processor to processor is n. Hence the latency of routing is $\Omega(n)$. We assume that the packets are targeted to random addresses. Therefore we cannot assume that a processor receives only a constant number of packets from any given processor. However, it is known that with high probability the number of packets is $O(\log n / \log\log n)$ [12], if each processor has $O(n)$ packets. For this reason a good practical goal would be an algorithm that routes $n \log n$ packets (from each of the n processors) in $O(n \log n)$ time.

3.1 Greedy routing algorithms

We will look closer at three different variations of the greedy principle. From the routing point of view, the contents of the packets are not interesting. Therefore we assume that packets are of the form (x, y), where $x, y \in \{0, \ldots, n-1\}$, $x + y = n - 1$. Each node (processor or router) has 0, 1 or 2 packets to route, coming from the left, up or from the processor itself, and to be routed right or down. The behaviour of the general routing algorithm depends on the function first,right:=protocol(i, j, S, x) that determines, which packet in S is privileged to choose direction and to which direction it goes. The last parameter x is the name of the protocol. The general for of the algorithm is

Algorithm 1 (Greedy routing algorithm)

> **proc** *Greedy*
> **for** $i = 0..n-1$ **pardo**
>   **for** $j = 0..n-1$ **pardo**
>   *Let* $S_{ij} = \{p_0, p_1\}$ *be the set of incoming packets at* n_{ij}
>   **if** n_{ij} *is processor (i.e.* $i + j = n - 1$*)* **then**
>   **for** $p \in S_{ij}$ **do**
>   **if** $p = (i, j)$ **then** *remove p from* $S_{i,j}$
>   **if** $|S_{i,j}| < 2$ *and* n_{ij} *has packets to route* **then**
>   *add* $2 - |S_{i,j}|$ *packets to* $S_{i,j}$
>   *first,right := protocol(*$i, j, S_{i,j}, x$*)*
>   **if** *right=1* **then**
>   *route first packet to right and second (if exists) to down*
>   **else**
>   *route first packet to down and second (if exists) to right*

We now consider some special cases of the general algorithm.

One-sided greedy protocol Greedy-a. In the first of our protocols, a packet is first sent right until it reaches its target column. Packets that have already reached the target column and are moving down are privileged with respect to the packets still moving to the right. Packets move downwards only if they have reached the target column. In particular, this rule prohibits a processor from sending a packet downwards. We have

> **proc** protocol(i, j, S, a)
> **if** some packet $p_k \in S$ $(k = 0, 1)$ came from up **then return** $k, 0$
> **elsif** some packet $p_k \in S$ $(k = 0, 1)$ is (m, j) for some m **then return** $k, 0$
> **else return** 0,1

Two-sided lateral greedy protocol Greedy-b. A possible weakness of protocol a is that it does not fully use the transportation capacity of the network. As new packets are sent only along the X-axis, only half of the network links are used. Protocol b sends packets to both arcs. However, the risk of deflection also increases. To keep the risk of deflection homogeneous, packets are forwarded randomly until they come to the edge, where they cannot continue in both directions without losing the possibility to reach the target in n steps. Packets with this property are privileged over the packets "in the middle" that can choose either route.

> **proc** protocol(i, j, S, b)
> Let k:=random(0,$|S| - 1$) and $p = (h, m)$ be the k'th element in S
> **if** $h = i$ **then return** $k, 1$
> **elsif** $m = j$ **then return** $k, 0$
> **else return** $k, random(0, 1)$

Diagonal greedy protocol Greedy-c. The third of our protocols tries to delay the moment, when there is difference in only one coordinate and the risk of being pushed to the next round.

> **proc** protocol(i, j, S, c)
> Let k:=random$(0,|S|-1)$ and $p = (h, m)$ be the k'th element in S
> **if** $(h - i) \bmod n < (m - j) \bmod n$ **then return** $k, 1$
> **else return** $k, 0$

3.2 Scheduled hot-potato routing algorithm

By Lemma 1 it is clear that one of the packets sent to processor P_i at time τ reaches P_i. This suggests that we could treat each routing round as the arbitrary OCPC [1] variation of the OCPC model (Optically Connected Parallel Computer, [3]). By dividing the packets of each processor to n subsets of size $h' = h/n$ and treating the routing of each subset as a separate routing problem on arbitrary OCPC, it would seem that we could extend the work-optical routing result ($h' = \Omega(\log n \log\log n)$ yields constant routing cost per packet) in [1] to our sparse optical torus. However, in [1] the 'arbitrary' means that a random packet of the conflicting packets succeeds whereas using the routing strategy of Greedy-a, there is no randomness in the determination of succeeding processors.

The basic problem in using the routing strategy of greedy-a to determine the routing paths of the packets is that given any fixed target, Greedy-a sets the senders to non-equal position. Among the packets sent to a fixed target, the one having shortest distance to its turning point always succeeds and others fail. However, we could advance this property by properly scheduling the right to sent to P_i so that only one processor P_j has the "right" to send to P_i at any given time. The idea in Algorithm 2 is this way to completely avoid deflections!

Algorithm 2 (Scheduled hot-potato routing algorithm)

> **proc** *Scheduled-hot-potato*
> **for** $i = 0..n-1$ **pardo**
>   *Processor P_i uses sequential n-way radix sort to semisort*
>   *its packets by their target to n linked list*
> *Let t be a global step counter (in each processor)*
> **for** $i = 0..n-1$ **pardo**
>   **for** $j = 0..n-1$ **pardo**
>   *Processor P_i: acts as in algorithm Greedy-a*
>   *except that it attempts to send packets only to $P_{(i+t) \bmod n}$*
>   *Deflection node $R_{i,j}$ $(i + j \neq n - 1)$: acts as in Greedy-a*

The idea in Algorithm 2 is that processor P_i sends packets to processor $P_{(i+x) \bmod n}$ in steps $x, n + x, 2n + x, 3n + x, \ldots$. Conflicts are avoided, since all processors send to different targets at any given time. Algorithm 2 is efficient only, if the distribution of packet (source, target)-pairs is roughly even.

3.3 Experimental results

We ran some simulations to get experience how good our different versions of greedy routing algorithms are. In simulations, processors send randomly addressed packets to each other, and we want to see how efficient the algorithms are initially with "fresh" packets (Figure 3), and what is the effect of increasing the slackness factor h (Figures 4 and 5).

We observe that Greedy-a and Greedy-b successfully route $(1-1/e)$ and respectively $2(1-1/e)$ packets per time unit, as will be theoretically confirmed in the next section. The diagonal algorithm Greedy-c decreases in efficiency, when n grows. Figure 4 shows that by increasing h, the efficiency of routing all packets approaches to the efficiency of routing fresh packets. Indeed, Figure 5 shows that $h = n \log n$ gives quite good efficiency results.

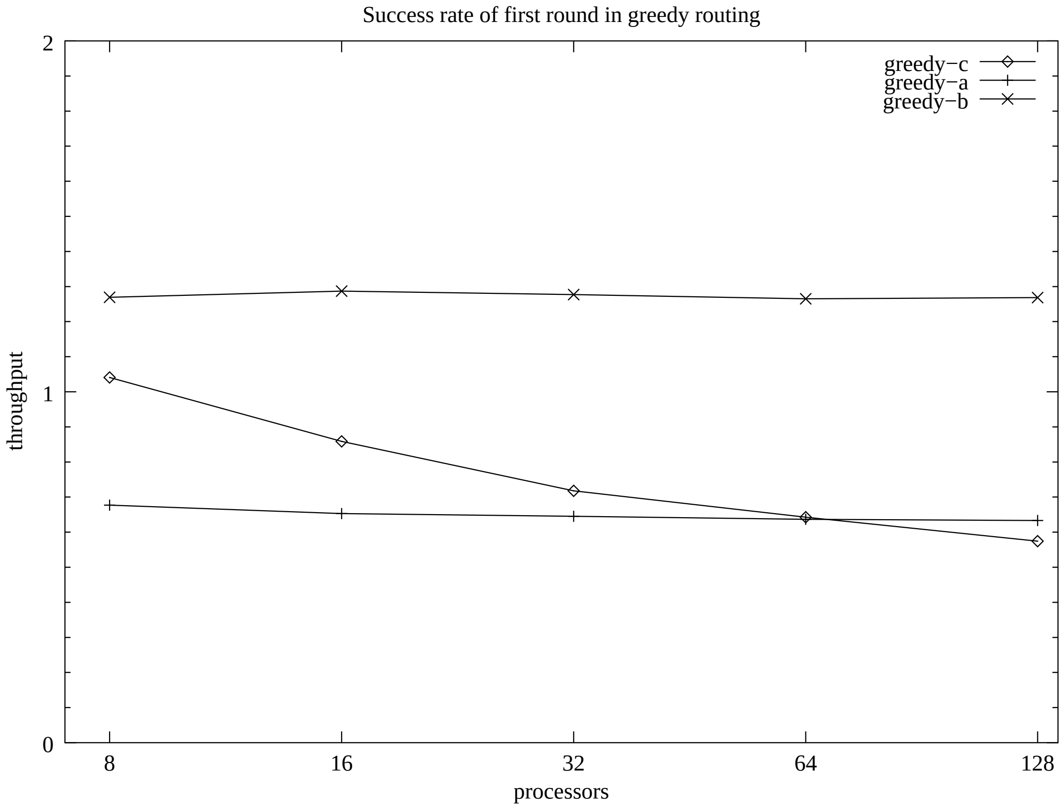

Fig. 3. Throughput of Greedy routing algorithms with fresh packets. Throughput is the number of successfully sent packets per processor in time unit.

4 Analysis

We would like to prove that with certain slackness value h Greedy-a and Greedy-b and Greedy-c guarantee that the routing cost is less than or equal to a certain value with high probability. Unfortunately such an analysis is very complicated and we can only partially analyse these protocols. Under some simplifying assumptions we can prove that when routing begins, packets reach their target with certain probability. By this probability we know, how many of the h packets reach successfully their target. After the first round the situation is no more "fresh" and analysing the success probability becomes harder and we cannot do it. However, even these partial analyses support the experimental results.

We first prove a result that may have some interest of its own.

4.1 Analysis of a recurrence equation

Let p be a constant such that $0 \leq p \leq 1$ and let $P(k,l)$ be a function defined by the recurrence equation

$$P(k,l) = \begin{cases} P(l,k) \text{ for all } k,l \\ \frac{1}{2}P(k-1,l) + \frac{1}{2}P(k,l-1) \text{ when } k = l > 0 \\ pP(k-1,l) + (1-p)P(k,l-1) \text{ when } k > l > 0 \\ pP(k-1,l) \text{ when } k > l = 0 \\ 1 \text{ when } k = l = 0 \end{cases}$$

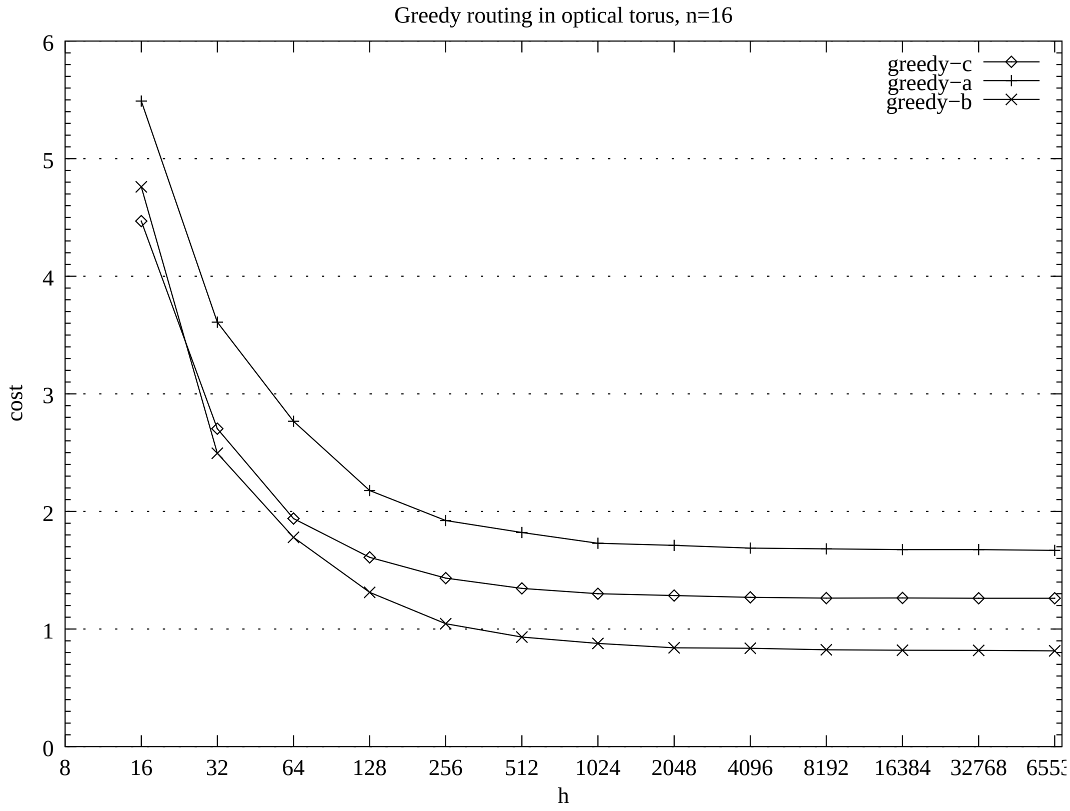

Fig. 4. The effect of slackness on the cost of greedy routing

Can we estimate the values of $P(k,l)$. If $p = \frac{1}{2}$, denote $Q(k) = P(k,k)$, and calculate $Q_0 = 1$, $Q_1 = \frac{3}{4}$, $Q_2 = \frac{45}{64}$, $Q_3 = \frac{702}{1024}$, ..., Q_k appears to approach $\frac{2}{3}$. Indeed, this is the case. We can prove generally

Theorem 3.
a. $\lim_{k\to\infty} P(k,k) = \frac{1-2(1-p)+\sqrt{1-4p(1-p)}}{2p}$
b. $\lim_{k\to\infty} \frac{1}{n}\sum_{i+j=k} P(i,j) = \frac{1-p}{p}(1-2(1-p)+\sqrt{1-4p(1-p)})$.

To prove *a*, denote $q = 1-p$ and $Q_k = P(k,k)$. When computing Q_k recursively, computation starts with a "diagonal" number Q_k and ends at the "diagonal" number Q_0, and there may be other "diagonal" numbers $Q_j, 0 < j < k$ in intermediate phases of the computation. Assume we already know all $Q_j, j < k$. We reduce the computation of Q_k to these as follows. Consider paths from (k,k) to $(0,0)$ in integer grid $\{(i,j)|0 \le i,j \le k\}$. Now

$$Q_k = 2\sum_{j=0}^{k-1} C_j \frac{1}{2} q^j p^{j+1} Q_{k-j-1}.$$

In this formula, 2 comes from the fact that on a path form (k,k) to $(0,0)$ we return first to the diagonal point $(k-j-1, k-j-1)$ either from the upper side of the diagonal or from the lower side of the diagonal. Each term in the sum corresponds to the first point on the diagonal $(k,k), (k-1, k-1), \ldots, (1,1), (0,0)$ where we first come after leaving (k,k). C_j is the number of paths from (k,k) to $(k-j-1, k-j-1)$ that do not contain any diagonal points $(k-1,k-1), \ldots, (k-j,k-j)$. It is easy to see that among the edges of each path leading from (k,k) to $(k-j-1, k-j-1)$, there is one that leaves (k,k) with probability $\frac{1}{2}$, there are j edges that diverge from the diagonal

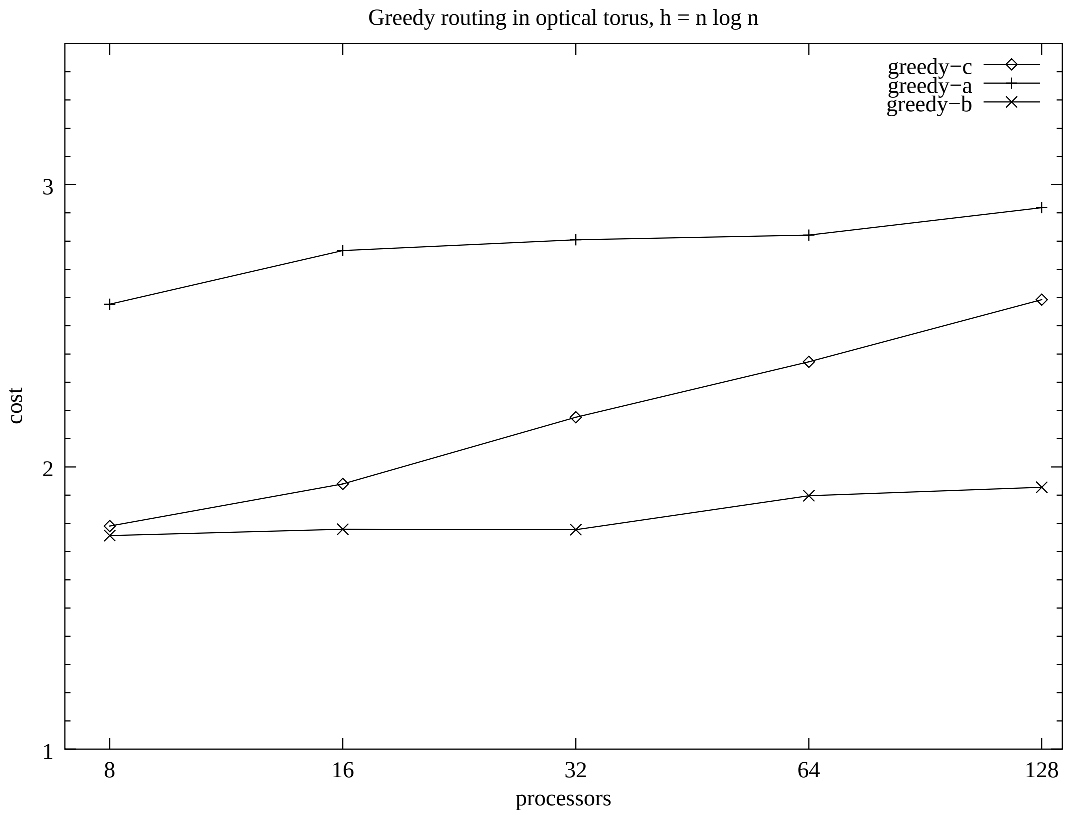

Fig. 5. The cost of greedy routing for $h = n \log n$

with probability q, and there are $j+1$ edges that converge towards the diagonal with probability p. Finally, the probability of reaching $(0,0)$ from $(k-j-1, k-j-1)$ is Q_{k-j-1}. The numbers C_j are known in literature as *Catalan* numbers, and we know

Lemma 2. *[10]*

$$C(z) = \sum_{i=0}^{\infty} C_i z^i = \frac{1-\sqrt{1-4z}}{2z}.$$

Analogously, denote $Q(z) = Q_0 + Q_1 z + Q_2 z^2 + \ldots$. By grouping factors in Q_k, we have

$$Q_k = p \sum_{j=0}^{k-1} C_j (pq)^j Q_{k-j-1} = p[C(pqz)Q(z)]_{k-1},$$

where $[F(z)G(z)]_k$ is the multiplier of z^k in the product of polynomials $F(z)$ and $G(z)$. Hence, we have

$$Q_0 = 1$$
$$[Q(z)]_k z^k = p[C(pqz)Q(z)]_{k-1} z^k, \text{ for } k > 0.$$

By adding columnwise we get

$$Q(z) = 1 + pC(pqz)Q(z)z = p\frac{1-\sqrt{1-4\times pqz}}{2\times pqz}Q(z)z = 1 + \frac{1-\sqrt{1-4pqz}}{2q}Q(z).$$

From this we can solve

$$Q(z) = \frac{2q}{2q-1+\sqrt{1-4pqz}} = \frac{1-2q+\sqrt{1-4pqz}}{2p(1-z)} = \frac{1}{1-z}R(z),$$

where

$$R(z) = \frac{1-2q+\sqrt{1-4pqz}}{2p}.$$

Observe now that $1/(1-z) = 1 + z + z^2 + \ldots$ and denote $R(z) = R_0 + R_1 z + R_2 z^2 + \ldots$ Then

$$Q_k = [(1+z+z^2+\ldots)(R_0 + R_1 z + R_2 z^2 + \ldots)]_k = R_0 + R_1 + \ldots + R_k.$$

Hence,

$$\lim_{k\to\infty} Q_k = R_0 + R_1 + \ldots = R(1) = \frac{1-2q+\sqrt{1-4pq}}{2p}.$$

To prove part b of the theorem, denote $S_n = \sum_{i+j=k} P(i,j)$. By recursion rule,

$$S_n = P(0,n) + P(1,n-1) + \ldots + P(\frac{n}{2}-1, \frac{n}{2}+1) +$$

$$P(\frac{n}{2}, \frac{n}{2}) +$$

$$P(\frac{n}{2}+1, \frac{n}{2}-1) + \ldots + P(n-1,1) + P(n,0) =$$

$$pP(0,n-1) + (1-p)P(0,n-1) + pP(1,n-2) + \ldots + (1-p)P(\frac{n}{2}-2, \frac{n}{2}+1) + pP(\frac{n}{2}-1, \frac{n}{2}) +$$

$$\frac{1}{2}P(\frac{n}{2}-1, \frac{n}{2}) + \frac{1}{2}P(\frac{n}{2}, \frac{n}{2}-1) +$$

$$pP(\frac{n}{2}, \frac{n}{2}-1) + (1-p)P(\frac{n}{2}+1, \frac{n}{2}-1) + \ldots + pP(n-2,1) + (1-p)P(n-1,0) + pP(n-1,0) =$$

$$S_{n-1} + (1-p)P(\frac{n}{2}-1, \frac{n}{2}) + (1-p)P(\frac{n}{2}, \frac{n}{2}-1) = S_{n-1} + 2(1-p)Q_{n/2}$$

for even n. The same argument applies to odd n. Hence,

$$S_n = 2(1-p)Q_0 + \ldots + 2(1-p)Q_{n/2}.$$

By $\lim Q_n = \frac{1-2q+\sqrt{1-4pq}}{2p}$ we get $\lim \frac{1}{n} S_n = \frac{1-p}{p}(1-2q+\sqrt{1-4pq})$. □

4.2 Analysis of greedy routing algorithms

Of the three greedy routing protocols, Greedy-a proves out to be the easiest to analyse. The random lateral protocol Greedy-b has some similarity with Greedy-a, while the diagonal protocol behaves differently.

Greedy-a. The route determined by Greedy-a is unique. If a packet avoids deflection when attempting to turn to the correct column, it will successfully reach its target. If many packets try to get to the same target, one of them is successful. Consider the n packets that are being sent to their random destinations at a given moment of time. How many of the n packets reach their destination? The probability that none of the n packets tries to arrive at a fixed processor is

$$(1-\frac{1}{n})^n \to \frac{1}{e}.$$

Hence, with probability $1-1/e$ there are packets coming to this destination and we know that exactly one of them is successful. As there are n destinations, the expected number of successful packets is $n(e-1)/e = 0.63 \times n$. Therefore, for large n and h, $e/(e-1) = 1.58$ should provide an approximation for the routing cost. This result matches with our experiments.

Greedy-b Even if Greedy-b is apparently quite different from Greedy-a, actually it behaves similarly.

In Greedy-b, the n processors send simultaneously $2n$ packets to $2n$ targets — remember that each processor has two receivers. First packets advance randomly until they reach the right row or column. Heuristically, we can identify randomly progressing packets with the horizontal packets of Greedy-a.

The probability that no packet is directed to a receiver is $(1-\frac{1}{2n})^{2n} \to \frac{1}{e}$ and hence each of the receivers has the probability $1-1/e$ of getting a packet. As there are $2n$ packets, the expected number of successful packets is $2(1-1/e)$, which is confirmed by the experiments.

Greedy-c In Greedy-c algorithm, let $P(k,l)$ denote the probability of a packet at distance (k,l) from the target to achieve the target. With probability $\frac{1}{2}$ it can choose the link in the node. In "fresh" routing situation, when target addresses are random, even in the other case, when the other packet chooses first, our packet has equal chances to decrease distance to its target. Hence, $P(k,l)$ is determined by

$$P(k,l) = \begin{cases} P(l,k) \text{ for all } k,l \\ \frac{1}{2}P(k-1,l)+\frac{1}{2}P(k,l-1) \text{ when } k=l>0 \\ \frac{3}{4}P(k-1,l)+\frac{1}{4}P(k,l-1) \text{ when } k>l>0 \\ \frac{3}{4}P(k-1,l) \text{ when } k>l=0 \\ 1 \text{ when } k=l=0 \end{cases}$$

By theorem 3b the average of P approaches to $1/3$, and hence about $2n/3$ should reach the target in the first round. However, experimental results do not satisfactorily match with this analysis, which must therefore be too simplistic.

4.3 Scheduled hot-potato routing

When applied to an h-relation, the Algorithm 2 first semisorts the packets in each processor by their target to n lists in $O(h)$ time by using sequential n-way radix sort. After that the routing time is determined by diameter n and $\max_{i,j} w(i,j)$, where $w(i,j)$ is the number of packets P_i has for P_j. It is well-known [12] that if $h=\Omega(n\log n)$, then $w(i,j)=O(\log n)$ with high probability. In other words, assuming that $h=\Omega(n\log n)$ there exists such constant c that after $n+ch$ routing steps all packets have reached their destination with high probability, and thus the routing cost per packet is upper bounded by another constant.

5 Conclusions

We presented an architecture, sparse optical torus, and algorithms for implementing a communication structure to be used in parallel computers. We believe that the simple, regular structure and all-optical communication are important benefits of the architecture. Also we believe that the presented routing algorithms based on greedy principle are useful and realistic. Software simulations support this claim. Unfortunately the theoretical analysis of the algorithms is still quite deficient.

A natural continuation of the work will be to extend the constructions to three dimensions. In three dimensional sparse torus, n^2 processors are set diagonally into an n^3 torus. The advantage of 3-dimensional sparse torus over 2-dimensional sparse torus is the decrease in the number of packets (or h in h-relation) that are needed for work-optimal routing. As for the scheduled Algorithm 2, we believe that also for the simpler Algorithms Greedy-a and Greedy-b, $h \in \Omega(n\log n)$ would be enough for work-optimal routing.

Often locality helps a lot in communication problems. It would be interesting to study the efficiency of a two-level structure, where the high level sparse optical toroid routes to an area of size $O(\log n)$ and in this area packets are routed locally (by a small PRAM, see [7]).

References

1. M. Adler, J. Byers, and R. Karp. Scheduling Parallel Communication: The h-Relation Problem. In *Proceedings of Symposium on Mathematical Foundations of Computer Science, MFCS'95, LNCS 969*, pages 1 – 20. Springer-Verlag, 1995.
2. S. Aleksić and K. Bengi. Multicast-capable access nodes for slotted photonic ring networks. *Proc. ECOC2000*, 3:83-84.
3. R.J. Anderson and G.L. Miller. Optical Communication for Pointer Based Algorithms. Technical Report CRI-88-14, Computer Science Department, University of Southern California, LA, 1988.
4. B. Bostica, M. Burzio, P. Gambini and L. Zuccheli. Syncronisation issues in optical packet switching networks. In G. Prati (ed). *Photonic Networks.* Springer Verlag, 1997.
5. T. Chich. *Optimisation du routage par déflexion dans les réseaux de télecommunication métropolitains.* PhD thesis, L'école Normale Supérieure de Lyon, France, 1997.
6. J.R. Feehrer and L.H. Ramfelt. Packet Syncronization for Syncronous Optical Deflection-Routed Interconnection Networks. *IEEE Transactions on Parallel and Distributed Systems*, 7(6):605–611, 1996.
7. M. Forsell, V. Leppänen, and M. Penttonen. Efficient Two-Level Mesh based Simulation of PRAMs. In *Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks, ISPAN'96*, pages 29 – 35. IEEE Computer Society, 1996.
8. L.A. Goldberg, Y. Matias, and S. Rao. An Optical Simulation of Shared Memory. In *SPAA'94, 6th Annual Symposium on Parallel Algorithms and Architectures, Cape May, New Jersey*, pages 257 – 267, June 1994.
9. M.C. Gupta. *Handbook of Photonics.* CRC Press LLC, Boca Raton, 1997.
10. D.E. Knuth. *The Art of Computer Programming, Vol. 1.* Addison-Wesley, 1968.
11. V. Leppänen and M. Penttonen. Work-Optimal Simulation of PRAM Models on Meshes. *Nordic Journal on Computing*, 2(1):51 – 69, 1995.
12. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
13. B.E.A. Saleh and M.C. Teich. *Fundamentals of Photonics.* John Wiley & Sons, Inc., U.S.A, 1991.
14. H. Schröder, O. Sýkora, and I. Vrto. Optimal Embedding of a Toroidal Array in a Linear Array. In *Foundations of Computation Theory*, pages 390 – 394, 1991.
15. A. Schuster. *Bounds and Analysis Techniques for Greedy Hot-Potato Routing*, chapter 11, pages 283–354. Kluwer Academic Publishers, 1997.
16. J. Sibeyn. Solving Fundamental Problems on Sparse-Meshes. In *Scandinavian Workshop on Algorithm Theory, SWAT'98, LNCS 1432*, pages 288 – 299, 1998.
17. T.J. Soukop, R.J. Feuerstein, and V.P. Heuring. Implementation of a fiber-optic delay-line memory. *Applied Optics*, 31(17):3233–3240, 1992.
18. L.G. Valiant. General Purpose Parallel Architectures. In *Algorithms and Complexity, Handbook of Theoretical Computer Science*, volume A, pages 943–971, 1990.
19. P.M.B. Vitányi. Locality, Communication, and Interconnect Length in Multicomputers. *SIAM Journal on Computing*, 17(4):659 – 672, August 1988.