Parallel Computing 18 (1992) 1233–1257
North-Holland

PARCO 708

# Communication algorithms for isotropic tasks in hypercubes and wraparound meshes *

Emmanouel A. Varvarigos and Dimitri P. Bertsekas

*Laboratory for Information and Decision Systems, M.I.T., Cambridge, MA 02139, USA*

Received 28 May 1991
Revised 17 November 1991, 11 February 1992

*Abstract*

Varvarigos, E.A. and D.P. Bertsekas, Communication algorithms for isotropic tasks in hypercubes and wraparound meshes, Parallel Computing 18 (1992) 1233–1257.

We consider a broad class of communication tasks, which we call isotropic, in a hypercube and in a wraparound mesh of processors. These tasks are characterized by a type of symmetry with respect to origin node. We show that executing such tasks in a minimum number of steps is equivalent to a matrix decomposition problem. We use this property to obtain minimum completion time algorithms. For a special communication task, the total exchange problem, we find algorithms that are simultaneously optimal with respect to completion time, and average packet delay. We also prove that a particularly simple type of shortest path algorithm executes isotropic tasks in time which is optimal within a small bound.

*Keywords.* Communication algorithm; hypercube; symmetric routing algorithms; optimal completion time algorithms; total exchange problem.

## 1. Introduction

The processors of a multiprocessor system, when doing computations, often have to communicate intermediate results. The interprocessor communication time may be substantial relative to the time needed exclusively for computations, so it is important to carry out the information exchange as efficiently as possible.

Algorithms for routing messages between different processors have been studied by several authors under a variety of assumptions on the communication network connecting the processors. Saad and Shultz [18,19] have introduced a number of generic communication problems that arise frequently in numerical and other methods. For example they consider the problem where each processor is required to send a separate packet to every other node; following [3], we call this the *total exchange* problem. Saad and Schultz have assumed that all packets take unit time to traverse any communication link. Processors can either transmit along all their incident links simultaneously or they can transmit along a single incident link at any one time. Johnson and Ho [11] have developed minimum and nearly minimum completion

* Research supported by NSF under Grant NSF-ECS-8519058 and by the ARO under Grant DAAL03-b6-K-0171.
*Correspondence to:* Dimitri P. Bertsekas, Laboratory for Information and Decision Systems, M.I.T, Cambridge, MA 02139, USA.

time algorithms for similar routing problems as those of Saad and Schultz but using a different communication model and a hypercube network. Their model quantifies the effects of setup time (or overhead) per packet, while it allows packets to have variable length, and to be split and be recombined prior to transmission on any link in order to save on setup time. In the model of [11], each packet may consist of data originating at different nodes and/or destined for different nodes. The extra overhead for splitting and combining packets is considered negligible in the model of [11]. Bertsekas et al. [4], and Bertsekas and Tsitsiklis [3] have used the communication model of Saad and Shultz to derive minimum completion time algorithms for several communication problems in a hypercube. In particular, they have given an algorithm for the total exchange problem that executes in a minimum number of steps ($n/2$ for an $n$-processor hypercube). Several other works deal with various communication problems and network architectures related to those discussed in the present paper; see [5,8,9,10,13,15,16,20,22,23].

In this paper, we introduce a new class of communication tasks, called *isotropic*, which are characterized by transmission requirements that are symmetric with respect to origin node (a precise definition will be given later). For example, the total exchange problem is an isotropic task; the communication problem 'looks identical' to every node. The structure of isotropic tasks can be exploited particularly well in networks that have themselves a symmetric structure, such as a hypercube and a wraparound mesh. Consequently, we restrict attention to these two networks. We use the Saad and Schultz communication model, but as we will show in Section 4, our minimum completion time results are essentially independent of the communication model used. The idea is that to achieve minimal completion time, some critical network resource must be used 100% of the time, and this constraint is limiting for any communication model.

A central result of this paper is that executing isotropic tasks on a hypercube or a wraparound mesh is equivalent to solving a matrix decomposition problem. We use this result to characterize the class of algorithms that execute isotropic tasks in minimum time. Within this class and for the total exchange problem, we identify simple and easily implementable algorithms with further optimality properties, such as minimum or nearly minimum average packet delay. No algorithms of this type have been previously discussed in the literature. Minimizing average packet delay, in addition to maximum packet delay, is important for a number of reasons. If we assume that packets require one memory space from the time they are generated to the time they are delivered at their destination, then by minimizing the average packet delay we simultaneously minimize the memory requirements. Furthermore, a small average delay is important if processors can start processing each packet as soon as it is delivered, without having to wait for the delivery of all of them. Finally, minimum average delay algorithms tend to maintain a small number of transient packets in the network at all times. Indeed our minimum average delay algorithms have optimal or near-optimal transient storage requirements.

We also consider a class of particularly simple-minded algorithms, called *greedy*, that are required to satisfy just a very weak and natural restriction; they must never leave a communication link idle as long as there is a waiting packet that can reduce its distance to its destination by using this link. An interesting new result is that any algorithm with this property executes in nearly minimum time for the total exchange problem; the deviation from optimality is bounded by a small number. Similar results can be shown for greedy algorithms applied to other isotropic communication tasks.

There are two main contributions in this paper. The first is to relate the routing problem, which is a scheduling problem with a matrix decomposition problem, which is a problem in linear algebra. Such a connection is new and quite unexpected. It provides a simpler and more powerful characterization of optimal routing algorithms for the total exchange and other related problems than in earlier works (e.g. [4]). It also allows simple and elegant analyses of minimum average delay algorithms and the suboptimal greedy algorithms, which despite their practical significance, have not been discussed so far in the literature. We note that earlier works in data communications ([1] and [24]) have also shown the equivalence of certain optimal time slot allocation problems and matrix decomposition problems. However, these works involve a very different context where there is one transmitter, and several receivers connected with a direct link to the transmitter; network situations are not addressed and symmetry plays no role.

The second main contribution of this paper is to introduce isotropic tasks as a practically important and analytically interesting class of communication problems. It is clear that there is an incentive to formulate new routing problems in terms of isotropic tasks, whenever this is reasonable, in order to take advantage of the corresponding simple and elegant analysis. For example, it may be fruitful to analyze a 'nearly isotropic' communication problem as an isotropic problem with appropriate modifications. Examples of such analyses will be given in future publications.

The paper is organized as follows. Sections 2 through 7 deal with the hypercube, while Section 8 deals with the wraparound mesh. Section 2 defines the class of the isotropic tasks and introduces the key notion of the *task matrix*. A lower bound for the completion time of both isotropic and non-isotropic tasks is also given. Section 3 deals with the evolution of the task matrix when symmetric routings are used. It also transforms the problem of minimizing the task's execution time into the problem of writing the task matrix as the sum of a minimum number of permutation matrices. The solution to the matrix decomposition problem is given in Section 4. Section 5 describes greedy algorithms and proves their near-optimal performance. Algorithms with both optimal completion time and optimal or near optimal average delay for the total exchange problem are found in Section 6. Section 7 treats the case where each node can use simultaneously at most $k$ rather than all its incident links. Finally, Section 8 extends the hypercube algorithms and analysis to the case of a $d$-dimensional wraparound mesh.

## 2. The task matrix

We first introduce some terminology. The $d$-dimensional hypercube network has $n = 2^d$ nodes and $d2^{d-1}$ links. Each node can be represented by a $d$-bit binary string called *identity number*. There are links between nodes which differ in precisely one bit. As a consequence, each node has $d = \log n$ incident links. When confusion cannot arise, we refer to a $d$-cube node interchangeably in terms of its binary representation and in terms of the decimal representation of its identity number. Thus, for example, the nodes $(00 \cdots 00)$, $(00 \cdots 01)$, and $(11 \cdots 11)$ will also be referred to as nodes 0, 1, and $2^d - 1$, respectively. The *j-type* link (or $j$-link) of nodes $s = (s_1 \ldots s_j \ldots s_d)$ is the link connecting node $(s_1 \ldots s_j \ldots s_k)$ with node $(s_1 \ldots \bar{s}_j \ldots s_d)$. (We denote by $\bar{x}$ the complement of the binary number $x$, that is, $\bar{x} = 1 - x$.)

Given two nodes $v$ and $w$, the node $v \oplus w$ is the node with binary representation obtained by a bitwise exclusive OR operation of the binary representations of nodes $v$ and $w$.

The *Hamming distance* between two nodes is the number of bits in which their identities differ. The number of links on any path connecting two nodes cannot be less than the Hamming distance of the nodes. Furthermore, there is a path with a number of links which is equal to the Hamming distance, obtained, for example, by switching in sequence the bits in which the bit representations of the nodes differ (equivalently, by traversing the corresponding links of the hypercube). Such a path is referred to as a *shortest path* in this paper.

Information is transmitted along the hypercube links in groups of bits called *packets*. In our algorithms we assume that the time required to cross any link is the same for all packets, and is taken to be one unit. We assume that packets can be simultaneously transmitted along a link in both directions, and that their transmission is error free. Only one packet can travel along a link in each direction at any one time; thus, if more than one packet are available at a node and are scheduled to be transmitted on the same incident link of the node, then only one of these packets can be transmitted at the next time period, while the remaining packets must be stored at the node while waiting in queue. With the exception of Section 7 we assume that all incident links of a node can be used simultaneously for packet transmission and reception. Finally, we assume that each of the algorithms proposed in this paper is simultaneously initiated at all processors.

We now defined the communication tasks that are the subject of this paper.

**Definition 1.** A *communication task* $\mathcal{G}$ is defined as a set of triplets $(v, w, k)$, where $v$ is a node (source), $w$ is a node (destination), and $k$ is an integer (the number of packets whose source is $v$ and whose destination is $w$).

**Definition 2.** A communication task $\mathcal{G}$ is called *isotropic* if for each packet that node $v$ has to send to node $w$, there is a corresponding packet that node $v \oplus x$ has to send to node $w \oplus x$, where $v$, $w$, and $x$ are arbitrary nodes. Mathematically:

$$(v, w, k) \in \mathcal{G} \quad \Rightarrow \quad \text{for all nodes } x \text{ we have } (v \oplus x, w \oplus x, k) \in \mathcal{G}.$$

An example of an isotropic task is the total exchange, where $\mathcal{G}$ consists of all the triplets $(v, w, 1)$ as $v$ and $w$ range over all the pairs of distinct nodes [one packet for every origin-destination pair $(v, w)$].

In the algorithms that we propose, the packets carry with them a $d$-bit string called *routing tag*. The routing tag of a packet is initially set at $v \oplus w$, where $v$ is the source and $w$ is the destination of the packet. As the packet is transmitted from node to node, its routing tag changes. If at time $t$ a packet resides at a node $s$ and has $w$ as destination, then its routing tag is $s \oplus w$. For example, a packet which is currently at node 001010 and is destined for node 101000, has routing tag 100010.

An important data structure that will be used by our routing algorithms is that of the *task matrix* of node $s$ at time $t$, which will be denoted by $T_t(s)$. The task matrix $T_t(s)$ is defined for both isotropic and non-isotropic tasks and is a binary matrix whose rows are the routing tags of all the packets that are queued at node $s$ at time $t$. The routing tags appear as rows of the initial task matrices $T_0(s)$ in some arbitrarily chosen order. When no packets are queued at node $s$ at time $t$, the task matrix $T_t(s)$ is by convention defined to be a special matrix denoted $Z$. A task is said to be completed at time $t$ if $T_t(s) = Z$ for all $s$. The smallest $t$ for which the task is completed under a given routing algorithm is called the *completion time* of the algorithm.

A communication task can equivalently be defined in terms of its initial task matrices $T_0(s)$, $s = 0, \ldots, n-1$. The task is isotropic if and only if the task matrices $T_0(s)$ are the same for all nodes $s$. In what follows, whenever there is no reason to distinguish among the nodes, we simply denote the task matrix at time $t$ with $T_t$. When such a notation is used, we implicitly mean that $T_t(s) = T_t$, for all $s$. The initial task matrix for the total exchange problem is illustrated in *Fig. 1*.

We will now derive a lower bound for the completion time of any communication task (isotropic or non-isotropic).

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Fig. 1. The task matrix for the total exchange problem has $n-1$ rows and $d$ columns. The figure illustrates the case where $d = 3$.

**Theorem 1.** *Let $\mathcal{T}$ be the completion time of any algorithm that executes a task with initial task matrices $T_0(s)$, $s = 0, 1, \ldots, n-1$. Let also $r_i(s)$ (or $c_i(s)$) denote the sum of the elements of the ith row (or column, respectively) of the task matrix $T_0(s)$. Then the following inequality holds*

$$\mathcal{T} \geq \max_{i,j} \max\left\{\frac{1}{n}\sum_{s=0}^{n-1} c_j(s),\ \max_s r_i(s)\right\},$$

*where the outer maximization is carried out over all rows i and columns j.*

**Proof.** The column sum $c_j(s)$ of the $j$th column of $T_0(s)$ is equal to the number of packets that reside at node $s$ at time $t = 0$ and have the $j$th bit of their routing tag equal to 1. To arrive to their destination, these packets have to use a $j$-link at some future time. Thus, $\Sigma_s c_j(s)$ packets are going to use $j$-type links during the execution of the task. Since each node has only one $j$-link, there are only $n$ links of $j$-type in the hypercube. Taking into account that no two packets can be transmitted on the same link in the same time slot, we conclude that

$$\mathcal{T} \geq \frac{\Sigma_s c_j(s)}{n}$$

for all columns $j$. Therefore,

$$\mathcal{T} \geq \frac{1}{n} \max_{j=1,\ldots,d}\left(\sum_{s=0}^{n-1} c_j(s)\right). \tag{1}$$

On the other hand, the packet corresponding to the $i$th row of $T_0(s)$ is at a Hamming distance $r_i(s)$ from its destination. Thus the time $\mathcal{T}$ required to complete the task is at least $r_i(s)$ for all rows $i$ and nodes $s$. This gives

$$\mathcal{T} \geq \max_{i,s} r_i(s). \tag{2}$$

By combining (1) and (2), we finally obtain

$$\mathcal{T} \geq \max_{i,j} \max\left\{\frac{1}{n}\sum_{s=0}^{n-1} c_j(s),\ \max_s r_i(s)\right\},$$

where the outer maximization is carried out over all rows $i$ and columns $j$. □

The preceding lower bound cannot always be attained by some algorithm. The following Corollary 1 specializes this lower bound for the case of isotropic tasks. As we will show later there is always an algorithm that achieves the lower bound of Corollary 1.

**Definition 3.** The *critical sum h* of a matrix is equal to $\max_{i,j}\{r_i, c_j\}$, where $r_i$ is the sum of the entries of row $i$, $c_j$ is the sum of the entries of column $j$, and the maximization is performed over all rows $i$ and columns $j$. A row or column with sum of entries equal to $h$ is called a *critical line.*

**Corollary 1.** *Let an isotropic communication task have initial task matrix $T_0$ and h be the critical sum of $T_0$. Then a lower bound for the time $\mathcal{T}$ required to complete the task is h.*

**Proof.** Using Theorem 1 and the fact that for isotropic tasks we have $T_0(s) = T_0$, $c_j(s) = c_j$, $r_j(s) = r_j$ for all nodes $s = 0, 1, \ldots, n-1$, we obtain $\mathcal{T} \geq \max_{i,j}\{c_j, r_i\} = h$, for any algorithm that executes the task. □

## 3. Symmetric routing algorithms

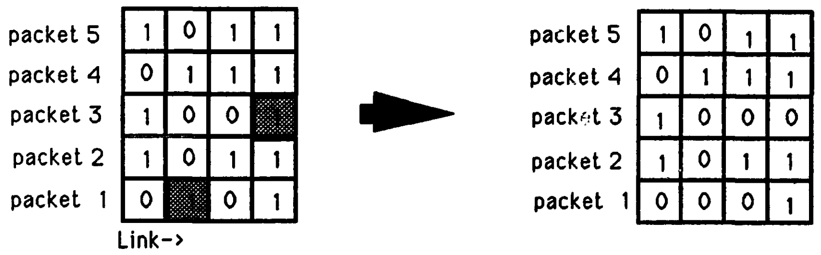
In this section we will be interested in isotropic tasks and a class of routing algorithms that satisfy a certain symmetry condition.

**Definition 4.** Given a task matrix $T_t(s)$ for each node $s$ at time $t$, a *switching scheme* with respect to $T_t(s)$ is a collection of matrices $\{S_t(s) \mid s = 0, \ldots, n-1\}$ with entries 0 or 1. The matrix $S_t(s)$ has the same dimensions as $T_t(s)$, satisfies $S_t(s) \leq T_t(s)$ (i.e. if an entry of $T_t(s)$ is a zero, the corresponding entry of $S_t(s)$ must also be zero), and has at most one nonzero entry in each row or column. The switching scheme is called *symmetric* if for every $t$ the matrices $S_t(s)$ are independent of $s$, that is, if for some matrix $S_t$ we have $S_t(s) = S_t$ for all $s$.

Given a time $t \geq 0$ and a task matrix $T_t(s)$ for each node $s$, a switching scheme $\{S_t(s) \mid s = 0, \ldots, n-1\}$ with respect to $T_t(s)$ defines the packet (if any) that will be transmitted on each link at the time slot beginning at time $t$. In particular, if the $(i, j)$th element of $S_t(s)$ is a one, the packet corresponding to the $i$th row of $T_t(s)$ will be transmitted on the $j$th link of node $s$. The requirement that each column of $S_t(s)$ contains at most one nonzero entry guarantees that at most one packet is scheduled for transmission on each link.

The task matrices at a given time slot together with a corresponding switching scheme, define the task matrices for the next time slot. Given a communication task defined by the task matrices $T_0(s)$, $s = 0, \ldots, n-1$, a routing algorithm can be defined as a sequence $\{S_0(s), S_1(s), \ldots\}$, such that $S_0(s)$ is a switching scheme with respect to the task matrix $T_0(s)$, $S_1(s)$ is a switching scheme with respect to the task matrix $T_1(s)$ (which is defined by $T_0(s)$ and $S_0(s)$), and, recursively, $S_{t+1}(s)$ is a switching scheme with respect to the task matrix $T_{t+1}(s)$ (which is defined by $T_t(s)$ and $S_t(s)$).

The key fact, proved in the following theorem, is that if at some time $t$, the task matrices are the same for all nodes $s$, and a symmetric switching scheme with respect to $T_t(s)$ is used, then the next task matrices $T_{t+1}(s)$ will be the same for all nodes. As a result, for an isotropic task, one may use a routing algorithm defined by a sequence of symmetric switching schemes. Such a routing algorithm will be called *symmetric.* Its action is specified at a single node and is essentially replicated at all the other nodes; this is a very desirable property for implementation purposes.



Packet 1 is transmitted on link 2 and packet 3 is transmitted on link 4.

Fig. 2. The change in the task matrix due to packet transmissions (packets 1 and 3 are transmitted on links 2 and 4, respectively).

**Theorem 2.** *Assume that for a given routing algorithm, at some time t we have a set of nonzero task matrices $T_t(s)$, which are the same for all nodes s. Then if $S_t$, a symmetric switching scheme with respect to $T_t(s)$ is used by the algorithm at time t, the task matrices $T_{t+1}(s)$ will be the same for all s. In particular, we have*

$$T_t(s) = T_t,\ S_t(s) = S_t,\ \forall s \quad \Rightarrow \quad T_{t+1}(s) = T_{t+1},\ \forall s,$$

*where $T_{t+1}$ is a task matrix consisting of the nonzero rows of the matrix $T_t - S_t$, except if $T_t = S_t$ in which case $T_{t+1}$ is equal to the special matrix Z and the algorithm terminates.*

**Proof.** Suppose that at time slot $t$, node $s$ sends a packet with routing tag $x_1 \cdots x_j \cdots x_d$ over its $j$-link to node $s \oplus e_j$. Then by the symmetry assumption, node $s \oplus e_j$ also sends a packet with routing tag $x_1 \cdots x_j \cdots x_d$ over its $j$-link to node $(s \oplus e_j) \oplus e_j = s$. This packet arrives at node $s$ with routing tag $x_1 \cdots \bar{x}_j \cdots x_d$. Thus each row of the task matrix $T_t$, which corresponds to a packet transmitted at slot $t$, is replaced by a row $x_1 \cdots \bar{x}_j \cdots x_d$ if $x_1 \cdots \bar{x}_j \cdots x_d$ is nonzero and is discarded otherwise; see *Fig.* 2. Since the transmitted packets (if any) on the $j$-link correspond to the nonzero entry of the $j$th column of the matrix $S_t$, we conclude that $x_j = 1$ and, therefore, $\bar{x}_j = 0$. Thus the routing tag $x_1 \cdots \bar{x}_j \cdots x_d$ is either zero or else it is a row of the matrix $T_t - S_t$. By symmetry, at the beginning of slot $t$ there is a packet with routing tag $x_1 \cdots x_j \cdots x_d$ at each node, and this packet will be replaced (if transmitted) by a packet with routing tag $x_1 \cdots \bar{x}_j \cdots x_d$ at the end of the slot $t$ if $x_1 \cdots \bar{x}_j \cdots x_d$ is nonzero and will exit the network otherwise. Thus the task matrix will change in the same way for each node. □

From Theorem 2 we see that if the communication task is isotropic with initial task matrix $T_0$, we can specify a symmetric routing algorithm by a sequence of symmetric switching schemes $S_0, S_1, \ldots$ as follows:

*Symmetric Routing Algorithm specification*

The initial task matrix $T_0$ of the isotropic task is given. For $t = 0, 1, \ldots,$ given the task matrix $T_t$, $S_t$ must be a symmetric switching scheme with respect to $T_t$; the task matrix $T_{t+1}$ is then specified by the nonzero rows of $T_t - S_t$, unless $T_t = S_t$ in which case the algorithm terminates.

We see therefore that a symmetric routing algorithm that terminates after $k + 1$ time slots amounts to a decomposition of the initial task matrix $T_0$ into a sum

$$T_0 = \bar{S}_0 + \bar{S}_1 + \cdots + \bar{S}_k,$$

where each $\bar{S}_i$, $i = 0, \ldots, k$, is a binary nonzero matrix with the same dimension as $T_0$, and with at most one nonzero element in each column or row. The corresponding switching schemes $S_i$, $= 0, \ldots, k$, consist of the nonzero rows of the matrices $\bar{S}_i$, $i = 0, \ldots, k$, respectively.

Thus, by restricting attention to symmetric routings, our original problem of finding optimal routings for isotropic communication tasks has been reduced to the simpler problem of 'clearing' the $T_0$ matrix (i.e. making all its entries equal to 0) in a minimum number of steps. At each step we are allowed to make 0 up to $d$ entries, provided that these entries do not belong to the same row or column. The entries should not belong to the same row because at each step a packet cannot be transmitted on more than one link. The entries should not belong to the same column so that no two packets will use the same outgoing link. We will derive optimal algorithms within this class. These algorithms will be shown to attain the lower bound of Theorem 1, so they are guaranteed to be optimal within the class of all routing algorithms.

## 4. Optimal completion time algorithms

We consider the problem of clearing the task matrix in the minimum number of steps. At each step we are allowed to clear at most 1 entry from each row or column. Our analysis will use some theorems and tools that were also used in [1] and [24] in a different context. We first introduce some more definitions. For any matrix, we use the term *line* to refer to a row or column of the matrix.

**Definition 5.** A *perfect* matrix is a square matrix with nonnegative integer entries and with the property that the sum of the entries of each line is the same for all lines.

**Definition 6.** A *permutation matrix* is any matrix with entries equal to 0 or 1 with the property that each line of the matrix has at most one nonzero entry.

It can be noted that the nonzero entries of a permutation matrix form an independent set of entries in the sense that no two of them belong to the same line. As a result, a set of entries of the task matrix which form a permutation submatrix can be cleared during the same step. In particular a permutation matrix $S$ can be used as a switching scheme for any node at any time as long as the task matrix at that node and time satisfies $S \leq T$ (see Definition 4). An important result for our purposes is Hall's Theorem (see [17] or [2], p. 120), which states that a perfect matrix can be written as a sum of $h$ permutation matrices, where $h$ is the sum of the entries of its lines. The following two theorems extend slightly Hall's Theorem.

**Theorem 3.** *Given any nonnegative integer square matrix $M$ with critical sum $h$, there exists a nonnegative integer matrix $E$ such that $M + E$ is a perfect matrix with critical sum $h$.*

**Proof.** We give a constructive proof. Let $r_i$ ($c_j$) be the sum of the entries of row $i$ (column $j$). We augment each element $M_{ij}$ of the matrix such that $r_i < h$ and $c_j < h$ by $\min(h - r_i, h - c_j)$ one at a time and update $M$ after each change, thus obtaining a matrix with at least one more critical line and critical sum equal to $h$. At the end of this process we will have added to $M$ a nonnegative integer matrix $E$, thereby obtaining a matrix $M + E$ with critical sum $h$ and such that for each pair $(i, j)$ either row $i$ is critical or column $j$ is critical. For this to be true, either all rows of $M + E$ must be critical or else all columns must be critical. Assume without loss of generality that all rows are critical. Then, the sum of the elements of $M + E$ is $mh$, where $m$ is the number of rows and columns, while each column sum is at most $h$. It follows that each column sum of $M + E$ is exactly equal to $h$, so each column is critical, and $M + E$ is perfect. □

**Theorem 4.** *A nonnegative integer matrix with critical sum h can be written as the sum of h permutation matrices.*

**Proof.** Let $T$ be a nonnegative integer matrix with dimensions $m \times d$. We assume, without loss of generality that $m \geq d$. We can extend $T$ to a square matrix $M = [T \mid 0]$ by adding $m - d$ zero columns. Then, by Theorem 3, $M$ can be augmented to a perfect matrix $M + E$ with line sums equal to $h$. By Hall's theorem we conclude that $M + E$ can be written as a sum $\Sigma_{k=1}^{h} P_k$ of $h$ permutation matrices $P_1, P_2, \ldots P_h$. Since $E$ has nonnegative integer entries, $M$ can also be written as a sum $\Sigma_{k+1}^{h} \hat{P}_k$ of square permutation matrices $\hat{P}_1, \hat{P}_2, \ldots \hat{P}_h$; each $\hat{P}_k$ is obtained by setting to zero some of the entries of $P_k$. Since $M = [T \mid 0]$, $T$ can be written as a sum of $h$ permutation matrices of dimension $m \times d$. □

The following is the main result of this section.

**Theorem 5.** *The optimal completion time for an isotropic communication task is equal to the critical sum h of its task matrix.*

**Proof.** From Theorem 4 we know that the initial task matrix $T_0$ can be written as the sum $\Sigma_{k=1}^{h} \bar{S}_k$ of permutation matrices $\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_h$. Consider the symmetric switching scheme $\{S_k\}$, where for $k = 1, \ldots, h$, $S_k$ is obtained from $\bar{S}_k$ by removing the zero rows. Then the task matrix at times $t$ with $1 \leq 1 < h$ consists of the nonzero rows of $T_0 - \Sigma_{k=1}^{t} \bar{S}_k$, and at time $t = h$ is equal to $Z$. Hence the communication task is completed after $h$ steps. Since, by Theorem 1, $h$ is also an upper bound, the corresponding symmetric routing must be optimal. □

It is easy to see that if at any step we clear one entry from each critical line of the matrix $T_t$ matrix, we can clear the task matrix within the optimal number of steps. On the other hand we cannot clear the matrix in $h$ steps if we are not clearing an entry from each critical line at each step. In order to see this, let $h_t$ be the critical sum of the task matrix $T_t$. We observe that the critical sum of the task matrix can decrease by at most 1 at each step ($h_t \geq h_{t-1} - 1$). Thus, if during slot $t$ there is a critical line which is not served, then $h_t = h_{t-1}$ and it is not possible to clear the matrix in $h_0 = h$ steps. Thus, we conclude that a symmetric switching scheme achieves optimal completion time if and only if it adheres to the following rule:

*Optimal Completion Time Rule (abbreviated OCTR):*
At each step an entry is cleared from each critical line of the task matrix.

We finally note that if the initial task matrix contains a column, say the $j$th, which is critical, then the $j$-type links constitute a critical resource in the sense that they must all be used 100% of the time during the execution of any optimal completion time algorithm. Under these circumstances it is impossible to reduce the optimal completion time by using an algorithm that allows packets to be split and be recombined during its course. In the unusual case where the only critical lines are rows, the optimal completion time could be reduced under a different communication model, e.g. wormhole routing [12,7].

We will now use the preceding results to find optimal algorithms for the total exchange task.

### *4.1. Total exchange*

In the total exchange task, we have initially $n - 1$ packets with different routing tags queued at each node. The tags are different because each node has to send $n - 1$ distinct packets, one to each node of the hypercube. The critical sum of the initial task matrix $T_0$, and therefore also the optimal completion time, is $n/2$. (To see this, note that if we add to 00...00 string as an $n$th row of $T_0$, half of the entries of each column will be equal to 0 and half of them will be equal to 1.) Any algorithm that works according to the OCTR is optimal as far as completion time is concerned. Since there is a decomposition of $T_0$ into $n/2$ distinct permutation matrices and these $n/2$ matrices can be cleared in any desired order, it follows that the number of optimal total exchange algorithms is at least $(n/2)!$. In fact, there are additional optimal algorithms because there are more than one decompositions of the task matrix into permutation matrices. This provides a lot of flexibility to select an algorithm that is optimal not only with respect to completion time but also with respect to some other optimality criteria. Section 6 describes an algorithm that achieves optimal completion time and optimal average delay for the case when the dimension $d$ of the hypercube is a prime number. When $d$ is not prime, the same algorithm achieves near-optimal average delay, as well as optimal completion time. (We say that the average delay of an algorithm is "near-optimal" if it agrees with the optimal average delay in the term of highest order of magnitude.)

A generalization of the total exchange task is the $(K, L)$ *neighborhood exchange* task. In this task, every node $s$ has to send a packet to all the nodes $r$ whose Hamming distance from $s$ satisfies $K \leq \text{Ham}(s, r) \leq L$. For $K = 1$ and $L = d$ we get the total exchange problem but for $K \neq 1$ and/or $L \neq d$, this task apparently has not been discussed elsewhere. The initial task matrix $T_0$ has as rows all the $d$-long binary strings with $i$ ones, where $K \leq i \leq L$. The critical sum of this matrix is

$$h = \max\left\{L, \sum_{i=K}^{L} \frac{\binom{d}{i} i}{d}\right\}.$$

To see this, note that the task matrix has $\binom{d}{i}$ rows, each having $i$ ones. Since by symmetry the $d$ columns have equal columns sums, each column sum will be equal to $\sum_{i=K}^{L}\binom{d}{i}i/d$. By Theorem 5, the critical sum $h$ is the time required to execute the task.

## 5. Using greedy algorithms

In this section we will show that any 'reasonable' switching scheme (it does not have to be deterministic) will give a completion time for an isotropic task which is larger than the optimal by at most $d - 1$ time units. By the term 'reasonable' switching scheme, we mean a symmetric switching scheme $\{S_0, S_1, \ldots\}$ with the property that a communication link is never idle while there is a waiting packet that can reduce its distance to its destination by using this link. Mathematically, we require that for all $t$ and $j = 1, \ldots, d$, if the $(i, j)$th entry of $T_t$ is nonzero, then either the $i$th row of $S_t$ is nonzero or the $j$th column of $S_t$ is nonzero (or both). We call this the *non-wasting* property and we call the corresponding switching scheme *greedy*.

Since the task is isotropic and we are using a symmetric switching, the task matrices $T_t$ are the same at all nodes at each time $t$. Let $h_c$ and $h_r$ be the maximal column and row sum of $T_0$, respectively. We will prove that the $(i, j)$th entry $(T_0)_{ij}$ of $T_0$ will become zero after at most $h_r + h_c - 1$ steps. Indeed, assume that $(T_0)_{ij}$ is initially not 0 and that $(T_0)_{ij}$ is not cleared during the steps $1, 2, \ldots, h_r + h_c - 2$. Then the non-wasting property implies that one entry of row $i$ or one entry of column $j$ (or both) were cleared at each of these steps. Thus by time $r_i + c_j - 2$, all the entries of the $i$th row and the $j$th column, except for $(T_0)_{ij}$, have been cleared. Then by the non-wasting property, we conclude that at time $r_i + c_j - 1$ the entry $(T_0)_{ij}$ is cleared. Since $r_i + c_j \leq h_r + h_c$ the result follows.

Since $h_r \le d$ and $h_c$ is at most equal to the optimal completion time, we see that a greedy algorithm executes an isotropic task within time that is within $d-1$ time steps of the optimal. For tasks with $h_r < d$ this estimate can be improved. For example a greedy algorithm executes the $(K, L)$ neighborhood exchange task within $L-1$ steps of the optimal time.

## 6. Algorithms achieving simultaneously optimal completion time and optimal average delay

In the previous sections we have only been concerned with completion time optimality. A second important aim, which has not been considered so far in the literature, is the simultaneous minimization of the average delay suffered by a packet. In particular if $W_i$ is the time between the start of the execution of the task and the time that packet $i$ reaches its destination, we want to minimize the average delay, given by $\sum_{i=1}^{N} W_i/N$, where $N$ is the number of packets involved in the task. In this section we find algorithms for the total exchange and neighborhood exchange problems that achieve both optimal (or near-optimal when $d$ is not prime) average delay and optimal completion time.

In order to achieve optimal completion time, the OCTR is followed at every step. With this rule, packets follow shortest paths to their destination and for both the total exchange and the neighborhood exchange tasks, links are utilized 100% of the time. For algorithms where these properties hold, we will see that a sufficient condition to achieve optimal average delay is to transmit at each time slot the packets that are nearest to their destination (equivalently, whose routing tags have the least number of 1's). The intuition behind this comes from queueing theory situations where to achieve minimum average customer delay, the customers requiring less service should be served first. This priority rule is made precise in the following directive.

*Optimal Average Delay Directive (abbreviated OADD):*

No packet is, at any slot, transmitted over a link of the network if some packet which is nearer to its destination is not transmitted during the same slot.

In the appendix we prove the following theorem, which holds for any network of directed links – not just a hypercube.

**Theorem 6.** *Consider a network and an arbitrary communication task. An algorithm that executes the task and in which*
(a) *packets are sent to their destinations over shortest paths,*
(b) *all links are used at all time slots prior to the algorithm's termination (100% utilization) and at every slot packets are transmitted according to the OADD,*
*is optimal with respect to the average delay criterion.*

The question that arises now is whether we can follow the OADD at each step simultaneously with the optimal completion time rule of the previous section. In general, insisting on optimal completion time can prevent us from minimizing the average delay. Fortunately, at least for the total exchange and the neighborhood exchange problems it is possible to achieve either optimality or near-optimality within a small bound.

### 6.1. *Case where d is prime*

Consider the total exchange problem. We will show that when $d$ is prime we can simultaneously achieve strictly optimal average delay and completion time. In the algorithm that we will propose the packets with initial routing tags (111...111), (011...111), (101...111),...,(111...101), (111...110) will not be transmitted along any link prior to the last $d$ steps. This implies that during the first $\frac{n}{2}-d$ slots the maximum column sum of the task matrix $T_t$ is greater than $d$. As a result, the critical lines of the matrices $T_t$, $t = 1,\ldots,(n/2)-d$ will be columns. Therefore, during the first $n/2-d$ steps we are allowed to clear any $d$ entries of the task matrix that we want, provided that at each step we clear exactly one entry from each column (equivalently from each critical line), with no two of them belonging to the same row.

In the following discussion we will show how the previous rules (OCTR and OADD), which are sufficient in order to achieve the two kinds of optimality, can be implemented for the total exchange task for the case where $d$ is a prime number. (The neighborhood exchange problem is simpler and will be discussed briefly later.) The proposed algorithm will incorporate the idea of equivalence classes that was introduced in [3] (Section 1.3) for the purpose of constructing a multinode broadcast algorithm for the $d$-cube. The reader should refer to [3] for a more complete description of these concepts.

The set of all possible routing tags is partitioned into $d$ sets $N_k$, $k=1,\ldots,d$. The set $N_k$ has $\binom{d}{k}$ elements each of which is a routing tag having $k$ unity bits and $d-k$ zero bits. Each set $N_k$ is in turn partitioned in disjoint subsets $R_{k1},\ldots,R_{kn_k}$, which are equivalence classes under a single bit rotation to the left.

We define a layered graph with $n$ nodes. Each node is identified by a $d$-long binary number and corresponds to a row (or routing tag) of $T_0$. Each layer of the graph contains $d$ routing tags belonging to the same equivalence class. The links of the graph start from an element of $R_{ij}$ and end at an element of $R_{(i-1)k}$ for some $k$. The links have the following useful property: if $s$ and $t$ are distinct routing tags belonging to the same layer and $(s, s\oplus e_j)$, $(t, t\oplus e_l)$ are the arcs starting at nodes $s$ and $t$, then $j\neq l$. Note that because $d$ is prime, each class $R_{ij}$ has $d$ elements, as illustrated in *Fig. 3.*

We can describe now an algorithm that clears the entries of the task matrix and follows both OCTR and OADD, assuming $d$ is prime. The algorithm consists of $\lceil(n-1)/d\rceil-1$ phases (from *Fig. 3.* it is easy to see that for $d$ prime $\lceil(n-1)/d\rceil-1=(n-2)/d)$. Each of the $\lceil(n-1)/d\rceil-2$ first phases corresponds to the clearance of the routing tags (rows of the task matrix) of a specific equivalence class $R_{ij}$. Because the routing tags that belong to different equivalence classes do not always have the same number of ones, each class requires

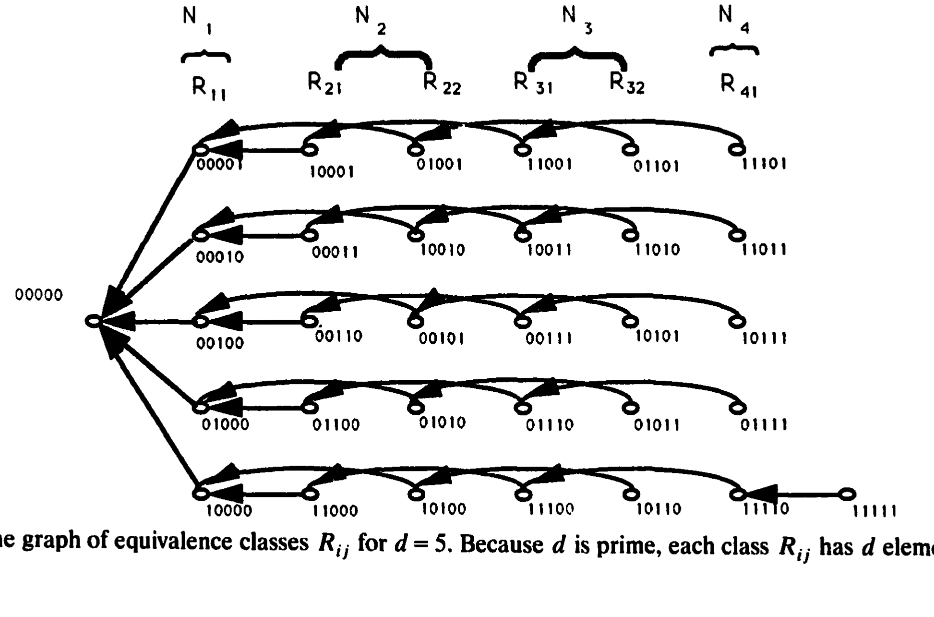


Fig. 3. The graph of equivalence classes $R_{ij}$ for $d=5$. Because $d$ is prime, each class $R_{ij}$ has $d$ elements.

a different number of slots to be cleared. The equivalence classes are cleared in the order

$$R_{11}R_{21}\ldots R_{2n_2}\ldots R_{(d-2)n_{d-2}}\{R_{(d-1)1}(11\ldots 1)\}. \tag{3}$$

The clearance of both $R_{(d-1)1}$ and (11...11) takes place during the last phase and will be described later.

At phase 1, the rows of $T_0$ corresponding to the $d$ routing tags of class $R_{11}$ are selected. Since each routing tag of this class has only one unity bit and since this bit is at a different position for each tag, the routing tags of class $R_{11}$ can be cleared in one time slot. Therefore, the first phase requires one time slot. At phase 2, the routing tags of the next class $R_{21}$ are selected to get cleared. In order to clear the corresponding rows of the task matrix we first find the routing tags of this class on the graph. At each slot we clear the bits of the routing tag in the order that the arcs of the graph indicate. Thus if an arc on the graph is pointing from $s_1\ldots s_k\ldots s_d$ ($s_k = 1$) to $s_1\ldots \bar{s}_k\ldots s_d$ and a row of the task matrix is equal to $s_1\ldots s_k\ldots s_d$, then when this row is cleared the first entry to be cleared is the row's $k$th entry. The next entry of that row that will be cleared depends on the successor of node $s_1\ldots \bar{s}_k\ldots s_d$ on the graph. If the successor of $s_1\ldots \bar{s}_k\ldots s_d$ is, say, $s_1\ldots \bar{s}_k\ldots \bar{s}_j\ldots s_d$ then the $j$th bit is the next entry of the row to be cleared. Thus, for $d = 5$ we see from *Fig. 3* that at the second phase the packets with routing tags 10001, 00011, 00110, 01100, 11000 are cleared. At the first slot of this stage these packets are transmitted on links 1, 5, 4, 3, 2, respectively, and the task matrix changes to one having routing tags 00001, 00010, 00100, 01000, 10000 instead of those of class $R_{21}$. At the second slot of the second phase these routing tags are cleared in the obvious way.

This procedure continues up to the point where only class $R_{(d-1)1}$ and the single member class 11...11 remain. At this point, we cannot clear the class $R_{(d-1)1}$ because we will violate the optimal completion time rule. This happens because now the row corresponding to the routing tag 11...11 has also become critical. In order to follow the OCTR, one entry from each critical line of the task matrix must be cleared at each step. In the last phase both the rows and the columns are critical lines. One of the ways to clear the last two equivalence classes is indicated in *Fig. 4*. All it does is follow the OCTR rule.

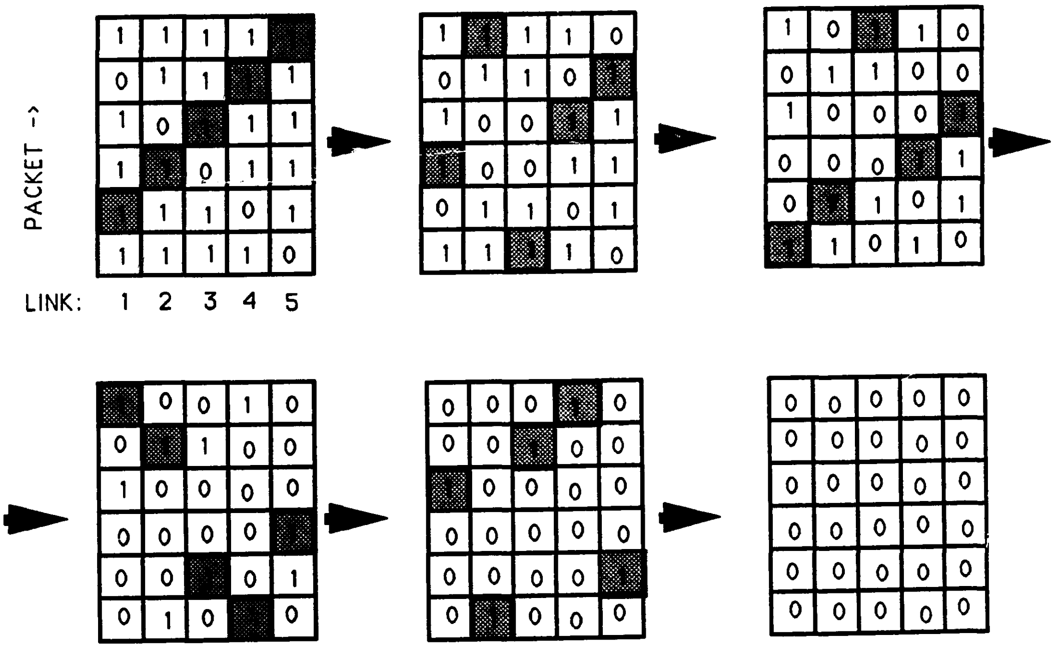


Fig. 4. The clearance of the class $R_{(d-1)1}$ together with (11 ··· 11) for $d = 5$.

Both the OCTR and the OADD are followed up to the last phase of the total exchange algorithm. At this point a slight deviation from OADD takes place, because the row 11...11 begins getting cleared before the class $R_{(d-1)1}$ has been cleared. The question is whether by violating the OADD during the last $d$ slots of the algorithm, the average delay optimality is lost. (The completion time optimality is not lost because we still follow the OCTR.)

The answer to this question is that strict average delay optimality is still maintained. In order to see that, consider a modification of the original algorithm. In particular the modified algorithm is the same with the original one during the first $\lceil (n-1)/d \rceil - 2$ phases and differs from it in that the OADD is followed at the last phase too. In other words, in the modified algorithm the class $R_{(d-1)1}$ is completely cleared before starting to transmit the packets with routing tags $11\cdots 11$. Although the modified algorithm is suboptimal with respect to completion time, it is guaranteed to achieve optimal average delay. This is because the corresponding schedule in the auxiliary problem (see the appendix) satisfies the properties of Prop. 1. Therefore, in order to prove that the original algorithm has optimal average delay, it is enough to show that it has the same average delay with the modified algorithm. In order to see that, we can forget about the first $\lceil (n-1)/d \rceil - 2$ phases and consider only the additional delay that the last phase $\lceil (n-1)/d \rceil - 1$ introduces for the last $d+1$ packets. If OADD were followed at the last phase also, this average additional delay would be $((d-1)d + 2d - 1)/(d+1) = d - 1/(d+1)$ for each of the last $d+1$ packets. If the scheduling indicated in *Fig. 4* is followed, the average delay for the last phase of the $d+1$ packets will be equal to $(d^2 + d - 1)/(d+1) = d - 1/(d+1)$, again. Since OADD is followed up to the last phase and the deviation from it during the last phase does not cause any additional delay, we conclude that the original algorithm achieves optimal average delay as well as optimal completion time.

## 6.2. *Evaluation of the optimal average delay for total exchange when d is prime*

We now calculate the optimal average delay for the total exchange problem, for the case where the dimension $d$ of the hypercube is prime. Since the average delay of the proposed algorithm is equal to the average delay of the modified algorithm (both described in the previous subsection) it is enough to calculate the average delay of the modified algorithm. The average delay of a packet given that its initial routing tag belongs to the set $N_i$ is

$$D_i = \frac{M_i + m_i}{2},$$

where $m_i$ and $M_i$ are the minimum and maximum delay, respectively, suffered by packets with initial routing tags belonging to $N_i$. For the sets $N_1$, $N_{d-1}$, $N_d$ we have that $m_1 = M_1 = 1$, $m_{d-1} = M_{d-1}$, $M_d = m_d$. However, in general, $m_i$ and $M_i$ need not be equal.

It is not difficult to see that for $i \geq 1$:

$$m_i = i + \sum_{j=1}^{i-1} \frac{\binom{d}{j}}{d} j,$$

and

$$M_i = m_{i+1} - (i+1), \quad \text{for } i = 1, 2, \ldots, d-1, \quad \text{and } M_d = m_d = \frac{n}{2} + d - 1.$$

Since there are $\binom{d}{i}$ packets with initial tags belonging to class $N_i$, it is concluded that the average delay $D_{TE}$ for the total exchange algorithm is

$$D_{TE} = \sum_{i=1}^{d} \frac{M_i + m_i}{2(n-1)} \binom{d}{i}$$

$$= \frac{1}{d(n-1)} \sum_{i=1}^{d} \binom{d}{i} \sum_{j=1}^{i} \binom{d}{j} j - \frac{1}{2d(n-1)} \frac{(2d-1)!}{[(d-1)!]^2}$$

$$+ \frac{2d - 1 + \left(\frac{n}{2} - 1\right) d}{2(n-1)}.$$

This formula gives the optimal average delay for the total exchange task when $d$ is prime, but does say tell anything about its order of magnitude. In the following lemma (which is true even if $d$ is not prime) we prove that $D_{TE}$ is actually $\Theta(n)$.

**Lemma 1.** *For all d the optimal average delay $D_{TE}$ of the total exchange task in $\Theta(n)$.*

**Proof.** It is easy to see that $D_{TE}$ is $O(n)$ since the optimal completion time $T_{TE}$ is equal to $n/2$ and $D_{TE} \leq T_{TE}$. It remains to show that $D_{TE} = \Omega(n)$.

Let $S_1$ be the set of $n/4$ routing tags with the larger number of ones that are cleared last in the optimal average delay algorithm. For each routing tag $t$ in $S_1$ with $k$ ones (except for $t = 11 \cdots 11$) there is a corresponding routing tag $\bar{t}$ (the bitwise complement of $t$) with $d - k$ ones. Let these corresponding routing tags be a set $S_2$ with cardinality $n/4 - 1$. Consider now the submatrix of the initial task matrix $T_0$ that correspond to routing tags in $S_1 \cup S_2$. The critical sum of this submatrix is equal to $\frac{n}{4}\frac{1}{d}d = \frac{n}{4}$. This is so because there are $n/4$ pairs of tags $t$ and $\bar{t}$ (we include the all 0 tag for completeness) and each pair has a total of $k + d - k = d$ bits equal to one. Let now $S_3$ be the set of routing tags of $T_0$ that do not belong to $S_1 \cup S_2$. The critical sum of the submatrix of $T_0$ that corresponds to $S_3$ is equal to $n/2 - n/4 = n/4$. This means that the routing tags of $S_3$ require at least $n/4$ steps to get cleared. Since in the optimal average delay algorithm the routing tags of $S_1$ are cleared after those of $S_3$, their delay will be at least $n/4$ steps. By taking into account that there are $n/4$ routing tags in $S_1$ and each of them has a delay greater than $n/4$, we conclude that the average delay $D_{TE}$ will be at least $n/16 = \Omega(n)$. □

## 6.3. *Case where d is not prime*

When $d$ is not a prime number, strict optimality with respect to the average delay criterion is not guaranteed. The reason is that when $d$ is not prime, some of the classes $R_{ij}$ have less than $d$ elements; we call such classes *degenerate.* However, in this case we can still find algorithms that complete the total exchange task in an optimum number of steps and achieve near-optimal average delay. In order to do so, we can first clear the nondegenerate classes by applying both the OCTR and the OADD and then clear the degenerate classes by just following the OCTR (relaxing the OADD). It can be shown (see e.g. [14], p. 14) that there are at most $O(n^{1/2})$ routing tags belonging to degenerate classes while there is a total of $n$ different routing tags. Thus, the loss of average delay optimality introduced by the degeneracy is negligible. In particular, since the delay of the packets that belong to degenerate classes is at most $n/2$ (because OCTR is followed), the loss in optimality can be upper-bounded by $O(n^{1/2})\frac{n}{2}\frac{1}{n} = O(n^{1/2})$. Since the optimal average delay is $\Theta(n)$ (Lemma 1), we conclude that the algorithm for $d$ not prime is near-optimal with respect to average delay. In addition, it is guaranteed to be optimal with respect to completion time since the OCTR is never violated.

An interesting question is whether by following only the OCTR rule, near-optimal average delay is guaranteed. The answer to this question is negative, and will be illustrated by two examples. In these examples we will assume for convenience that $d$ is prime. First, consider the following algorithm $\mathscr{V}$, which consists of two phases. In the first phase, the entries of the $d \times d$ submatrix that corresponds to each class $R_{ij}$, $i \neq d$, are cleared up to the point where there is only one non-zero entry per row and one non-zero entry per column in each such submatrix. The all-ones row is not cleared at all during the first phase. For $d$ prime, the first phase requires $n/2 - (n-2)/d - 1$ steps. Up to that time no packets have been delivered to their destination. In the second phase, the remaining entries are cleared in any way by using the OCTR rule. Then the OCTR rule is followed in both phases. The average delay of algorithm $\mathscr{V}$ satisfies $D^{\mathscr{V}} \geq n/2 - (n-2)/d$. Thus we have found an algorithm which follows the OCTR rule, and which has average delay greater than $n/2 - o(n)$. Consider now a second algorithm $\mathscr{M}$, where every tag $t$ is grouped together with its bitwise complement $\bar{t}$ (the all ones tag is grouped together with the all zero tag). There are $n/2$ such pairs. Algorithm $\mathscr{M}$ consists of $k = \lfloor n/(2d) \rfloor$ phases. Each of the first $k-1$ phases has duration $d$, while the last phase has duration $n/2 - kd + d$. In each of the first $k-1$ phases, we pick any $d$ distinct pairs $(t, \bar{t})$ at a time (i.e. $2d$ tags at a time), and clear them completely (in $d$ steps) before picking a new set of $d$ pairs. In the last phase we just follow the OCTR rule to clear the remaining entries. Algorithm $\mathscr{M}$ follows the OCTR at each step, and has average delay

$$D^{\mathscr{M}} \leq \frac{2d(d + 2d + \cdots + kd) + (n - 2kd + 2d)n/2}{n}$$

$$= \frac{d^2 k(k+1)}{n} + O(d) = n/4 + O(d).$$

This shows that the optimal average delay $D_{TE}$ is less than $n/4 + o(n)$. At the same time, the previous algorithm $\mathscr{V}$ followed the OCTR and had $D^{\mathscr{V}} \geq n/2 - o(n)$. Therefore, the OCTR does not guarantee near-optimal average delay, since the potential loss of optimality is of the same order of magnitude with the optimal average delay itself.

We should note here the preceding analysis of the joint optimization of the completion time and the average delay (and the average storage requirements at the nodes) can be extended to other isotropic tasks as well. As an example, in the $(K, L)$ neighborhood exchange problem simultaneously optimal completion time and near-optimal packet delay are achieved if the equivalence classes are cleared in the order $R_{K1} \ldots R_{Kn_K} \ldots R_{L1} \ldots R_{Ln_L}$. For general isotropic tasks, it is not difficult to find algorithms that achieve optimal completion time and near-optimal average delay by insisting on the OCTR and ignoring the OADD when both cannot be satisfied. A bias in the opposite direction will result in the opposite results.

## 7. Case where only $k$ links of each node are used

In this section we consider the case where due to hardware limitations the hypercube nodes can use only $k \leq d$ incident links during a slot. The analysis of this case follows the same lines with the case where a node can use all its incident links. For isotropic tasks the critical sum $h$ of the initial task matrix $T_0$ is again a lower bound of the completion time of a task (since $k \leq d$). We define the *total sum* $\sigma$ of a matrix $T_0$ to be equal to the sum of all the entries of $T_0$, i.e. $\sigma = \Sigma_{ij}(T_0)_{ij}$. Then if $\mathscr{T}$ is the completion time of an algorithm that executes the task it can be seen that

$$\mathscr{T} \geq \max\left\{\left\lceil \frac{\sigma}{k} \right\rceil, h\right\}.$$

We defined a *k-permutation matrix* to be a permutation matrix with at most $k$ nonzero entries. By restricting attention to symmetric routings that use $k$-permutation matrices as switching assignments and transforming the scheduling problem to a matrix decomposition problem, we can prove the following theorem.

**Theorem 7.** *Let $T_0$ be the $p \times d$ initial task matrix of an isotropic task. The optimal completion time of the task when each node can use up to $k \le d$ links during a slot, is equal to*

$$\max\left\{\left\lceil \frac{\sigma}{k} \right\rceil, h\right\},$$

*where $h$ and $\sigma$ are the critical and the total sum of $T_0$, respectively.*

**Proof** (abbreviated). We will show that a matrix $T_0$ with critical sum $h$ and total sum $\sigma$ can be written as the sum of $S := \max\{\lceil \frac{\sigma}{k} \rceil, h\}$ $k$-permutation matrices. We assume that $p > k$, otherwise the proof is trivial. We add to any entry $(i, j)$ of $T_0$ the quantity

$$\min(S - r_i, S - c_j, kS - \sigma),$$

updating in the meantime $\sigma$, $r_i$, and $c_j$. This process terminates when the new matrix $T_0'$ has total sum $\sigma' = kS$, and its critical sum is less than or equal to $S$. We add $p - k$ extra columns, and $d - k$ extra rows to $T_0'$ to form a $(p + d - k) \times (p + d - k)$ matrix $T_0''$, so that the $(d - k) \times (p - k)$ added submatrix (call it $C$), contains only zeros, and every line sum of $T_0'$ is equal to $S$ (it can be proved as in Theorem 3 that this is always possible). During this procedure we do not add any numbers to entries of $T_0'$, but only to the new rows and columns. Then $T_0''$ can be written as the sum of $S$ $k(p + d - k) \times (p + d - k)$ permutation matrices. Each permutation matrix must contain an entry from each extra column and an entry from each extra row, and these entries must be different (because $C$ has only zeros). Therefore, the remaining $(p + d - k) - (d - k) - (p - k) = k$ entries must be from $T_0'$. This proves that $T_0'$, and, therefore, $T_0$ can be written as the sum of $S$ $k$-permutation matrices. □

For $1 < k < d$, the above estimate has not been derived elsewhere. It is worth noting that if $\lceil \frac{\sigma}{k} \rceil \le h$, then using only $k \le d$ links of a node at each slot rather than $d$, does not increase the time required to complete the task. For the total exchange task it can be seen that $\sigma = nd/2$ and the optimal completion time when up to $k$ links per node are used is equal to

$$T_{TE}^{(k)} = \left\lceil \frac{n}{2} \frac{d}{k} \right\rceil.$$

## 8. Optimal algorithms for isotropic tasks in wraparound meshes

In this section we focus on a class of communication tasks (which we will call isotropic again) in the $d$-dimensional wraparound mesh of processors. The performance criteria for various algorithms will be the completion time and the average packet delay. All the algorithms to be given for the wraparound mesh are optimal with respect to completion time. For the total exchange problem, we find algorithms that simultaneously achieve optimal completion time and near-optimal average delay. For the two-dimensional and one-dimensional wraparound meshes, we present algorithms which achieve strict optimality for both performance criteria. Finally, we prove that a greedy switching scheme in a $d$-dimensional wraparound mesh executes isotropic tasks in near-optimal time.

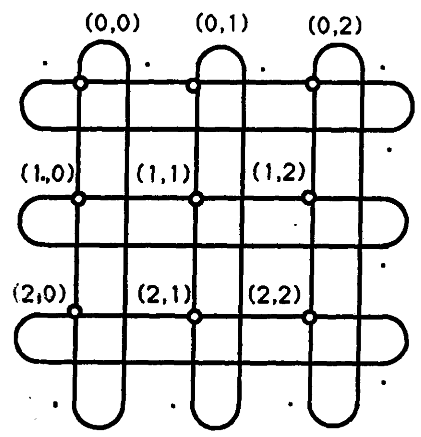


Fig. 5. A two-dimensional wraparound mesh with $n = 9$ processors.

## *8.1. d-dimensional wraparound mesh and task matrices*

The $d$-dimensional wraparound mesh consists of $n$ processors arranged along the points of a $d$-dimensional space that have integer coordinates. We assume that there is an equal number

$$p = n^{1/d}$$

along each dimension (although this is not essential). The processors along the $i$th dimension are obtained by fixing coordinates $x_1, \dots, x_{i-1}, x_{i+1}, \dots x_d$, and have identities $(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_d)$, $x_i = 0, \dots, p-1$. Two processors $(x_1, \dots, x_i, \dots, x_d)$ and $(y_1, \dots, y_i, \dots, y_d)$ are connected by a two-directional link, if and only if for some $i$ we have $|x_i - y_i| = 1$ and $x_j = y_j$ for all $j \neq i$. In addition to these links, in the wraparound mesh, there are also the two-directional links of the form

$$((x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_d), (x_1, \dots, x_{i-1}, p-1, x_{i+1}, \dots, y_d)).$$

A two-dimensional wraparound mesh is illustrated in *Fig. 5.* For an integer vector $x = (x_1, \dots, x_d)$, we use the notation

$$x \bmod(p) = (x_1 \bmod(p), \dots, x_d \bmod(p)).$$

We will be interested in isotropic tasks which are defined as follows.

**Definition 7.** A communication task in a $d$-dimensional wraparound mesh is *isotropic* if:
(a) When processor $x$ has to send $m$ packets to processor $y$, processor $x$ also has to send $m$ packets to processor $[2x - y] \bmod(p)$ and
(b) For each packet that processor $x$ has to send to processor $y$, every other processor $z$ has to send a packet (called *corresponding* packet) to processor $[z + y - x] \bmod(p)$.

Part (a) in the above definition specifies that the transmission requirements of a node are the same in both directions along each dimension. Part (b) specifies that the transmission requirements are the same for all nodes. An example of an isotropic task is the total exchange. The following definitions will be necessary for our analysis.

**Definition 8.** The $i^+$ link of a node $x = (x_1, \dots, x_i, \dots, x_d)$ is the link connecting $x$ with node $(x + e_i) \bmod(p)$, where $e_i$ is the unit vector whose $i$th entry is equal to one. Similarly, the $i^-$ link of a node $x$ is the link connecting $x$ with node $(x - e_i) \bmod(p)$.

**Definition 9.** The *distance along the* $i^+$ *dimension* between processors $x=(x_1,\ldots,x_i,\ldots,x_d)$ and $y=(y_1,\ldots,y_i,\ldots,y_d)$ is defined as

$$P_i(x,y)=\begin{cases} y_i-x_i, & \text{if } y_i\geq x_i;\\ y_i-x_i+p, & \text{otherwise.}\end{cases}$$

Similarly the *distance along the* $i^-$ *dimension* is defined as

$$N_i(x,y)=\begin{cases} x_i-y_i, & \text{if } y_i\leq x_i;\\ x_i-y_i+p, & \text{otherwise.}\end{cases}$$

Let us also introduce some new notation. For $p$ even consider the set of packets $a$ originating at node 0 with destinations $f(a)$ satisfying $P_i(0,f(a))=p/2$,

$$A_i=\{a \mid P_i(0,f(a))=p/2\}.$$

For $p$ odd let $A_i$ be the empty set. We arbitrarily partition $A_i$ into two disjoint subsets $A_i^+$ and $A_i^-$ such that $A_i=A_i^+\cup A_i^-$ and $0\leq |A_i^+|-|A_i^-|\leq 1$.

The *initial task matrix* in the context of the wraparound mesh is an $n_p\times(2d)$ matrix, where $n_p$ is the number of packets that each node has to send and $d$ is the dimension of the mesh (so that $2d$ is the number of outgoing links from each node). Each column of the task matrix corresponds to an outgoing link in the order $1^+, 1^-, 2^+, 2^-,\ldots,i^+, i^-,\ldots,d^+, d^-$. Let $a$ be the packet that originates at node 0 and corresponds to row $R$, and let $f(a)$ be the packet's destination. Then the $i^+$th and the $i^-$th entry of $R$ are given by

$$R_{i^+}=\begin{cases} P_i(0,f(a)), & \text{if } P_i(0,f(a))<N_i(0,f(a)) \text{ or } a\in A_i^+,\\ 0 & \text{otherwise,}\end{cases}$$

and

$$R_{i^-}=\begin{cases} N_i(0,f(a)), & \text{if } N_i(0,f(a))<P_i(0,f(a)) \text{ or } a\in A_i^-,\\ 0 & \text{otherwise,}\end{cases}$$

| | | | | |
|---|---|---|---|---|
| | 0 | 2 | 0 | 2 |
| | 0 | 1 | 0 | 2 |
| | 1 | 0 | 2 | 0 |
| | 0 | 2 | 0 | 1 |
| | 2 | 0 | 1 | 0 |
| | 0 | 0 | 2 | 0 |
| | 2 | 0 | 0 | 0 |
| | 0 | 1 | 0 | 1 |
| | 0 | 1 | 1 | 0 |
| | 1 | 0 | 0 | 1 |
| | 1 | 0 | 1 | 0 |
| | 0 | 0 | 0 | 1 |
| | 0 | 0 | 1 | 0 |
| | 0 | 1 | 0 | 0 |
| | 1 | 0 | 0 | 0 |
| LINK: | 1+ | 1- | 2+ | 2- |

Fig. 6. The task matrix for the total exchange problem in a two-dimensional wraparound mesh with $n=16$.

respectively. The row $R$ is also the initial routing tag of packet $a$. Routing tags carried by packets in the course of algorithms are defined analogously, with node 0 in the preceding equation replaced by the node where the packet resides. The task matrix for the total exchange task in a two-dimensional wraparound mesh with $n = 16$ is shown in *Fig. 6.*

A difference between the task matrix in a hypercube and the task matrix in a mesh is that the entries of the latter are not necessarily binary, but can take any value between 0 and $\lfloor p/2 \rfloor$.

As in the case of the hypercube, we will restrict attention to symmetric switching schemes. A symmetric switching scheme is characterized by the property that if processor 0 sends a packet with some routing tag over its $i^+$ (or $i^-$) link, then during the same slot every other processor sends over its $i^+$ (respectively $i^-$) link a packet with the same routing tag. For symmetric switching schemes, it can be shown that executing an isotropic task is equivalent to a matrix decomposition problem. The proofs of the following theorems follow similarly as in the hypercube case and are omitted.

**Theorem 8.** *The optimal time required to execute an isotropic task is equal to the minimum number of steps required to clear its task matrix, i.e. to make all its entries equal to zero. At each step, one is allowed to decrement by one unit independent entries of the task matrix, that is, entries that do not belong to the same line.*

**Theorem 9.** *The optimal time required to execute an isotropic task is equal to the critical sum of its task matrix.*

A symmetric switching scheme achieves optimal completion time if and only if it follows the following rule.

*Optimal Completion Time Rule (OCTR):*

At every step, an entry from each critical line of the task matrix is decreased by one.

It can be shown that it is always possible to follow the OCTR. In the following subsection this rule will be used to obtain optimal algorithms for the total exchange problem.

## *8.2. Optimal completion time algorithms for the total exchange in wraparound meshes*

In order to calculate the optimal number of steps required for a total exchange we simply have to calculate the critical sum of the initial task matrix. For a wraparound mesh with odd $p$ the critical sum of, say link $d+$, can be calculated to be

$$T_{TE} = \sum_{j_1=-(p-1)/2}^{(p-1)/2} \sum_{j_2=-(p-1)/2}^{(p-1)/2} \cdots \sum_{j_d=0}^{(p-1)/2} j_d. \tag{4}$$

To see this, note that for any valuc $j_d$ of the entry $R_{d+}$, the entries $R_{i+}$, and $R_{i-}$ can take all the values between 0 and $(p-1)/2$, or, equivalently, the index $j_i$ in equation (4) can take all the values between $-(p-1)/2$ and $(p-1)/2$. Equation (4) gives

$$T_{TE} = \sum_{j_1=(p-1)/2}^{(p-1)/2} \cdots \sum_{j_{d-1}=-(p-1)/2}^{(p-1)/2} \frac{(p-1)(p+1)}{8} = \frac{p^2-1}{8} p^{d-1} = \frac{pn - n/p}{8},$$

where we have used the relation $n = p^d$.

If $p$ is even, we can similarly find that

$$T_{TE} = \frac{pn}{8}, \quad \text{if } d > 1,$$

and for the ring case

$$T_{TE} = \frac{n(n+2)}{8}, \quad \text{if } d = 1.$$

Note that the case $p = 2$ corresponds to a hypercube having *two* links in each direction between any pair of nodes whose identities differ by one bit. In this case the TE can be executed in time $n/4$. This can be seen by splitting the task matrix in two parts, each with critical sum in/4, and having one of the two sets of links clear the one matrix, and the other set of links simultaneously clear the second matrix.

## 8.3. Near-optimality of greedy algorithms

Let $h_c$ and $h_r$ be the maximum column and row sum of the initial task matrix. Then by using arguments similar to those presented in Section 5, we can show that any greedy algorithm executes the task in at most

$$h_r + h_c - 1$$

steps. Thus any greedy algorithm is suboptimal by at most $h_r$ steps ($h_r \leq (p-1)d/2$ for $p$ odd and $h_r \leq pd/2$ for $p$ even). In particular, for the total exchange task any greedy algorithm achieves a completion time of at most

$$\frac{pn - n/p}{8} + \frac{(p-1)d}{2}$$

for $p$ odd and

$$\frac{np}{8} + \frac{pd}{2}$$

for $p$ even. As an example, if $n = 1000$ and $d = 3$, then an optimal algorithm will take 1250 steps and any greedy algorithm will take at most 1265 steps. The conclusion is that any greedy algorithm is very close to being optimal.

## 8.4. Algorithms with optimal completion time and near-optimal average delay for the total exchange

If the OCTR is followed by an algorithm, this algorithm is guaranteed to have optimal completion time. With this rule, packets arrive to their destinations over shortest paths. In the algorithms that we propose for the total exchange problem, this rule is always followed. It is not difficult to see that for the total exchange 100% utilization of the links is achieved. If in addition to following the OCTR it is assured that the packets which are transmitted at each slot are those that are nearer to their destinations at that slot, then the average delay of a packet is also optimal. This priority rule is captured by the Optimal Average Delay Directive which was introduced in Section 6.

We will now present an algorithm for the total exchange in a $d$-dimensional wraparound mesh in which the OCTR is always followed and the OADD is partially followed. In the next subsection, algorithms that follow both rules and achieve both kinds of optimality will be given for the case of the two-dimensional and the one-dimensional wraparound mesh.

Instead of describing the algorithm in terms of transmissions over the links, we will equivalently describe the order in which the entries of the initial task matrix are cleared. Before proceeding it is necessary to describe some new concepts.

The $n-1$ routing tags of the packets sent by a node are partitioned into disjoint sets $N_i$, $i = 1, 2, \ldots$ The set $N_i$ contains all the routing tags with sum of entries equal to $i$ and corresponds to packets that are initially located at a distance of $i$ hops from their destination. In the algorithms that we will propose, the rows of the task matrix which belong to the set $N_i$ are cleared during phase $i$ of the algorithm and the OCTR is followed. Phase $i$ begins after phase $i-1$ has finished. This is in accordance with the OADD.

Consider the rows of the initial task matrix which correspond to the packets in $N_i$. These rows form a submatrix of $T_0$ which we denote by $M_i$. For the algorithm to work it is enough to prove that each $M_i$ can be separately cleared while simultaneously attaining 100% utilization of the links. Consider the case where $p$ is odd. If we denote with $|N_i|$ the cardinality of $N_i$ and by $c_i$ the column sums of $M_i$ (the column sums are equal for all columns by symmetry), then $i|N_i| = 2dc_i$, from which $c_i = |N_i|i/2d$. Since $|N_i| \geq 2d$ [because at least the packets with routing tags $(i, 0, \ldots, 0)$, $(0, i, 0 \ldots, 0)$, $(0, 0, i, 0, \ldots, )$, $(0, 0, 0, i, 0 \ldots, 0), \ldots, (0, 0, \ldots, i, 0)$, $(0, 0, \ldots, 0, i)$ belong to $N_i$], we conclude that $c_i \geq i$ and therefore the critical sum of $M_i$ is $c_i$ (obviously, the row sums of $M_i$ are equal to $i$). This proves that $M_i$ can always be cleared in $c_i$ steps by following the OCTR. Since the sum of all the entries of the matrix $M_i$ is $2dc_i$, at each step exactly $2d$ entries are cleared (because no more than $2d$ entries can be decremented at the same step). As a consequence, all the outgoing links of a node are always used. By clearing the submatrices $M_1, M_2, \ldots, M_i, \ldots$ in this order, we follow the OCTR and at the same time the packets that are initially $i$ hops away from their destination are received by the destination node before the packets that belong to sets $N_{i+1}, N_{i+2}, \ldots$ start being transmitted.

This algorithm is expected to have near-optimal average delay and optimal completion time. However, strict optimality of the average delay is not necessarily achieved. The reason is that the OADD is not followed *within* each phase. Suppose now that the submatrices $M_i$ can be subdivided into square matrices $M_{i1}, M_{i2}, \ldots, M_{ij}, \ldots, M_{i(|N_i|/2d)}$ such that

$$M_i = \begin{pmatrix} M_{i1} \\ \vdots \\ M_{i(|N_i|/2d)} \end{pmatrix}.$$

Suppose also that every submatrix $M_{ij}$ can be completely cleared before starting to clear $M_{i(j+1)}$, and while clearing $M_{ij}$ 100% utilization of the links is achieved. In such a case, simultaneously optimal completion time and average delay are guaranteed. We could not prove that this is possible for every dimension $d$. However, this is possible for $d$ equal to 1 and 2, so in these cases algorithms that achieve simultaneously optimal completion time and average delay can be found. These algorithms are the subject of the following subsection.

## *8.5. Optimal completion time/optimal average delay algorithms for the total exchange in wraparound meshes with $d = 1$ and $d = 2$*

Consider first a total exchange in the two-dimensional wraparound mesh. We partition the $n-1$ routing tags of the packets sent by a node, in sets $N_i$ as described in the previous subsection. Each set $N_i$ is in turn partitioned in classes $R_{ab}$, where $a + b = i$, $a > 0$ and $b \geq 0$. The rows of $T_0$ that correspond to class $R_{ab}$ are $(a, 0, b, 0)$, $(0, b, a, 0)$, $(0, a, 0, b)$, $(b, 0, 0, a)$. Because the classes $R_{i0}$ and $R_{0i}$ would coincide, we introduced the restriction $a$  0 so that only one of them is considered. The submatrix of the initial task matrix whose rows correspond to class $R_{ab}$ has dimensions $4 \times 4$ and is denoted $M_{ab}$. Each submatrix $M_{ab}$ can be cleared in exactly $i = a + b$ steps with 100% link utilization (see *Fig. 7*). If the submatrices are cleared in the order

$$M_{10}, M_{20}, M_{11}, M_{30}, M_{21}, M_{12}, \ldots, M_{i0}, M_{(i-1)1}, M_{1(i-1)}, \ldots$$

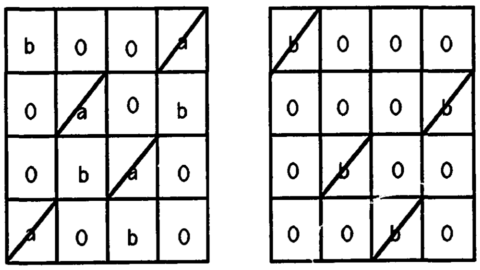


Fig. 7. Clearing the submatrix corresponding to class $R_{ab}$.

it can be seen that strictly optimal average delay, as well as optimal completion time are achieved, because both the OCTR and OADD are followed at each step.

We next consider the total exchange in a one-dimensional wraparound mesh (ring). The initial task matrix for the ring has dimensions $(n-1) \times 2$. Algorithms that simultaneously achieve strictly optimal average delay and completion time can be found using the principles described earlier. We omit the details and just give the results. If $T_{TE}$ is the optimal completion time of the total exchange and $D_{TE}$ is the optimal average delay we can show that

$$T_{TE} = \frac{n^2-1}{8}$$

$$D_{TE} = \frac{(n+1)(n+3)}{24}$$

for an odd number of processors $n$, and

$$T_{TE} = \frac{n(n+2)}{8}$$

$$D_{TE} = \frac{(n-2)(n-1)n+6n^2}{24(n-1)}$$

for an even number of processors. Note that for the ring and for a large $n$, the average delay is roughly one-third of the completion time for the total exchange. Most of the results of Section 8 can be extended to wraparound meshes with different number of processors in each dimension.

## Appendix

In this appendix we prove Theorem 6 of Section 6. For convenience we restate the theorem.

**Theorem 6.** *Consider a network and an arbitrary communication task. An algorithm that executes the task and in which*
(a) *packets are sent to their destinations over shortest paths,*
(b) *all links are used at all time slots prior to the algorithm's termination (100% utilization) and at every slot packets are transmitted according to the OADD,*
*is optimal with respect to the average delay criterion.*

**Proof.** Let $L = nd$ be the number of links of the network and $N$ be the number of packets involved in the communication task. Let $x_i$ be the minimum number of hops between the origin and the destination of packet $i$. For an algorithm that performs the communication task we denote by $W_i$ the time that elapses from the beginning of the algorithm to the slot when packet $i$ arrives at its destination. Let $C_{\text{opt}}$ be the optimal value of $\sum_{i=1}^{N} W_i$.

We consider the following auxiliary problem.

*Auxiliary problem*

Assume we are given $L$ servers and $N$ customers that have to be served. Assume also that customer $i$ requires at least $x_i$ slots of service ($x_i$ is integer) and can be served by at most one server during the same slot. Each customer can use different servers in different slots and each server can serve at most one customer per slot. Consider a schedule that assigns customers to server-slot pairs and let $W_i$ be the time that elapses between the beginning of the schedule and the slot when customer $i$ completes service. Find a schedule that minimizes $\sum_{i=1}^{N} W_i$.

Let $\hat{C}_{\text{opt}}$ be the optimal value of $\sum_{i=1}^{N} W_i$ in the auxiliary problem. It can be seen that $\hat{C}_{\text{opt}} \le C_{\text{opt}}$. The reason is that the auxiliary optimization problem has less constraints than the initial problem. (If we regard links as servers and packets as customers then the first problem has additional constraints on the servers that can be used by each customer and the order in which the servers are used. In particular the links used by the packets depend on the packet's destinations. Thus, for every algorithm $\mathscr{A}$ in the initial problem we can find a corresponding feasible schedule $\hat{\mathscr{A}}$ in the auxiliary problem which has the same cost.)

Consider an algorithm $\mathscr{A}$ for the initial problem which follows the OADD. Let $u(t)$ be the number of links that are used at slot $t$. It can be seen that for such an algorithm, $u(t)$, $t = 1, 2, \ldots,$ uniquely specifies the cost $\sum_{i=1}^{N} W_i$.

Consider now a schedule $\hat{\mathscr{A}}$ in the auxiliary problem with the property that at each slot the customers that are served are those which have the least residual time to complete service. Let $\hat{u}(t)$ be the number of servers that are used at time $t = 1, 2, \ldots$ It can be seen that if $u(t) = \hat{u}(t)$ for all $t$, then both $\mathscr{A}$ and $\hat{\mathscr{A}}$ have the same cost $\sum_{i=1}^{N} W_i$. Thus, if $\hat{\mathscr{A}}$ is optimal with respect to the average delay criterion, then $\mathscr{A}$ is an optimal algorithm in the initial problem since it achieves the same cost and we have already seen that $\hat{C}_{\text{opt}} \le C_{\text{opt}}$. Therefore, in order to prove Theorem 6 it is enough to prove the following proposition.

**Proposition 1.** *Let $\hat{\mathscr{A}}$ be a schedule in the auxiliary problem with the following properties:*

(a) *Item $i$ is served during exactly $x_i$ slots (i.e. the minimum adequate service time).*

(b) *At each slot the customers that are served are those that have least residual times to complete service.*

(c) *The number of servers used at time $t$, $\hat{u}(t)$, is the largest possible, given the assignments made at slots $1, 2, \ldots, t-1$.*

*Then $\hat{\mathscr{A}}$ minimizes $\sum_{i=1}^{N} W_i$.*

**Proof.** It can be seen that $\hat{\mathscr{A}}$ is a Shortest Processing Time (SPT) schedule, which is known to minimize the average delay $\sum_{i=1}^{N} W_i/N$ (see, e.g. [6]). □

## References

[1] G. Bongiovanni, D. Coppersmith and C.W. Wong, An optimum time slot assignment algorithm for an SS/TDMA system with variable number of transponders, *IEEE Trans. Commun.* COM-29 (1981) 721–726.

[2] Bertsekas, D.P., *Linear Network Optimization: Algorithms and Codes* (MIT Press, Cambridge, MA, 1991).

[3] Bertsekas, D.P., and Tsitsiklis, J.N., *Parallel and Distributed Computation: Numerical Methods,* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[4] Bertsekas, D.P., C. Ozveren, G.D. Stamoulis, P. Tseng, and Tsitsiklis, J.N., Optimal communication algorithms for hypercubes, Lab. for Information and Decision Systems Report LIDS-P-1847, M.I.T., Jan. 1989; also in *J. Parallel and Distributed Comput.* 11 (1991) 263–275.

[5] Bhatt, S.N., and Ipsen, I.C.F., How to embed trees in hypercubes, Yale University, Dept. of Computer Science, Research Report YALEU/DCS/RR-443, 1985.

[6] Coffman, E.G., *Computer and Job-Shop Scheduling Theory* (Wiley, New York, 1976).

[7] Dally, W.J., and Seitz, C.L., Deadlock-free message routing in multiprocessor interconnection networks, *IEEE Trans. Comput.* C-36 (1987) 547–553.

[8] Dekel, E., Nassimi, D., and Sahni, S., Parallel matrix and graph algorithms, *SIAM J. Comput.* 10 (1981) 657–673.

[9] Hedetniemi, S.M., Hedetniemi, S.T., and Liestman, A.L., A survey of gossiping and broadcasting in communication networks, *Networks* 18 (1988) 319–349.

[10] Johnsson, S.L., Communication efficient basic linear algebra computations on hypercube architectures, *J. Parallel and Distributed Comput.* 4 (1987) 133–172.

[11] Johnsson, S.L., and Ho, C.T., Optimum broadcasting and personalized communication in hypercubes, *IEEE Trans. Comput.* 38 (1989) 1249–1268.

[12] Kermani, P., and Kleinrock, L., Virtual cut-through: A new computer communicating switching technique, *Comput. Networks* 3 (1979) 267–286.

[13] Krumme, D.W., Venkataraman, K.N., and Cybenko, G., The token exchange problem, Tufts University, Technical Report 88-2, 1988.

[14] Leighton, F.T., *Complexity Issues in VLSI* (MIT Press, Cambridge MA, 1983).

[15] McBryan, O.A., and Van de Velde, E.F., Hypercube algorithms and their implementations, *SIAM J. Sci. Stat. Comput.* 8 (1987) 227–287.

[16] Ozveren, C., Communication aspects of parallel processing, Laboratory for Information and Decision Systems Report LIDS-P-1721, M.I.T., Cambridge, MA, 1987.

[17] Ryser, H.J., *Combinatorial Mathematics* (Mathematical Association of America, Rahway, NJ, 1963).

[18] Saad, Y., and Schultz, M.H., Data communication in hypercubes, Yale University Research Report YALEU/DCS/RR-428, October 1985 (revision of August 1987).

[19] Saad, Y., and Schultz, M.H., Data communication in parallel architectures, Yale University Report, March 1986.

[20] Saad, Y., and Schultz, M.H., Topological properties of hypercubes, *IEEE Trans. Comput.* 37 (1988) 867–872.

[21] Stamoulis, G.D., and Tsitsiklis, J., Efficient routing schemes for multiple broadcasts in hypercubes, Laboratory for Information and Decision Systems, Report LIDS-P-1948, February 1990.

[22] Stout, Q.F., and Wagar, B., Passing messages in link-bound hypercubes, in *Proc. 1986 Hypercube Conf. SIAM*, Philadelphia, PA (1987) 251–257.

[23] Topkis, D.M., Concurrent broadcast for information dissemination, *IEEE Trans. Software Engrg.* 13 (1983) 207–231.

[24] Wang, E.Y., Traffic control in a multichannel optical fiber communication network, MS. Thesis, Lab. for Information and Decision Systems Report LIDS-P-1784, MIT, 1988.