Technical Report No. 97-415

Novel Data Communication Algorithms on Hypercubes and Related Interconnection Networks and Their Applications in Computational Geometry*

Ke Qiu	Selim G. Akl
School of Computer Science	Dept. of Comp. and Info. Sci.
Acadia University	Queen's University
Wolfville, Nova Scotia	Kingston, Ontario
Canada	Canada

Abstract

We present several novel data communication algorithms for hypercubes. Specifically, we obtain (1) an algorithm that broadcasts m messages of unit size on a hypercube of size N in optimal time $O(m + \log N)$; and (2) algorithms for special cases of computing m prefix sums, also in optimal time $O(m + \log N)$. Unlike previous algorithms for performing similar tasks, our schemes require no use of pipelining. They can be implemented using the standard ASCEND/DESCEND strategy commonly used for hypercubes, making their implementations much easier. Moreover, while previous pipelined algorithms require that we know exact embeddings of binary trees into hypercubes, our algorithms use recursive properties of the hypercube. Because of this, our schemes can be easily implemented directly on other similar interconnection networks such as stars and pancakes (both members of the family of recursively decomposable Cayley graphs, to which the hypercube also belongs) without first having to find embeddings of tree-like structures of constant degree.

To demonstrate the applications of our data communication algorithms, we use them to solve several problems in computational geometry. In particular, we present two parallel algorithms that run in $O(m + \log N)$ time, where N is the network size. The first algorithm locates m planar points in a simple polygon with N vertices. This algorithm works on the hypercube, the star, and the pancake interconnection networks. To our knowledge, this algorithm is the first for the star and pancake to achieve this performance. For the hypercube, the running time of this algorithm matches that of a previous algorithm (designed for a binary tree) that uses pipelining [3]. The second algorithm locates m planar points in a planar subdivision with $O(N)$ vertices. This algorithm is for the hypercube only. When $m = o(\log^3 N)$, it is better than the algorithm by Lee and Preparata [11], designed to locate $O(N)$ points in $O(\log^3 N)$ time

*Work supported by the Natural Sciences and Engineering Research Council of Canada (NSERC)

on a cube-connected-cycles network. In addition, both algorithms are simpler than previous ones in that they are direct parallelizations of a straightforward sequential algorithm described in [15], p. 42.

Keywords: parallel algorithms, hypercube, star, pancake, interconnection networks, pipelining, computational geometry, data communication.

1 Introduction

Consider an interconnection network parallel computer (such as the hypercube) in which each processor stores several data items. Many algorithms for such a network are based on a pipelining approach. The latter typically implies that an embedding of another network (such as a tree) into the network at hand (the hypercube) is known. The resulting algorithms are usually complicated to implement and involve a considerable amount of computational overhead. In addition, by using special embeddings and pipelining, some of the properties of the underlying interconnection network are not used advantageously. This is the case when the network is the hypercube, one of the most popular networks for interconnecting processors in a parallel computer. In this paper, we develop several data communication algorithms for the hypercube and some other recursively decomposable Cayley graphs [8] without resorting to pipelining. These algorithms exploit the standard ASCEND/DESCEND schemes which are common for hypercube-like networks. The implementations are simple and their performances match those of previous algorithms that invoke embeddings and pipelining. More significantly, these algorithms can be easily applied to similar interconnection networks without having to find some embeddings first in order to use the technique of pipelining.

In addition, we will apply the data communication algorithms developed here to obtain several parallel algorithms on hypercubes and other networks for some problems in computational geometry: locating multiple points in a simple polygon and in a planar subdivision. These algorithms' performances either match those of previous ones or surpass them in some cases.

To better illustrate our ideas, we first design all of our algorithms on hypercubes. The ideas are then extended to other networks. As a result, we organize our paper as follows. We define the hypercube and briefly describe some of its properties in Section 2. Section 3 presents our data communication algorithms on hypercubes. Our planar point location algorithm for simple polygons is given in Section 4 while Section 5 gives an algorithm for locating multiple planar points in a subdivision. Section 6 discusses extension of our results to other interconnection networks, and finally, concluding remarks are offered in Section 7. Some of the material in this paper appears in [18] and is included here for completeness.

2 Hypercubes

A *hypercube of dimension* n, denoted Q_n, with N processors P_0, P_1, ..., P_{N-1}, where $N = 2^n$, is defined as follows. Let $i_{n-1}i_{n-2}\cdots i_{j+1}i_j i_{j-1}\cdots i_1 i_0$ be i's binary representation. Processor $P_{i_{n-1}i_{n-2}\cdots i_{j+1}i_j i_{j-1}\cdots i_1 i_0}$ is connected to processor $P_{i_{n-1}i_{n-2}\cdots i_{j+1}\bar{i}_j i_{j-1}\cdots i_1 i_0}$ along dimension j for

$0 \leq j < n$, where $\bar{i_j}$ is the complement of i_j. Therefore, each processor has n neighbors along dimensions 0, 1, ..., $n-1$. The n-dimensional hypercube is also referred to as an n-cube.

One of the many important properties of the n-cube is that it can be constructed recursively from lower dimensional cubes. More precisely, consider two identical $(n-1)$-cubes whose vertices are numbered likewise from 0 to $2^{n-1}-1$. By joining every vertex of the first $(n-1)$-cube to the vertex of the second having the same number, one obtains an n-cube. Conversely, separating an n-cube into the subgraph of all the processors whose leading bit is 0 and the subgraph of all the processors whose leading bit is 1, the two subgraphs are such that each node of the first is connected to one node of the second along dimension $n-1$. If we remove the edges between these two graphs, we get two disjoint $(n-1)$-cubes. For convenience, we call the first Q_{n-1} the left subcube, and the second the right subcube. In this case, we say that the n-cube has been decomposed into two Q_{n-1}'s. Note that the splitting suggested above gives privilege to the leading bit but there is no particular reason for this. Since an n-cube has n dimensions, the splitting can be done along any of these n dimensions.

Assume that each of N data items, $N = 2^n$, is stored in a processor of an n-cube, i.e., data t_0, t_1, ..., t_{2^n-1} are stored in processors P_0, P_1, ..., P_{2^n-1}, respectively. A hypercube algorithm is in the DESCEND class if it performs a sequence of basic operations on pairs of data located in pairs of processors whose binary indices differ successively by 2^{n-1}, 2^{n-2}, ..., 2^0. In the dual class ASCEND, an algorithm performs a sequence of basic operations on pairs of data located in pairs of processors whose binary indices differ successively by 2^0, 2^1, ..., 2^{n-1}. Many hypercube algorithms fall directly or indirectly into these two classes [14], while others consist of a sequence of algorithms in these two classes.

3 Data Communication Algorithms

An interconnection network is classified as either a weak model or a strong model, depending on how a processor communicates with its neighbors. In a weak model, in one time unit, a processor can send (receive) at most one datum of fixed length to (from) one and only one of the processors to which it is directly connected, i.e., each processor uses a single-port communication mode. On the other hand, in a strong model, in one time unit, a processor can send (receive) one datum of fixed length to (from) all the processors to which it is directly connected, i.e., each processor uses an all-port communication mode. All the algorithms in this paper are based on weak models.

For some positive integer m, we consider the following two fundamental data communication problems on an n-dimensional hypercube Q_n with processors P_0, P_1, ..., P_{N-1}, where $N = 2^n$:

1. broadcasting m messages of unit length from one processor to all processors in the hypercube; and

2. computing m sums $\sum_{i=0}^{N-1} x_{ij}$, for all $1 \leq j \leq m$, and storing these values in all the processors, assuming that each processor P_i, $0 \leq i \leq N-1$, stores m values x_{ij}, $1 \leq j \leq m$.

The first problem is also equivalent to sending one message of size m from one processor to all other processors. Clearly, a lower bound for this problem in the weak model is $\Omega(m+\log N)$ since the originating processor can send only one datum of fixed length at a time, and $n = \log N$ is the diameter of Q_n. By vertex symmetry of the hypercube, without loss of generality, we will assume that the originating processor wanting to broadcast messages is always P_0.

The standard algorithm that broadcasts one message of unit length on Q_n takes $O(\log N)$ time, and uses the standard ASCEND/DESCEND strategy [14]. See, for example, Program 3.1 of [10]. For m messages, one could trivially apply this algorithm m times and obtain a total running time of $O(m \log N)$. This is precisely what is done in [10], for example, to broadcast m messages on a hypercube (or, to use the terminology of [10], a single message of length m, i.e., a message consisting of m constant-size components).

Using the technique of pipelining, this task can be accomplished in the following two ways:

It is known that two complete binary trees each with $N/4$ leaves and whose roots are connected by a path of length 3 can be embedded in a hypercube of size N (see, for example, [12], pp. 406-407). Therefore, one way to broadcast m messages on a hypercube is as follows: the originating processor (one of the two roots) sends the m messages to the other root in $O(m)$ steps; the two roots now send the m messages to their descendents in a pipelined fashion in $O(m + \log N)$ time.

The second approach is to consider the problem of broadcasting m messages as a special case of the problem of performing m distinct prefix sums computations. Initially, processor P_0 contains messages M_1, M_2, ..., M_m, which are treated as values used in prefix sums while each of the other processors contains m 0 values. The binary associative operator is the exclusive OR. At the end of the prefix computation, each processor contains all messages. It is shown in [13] that m prefix sums can be computed on a hypercube of size N in $O(m+\log N)$ time by using the technique of pipelining. This approach also uses the fact that a binary tree can be embedded into a hypercube.

A disadvantage of both algorithms is that they use pipelining and thus more complicated control is required. In addition, neither can be implemented by the standard ASCEND/DESCEND scheme on hypercubes. Furthermore, in order to implement the pipelining technique, an embedding of a certain binary tree into the hypercube needs to be known. If we are to find a broadcasting algorithm for multiple messages on another interconnection network, these two algorithms are not readily implementable, unless we can find some embedding of graphs of constant degree, which is not always guaranteed.

In the following, we present a novel, yet very simple algorithm that takes $O(m + \log N)$ time for m messages and does not use pipelining. They can also be implemented on certain other interconnection networks easily, as will be shown later in Section 6.

Suppose that P_0 wishes to broadcast m messages M_1, M_2, ..., M_m, to all processors in Q_n. Our algorithm is based on the following simple idea. In at most $\lfloor m/2 \rfloor$ steps, the originating processor P_0 first sends half of the m messages to its neighboring processor $P_{N/2}$ along dimension $n-1$. Now we have two Q_{n-1}'s such that P_0 in the left Q_{n-1} contains messages M_1, M_2, ..., $M_{\lceil m/2 \rceil}$ (as well as $M_{\lceil m/2 \rceil+1}$, ..., M_m) and $P_{N/2}$ in the right Q_{n-1} contains messages $M_{\lceil m/2 \rceil+1}$, ..., M_m. We can now broadcast roughly $m/2$ messages in each Q_{n-1}

recursively, in parallel. Once this is done, each processor in the left subcube has messages M_1, M_2, ..., $M_{\lceil m/2 \rceil}$ and each processor in the right subcube has messages $M_{\lceil m/2 \rceil+1}$, ..., M_m. In another $\lceil m/2 \rceil$ steps, each processor in the left Q_{n-1} exchanges its messages in parallel with its neighbors in the right Q_{n-1} along dimension $n-1$ and now each processor in Q_n has all m messages.

In this algorithm, the recursion ends when either (1) a subcube of dimension n', where $n' \leq n$, has only one message to broadcast to all its processors in the $Q_{n'}$, and in this case, it simply uses the standard broadcasting algorithm to broadcast the single message within the $Q_{n'}$. This case happens when $m \leq N$; or (2) in each 0-cube (with 1 processor) each processor has 1 or more messages (roughly m/N of them). This is the case when $m > N$. In this case, nothing needs to be done.

Formally, the broadcasting algorithm is given as follows (note that initially P_j is P_0, and the algorithm is called with parameters n and m):

Algorithm BROADCAST($\hat{n}$, $\hat{m}$)

1. The originating processor P_j sends messages $\hat{M}_i$, $\lceil \hat{m}/2 \rceil + 1 \leq i \leq \hat{m}$, to a processor in the right subcube $Q_{\hat{n}-1}$, along dimension $\hat{n}-1$.

2. In parallel, each $Q_{\hat{n}-1}$ recursively broadcasts (at most) $\lceil \hat{m}/2 \rceil$ messages by calling BROADCAST($\hat{n}-1$, $\lceil \hat{m}/2 \rceil$); the recursion ends when either of the terminating conditions mentioned above is met.

3. Each processor exchanges all its messages with its neighbor along dimension $\hat{n}-1$. ■

As for the time complexity of the algorithm, let $t(n,m)$ be the time required to broadcast m messages in Q_n. Clearly, Step 1 takes $O(m)$ time. Step 2 takes $t(n-1, \lceil m/2 \rceil)$ time. Step 3 also takes $O(m)$ time. Therefore,

$$t(n,m) \;=\; t(n-1, \lceil m/2 \rceil) + O(m).$$

When $m < N$, we have

$$\begin{aligned}
t(n,m) &= t(n-1, \lceil m/2 \rceil) + cm \\
&= t(n-2, \lceil m/2^2 \rceil) + c\lceil m/2 \rceil + cm \\
&= t(n-3, \lceil m/2^3 \rceil) + c\lceil m/2^2 \rceil + c\lceil m/2 \rceil + cm \\
&\;\;\vdots \\
&= t(n',1) + c(m + \lceil m/2 \rceil + \lceil m/2^2 \rceil + \cdots) \\
&= t(n',1) + O(m(1 + 1/2 + 1/2^2 + \cdots)) \qquad (n' < n) \\
&= O(m + \log N),
\end{aligned}$$

where c is some constant. When $m \geq N$, we have

$$\begin{aligned}
t(n,m) &= t(n-1, \lceil m/2 \rceil) + cm \\
&= t(n-2, \lceil m/2^2 \rceil) + c\lceil m/2 \rceil + cm
\end{aligned}$$

$$
\begin{aligned}
&= t(n-3, \lceil m/2^3 \rceil) + c\lceil m/2^2 \rceil + c\lceil m/2 \rceil + cm \\
&\;\vdots \\
&= t(0, \lceil m/N \rceil) + c(m + \lceil m/2 \rceil + \lceil m/2^2 \rceil + \cdots) \\
&= t(0, \lceil m/N \rceil) + O(m(1 + 1/2 + 1/2^2 + \cdots)) \\
&= O(m).
\end{aligned}
$$

In any case, time required is bounded by $O(m + \log N)$.

We now consider the second problem: Given that processor P_i contains m values x_{ij}, $0 \le i \le N-1$ and $1 \le j \le m$, compute $\sum_{i=0}^{N-1} x_{ij}$, for all $1 \le j \le m$, so that every processor in the hypercube contains these m sums.

Clearly, this problem can be solved on an n-cube in time $O(m + \log N)$ by using the algorithm given in [13] that we mentioned earlier for computing m parallel prefix sums.

The following algorithm accomplishes this task in time $O(m+\log N)$, using the standard hypercube ASCEND/DESCEND scheme and requiring no pipelining. The idea is to let each subcube with $N/2$ processors recursively compute the required sums for half of all the values x_{ij} so that these sums are stored in all the processors in the left subcube (every processor in the left subcube contains $\sum_{i=0}^{N-1} x_{ij}$, for all $1 \le j \le \lceil m/2 \rceil$) and all the processors in the right subcube (every processor in the right subcube contains $\sum_{i=0}^{N-1} x_{ij}$, for all $\lceil m/2 \rceil + 1 \le j \le m$, respectively, and then exchange the sums between the two subcubes. Now every processor has all the sums required. To do so, we first divide the hypercube into two subcubes. Each processor in the left subcube sends the last $\lfloor m/2 \rfloor$ values to its neighbor in the right subcube along dimension $n-1$ while each processor in the right subcube sends the first $\lceil m/2 \rceil$ values to its neighbor in the left subcube, also along dimension $n-1$. Now each subcube has all the necessary values to compute $m/2$ sums recursively. Finally, each processor exchanges its (roughly) $m/2$ sums with its neighbor along dimension $n-1$ in $O(m)$ time.

The running time for this algorithm is also $O(m + \log N)$ and the analysis is similar to that of algorithm BROADCAST and is thus omitted.

4 Point Location Algorithm for Simple Polygons

The point location problem can be described as follows: Given a query point z, we want to determine whether it lies in a region R [15]. In this section, we first consider the version where z is a planar point and R is a simple polygon. We then extend our work to answer m queries: Given m planar points and a simple polygon, determine which points are in the polygon. Sequentially [15],

(a) one point can be located in a simple N-gon G in $O(N)$ time, without preprocessing;

(b) with $O(N \log N)$ preprocessing, one point can be located in a simple N-gon in $O(\log N)$ time; for convex and star-shaped polygons, the preprocessing time becomes $O(N)$. Thus, locating m points in a straightforward way requires $O(m \log N)$ time with preprocessing.

Using the standard broadcasting algorithm [10] and the two data communication algorithms described in the previous section, it is simple to parallelize the point location algorithm from [15] for one and m queries on a hypercube of size N. We will first give a trivial $O(\log N)$ algorithm for locating one point in an N-gon. We also assume that initially, each processor of the hypercube holds one edge of the N-gon and P_0 has the query point z. The idea of the sequential algorithm for locating one point (without preprocessing) is as follows: (1) Compute L, the number of intersections (to the left of z) of polygon edges with the horizontal line containing z; (2) If L is odd then z is internal to G (otherwise, z is external).

An algorithm for locating one point in an N-gon on a hypercube with N processors is as follows:

1. Processor P_0 broadcasts z to all other processors;

2. Each P_i sets a variable l_i to 1 if the edge stored in the processor intersects the horizontal line containing z to the left of z (otherwise, P_i sets l_i to 0).

3. If $L = \sum_{i=0}^{N-1} l_i$ is odd then z is internal to the polygon, else it is external. ■

Step 1 takes $O(\log N)$ time. Step 2 takes $O(1)$ time. Step 3 also takes $O(\log N)$ time (for example, we can use the standard DESCEND paradigm to add all the numbers l_i stored in processors and the final sum is obtained in P_0).

With m points z_j, $1 \leq j \leq m$, the algorithm is as follows:

1. P_0 broadcasts m points to every processor in the hypercube;

2. Do in Parallel: For each point z_j, $1 \leq j \leq m$, processor P_i, $0 \leq i \leq N-1$, sets a variable l_{ij} to 1 if the polygon edge stored in P_i intersects the horizontal line containing z_j to the left of z_j (otherwise, l_{ij} is set to 0);

3. Use the second data communication algorithm presented in the previous section to get m sums in P_0, namely, $L_j = \sum_{i=0}^{N-1} l_{ij}$, $1 \leq j \leq m$;

4. If L_j is odd then z_j is internal to the polygon, else it is external. ■

As for the time complexity of the algorithm: Steps 1 and 3 take $O(m+\log N)$ time, while Steps 2 and 4 require $O(m)$ time. Therefore, the entire algorithm requires $O(m + \log N)$ time to answer m queries without any preprocessing.

We note here that there exists a parallel algorithm for point location in a polygon using a binary tree of processors. It locates one point in $O(\log N)$ time using N processors [3]. So, using pipelining, m points can be located in $O(m + \log N)$ time on a binary tree, and consequently, on a hypercube because of the embedding of binary trees into the hypercube mentioned earlier. Of course, pipelining has to be used here.

5 Point Location Algorithm for Planar Subdivisions

A more general point location problem is defined for planar subdivisions [15]. A planar subdivision is a partition of the plane into regions bounded by straight-line edges. Each region is a polygon whose corners are the vertices of the subdivision. The problem of point location in a planar subdivision calls for determining the region of the subdivision occupied by each of a given set of query points [5]. Various parallel algorithms on different parallel computational models have been found for this version of point location problem. The reader is referred to [5] for a more comprehensive review.

A planar subdivision with $O(N)$ vertices consists of many simple polygons. Let these polygons be p_1, p_2, ..., p_s, for some $s \geq 1$. The edges of these polygons are stored in the processors as follows: the edges of p_i, $1 \leq i \leq s$, are stored in the first available processor, starting from processor P_0. Note that some edges may be stored twice since they can be shared by two neighboring polygons. Clearly, we have to assume that the total number of polygon edges is no more than N, the number of processors. For example, for the subdivision in Fig. 1, the edges are stored in a 16-processor hypercube as follows:

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
e_{11}	e_{12}	e_{13}	e_{21}	e_{22}	e_{23}	e_{31}	e_{32}	e_{33}	e_{41}	e_{42}	e_{43}	e_{44}	e_{51}	e_{52}	e_{53}

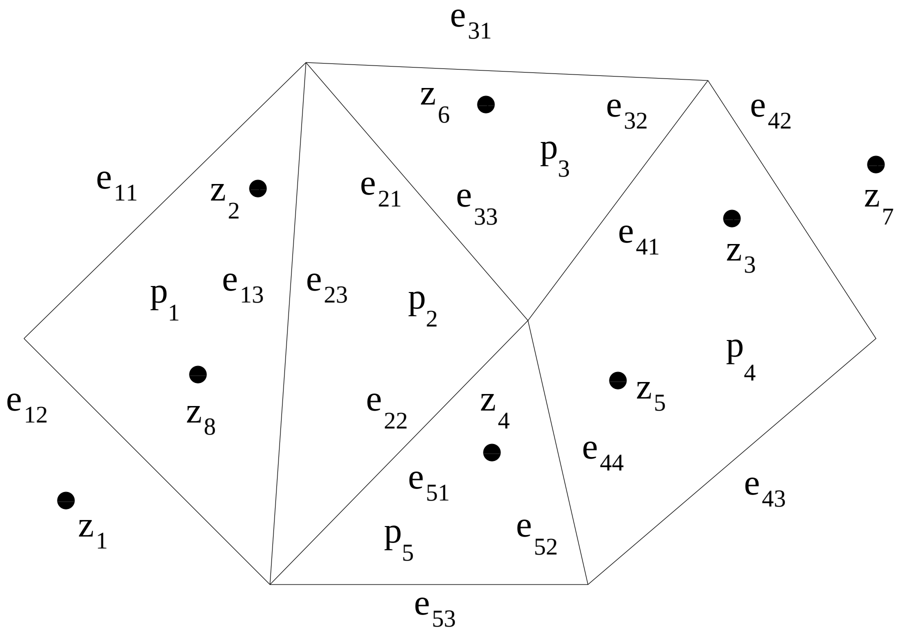

Figure 1: A Subdivision

The algorithm to locate m points z_j, for $1 \leq j \leq m$, in a subdivision is as follows:

1. P_0 broadcasts m points to every processor in the hypercube;

2. Do in Parallel: For each point z_j, $1 \leq j \leq m$, processor P_i, $0 \leq i \leq N-1$, sets a variable l_{ijk} to 1 if the edge of polygon p_k, $1 \leq k \leq s$, stored in P_i, intersects the horizontal line containing z_j to the left of z_j (otherwise, l_{ijk} is set to 0);

3. Compute, for each point z_j and each polygon p_k: $L_{jk} = \sum l_{ijk} \bmod 2$ where P_i stores an edge of p_k;

4. Compute $\sum_{k=1}^{s} L_{jk}$, for each $1 \leq j \leq m$, and store this value in P_0. ■

Steps 1 and 2 are the same as in our previous algorithm for locating m points in a simple polygon. Step 4 can be done by using our data communication algorithm for computing m sums (to be stored in P_0). The time required for the three steps is therefore $O(m + \log N)$. We now explain Step 3 in detail.

Clearly, since any point z_j can be located in at most one polygon of the subdivision, we have $\sum_{k=1}^{s} L_{jk} \leq 1$. Also, because we care only about the parity of L_{jk} and $\sum_{k=1}^{s} L_{jk}$, the addition in $\sum$ should be done modulo 2 (i.e., exclusive OR).

Step 3 computes s segmented sums, each corresponding to a polygon in the subdivision, for each of the m points to be located. That is, for each query point and each polygon, it computes the number of intersections to the left of the point of the polygon edges with the horizontal line containing that point. In other words, it is doing what Step 3 of the algorithm for locating m points in one simple polygon does, but for s polygons this time. Surprisingly, this step can also be done in time $O(m + \log N)$ on a hypercube of size N, as shown below.

Step 3 is done recursively. Initially, each processor communicates with its neighbor along the lowest dimension (dimension 0). That is, the processors are paired as (P_0, P_1), (P_2, P_4), ..., (P_{N-4}, P_{N-3}), (P_{N-2}, P_{N-1}). For each pair, the sums (m of them) are computed (more on this later) and stored in the processor with the smaller label for the first $m/2$ values and the processor with the larger label for the second $m/2$ values. Now, for the first $m/2$ points to be located, their required sums can be computed recursively in an Q_{n-1} consisting of processors P_0, P_2, P_4, ..., P_{2^n-2}, while in parallel, for the second $m/2$ points to be located, their required sums can be computed recursively in another Q_{n-1} consisting of processors P_1, P_3, P_5, ..., P_{2^n-1}. The termination condition for the recursion is similar to that of the broadcasting algorithm for multiple messages. Also, the time complexity for the step satisfies the equation $t(n,m) = t(n-1, m/2) + O(m)$, resulting in $t(n,m) = O(m + \log N)$.

The algorithm is in the class of ASCEND since each processor communicates with its neighbors along dimensions 0, 1, 2, ..., $n-1$, in this order.

For each point z to be located, when two processors communicate with each other in order to compute sums, some care needs to be taken. We can see that for any point z_j, the number of 1's in each processor can be more than one during the computation. However, for that point, the number of 1's in each processor at any moment during the computation is never more than three because (1) edges for each polygon are stored in a segment of consecutively labeled processors; (2) any segment of consecutively labeled processors contains edges of zero or more polygons whose edges are all stored in this segment plus at most two polygons whose edges are stored in the segment partially; and (3) each planar point can be located in at most one polygon of the subdivision. Therefore, when computing the sum modulo 2, the two 1's to be added must be from the same polygon.

The total time for the algorithm for locating m points in a subdivision with $O(N)$ vertices on a hypercube with N processors is therefore $O(m + \log N)$. Previously, an $O(\log^3 N)$ algorithm was developed by Lee and Preparata that runs on a cube-connected-cycles (CCC) network with $O(N)$ processors to locate $O(N)$ points [11]. Our algorithm performs better than this algorithm when $m = o(\log^3 N)$.

We now illustrate our algorithm by the following example. The subdivision is the same as given in Fig. 1. The subdivision, labeled polygon edges, and the 8 points to be located are all shown in the figure.

After Steps 1 and 2, we have:

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
z_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z_2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z_3	1	0	1	1	0	1	0	1	1	1	0	0	0	0	0	0
z_4	0	1	1	0	1	1	0	0	0	0	0	0	0	1	0	0
z_5	0	1	1	0	1	1	0	0	0	0	0	0	1	1	1	0
z_6	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0
z_7	1	0	1	1	0	1	0	1	1	1	1	0	0	0	0	0
z_8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

During the computations of Step 3, each processor first communicates with its neighbor along dimension 0 (the lowest dimension in the hypercube) in order to compute the sum. The results are shown below. The subscript to each 1 indicates the polygon associated with the value.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
z_1																
z_2	1_1															
z_3	1_1		1_1,1_2		1_2		1_3		1_3,1_4							
z_4	1_1		1_1										1_5			
z_5		1_1		1_1										1_4,1_5		1_5
z_6		1_1		1_1,1_2		1_2				1_3						
z_7		1_1		1_1,1_2		1_2		1_3		1_3,1_4		1_4				
z_8		1_1														

The recursion starts since we now have two 3-cubes (P_0, P_2, P_4, P_6, P_8, P_{10}, P_{12}, P_{14}), and (P_1, P_3, P_5, P_7, P_9, P_{11}, P_{13}, P_{15}). Each processor interacts with its neighbor along dimension 1 (the lowest dimension in each subcube). The results are given below.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
z_1																
z_2	1_1															
z_3			1_2				1_2,1_3				1_3,1_4					
z_4															1_5	
z_5														1_4		
z_6		1_2				1_2				1_3						
z_7				1_2				1_2,1_3				1_3				
z_8				1_1												

We now have four 2-cubes (P_0, P_4, P_8, P_{12}), (P_2, P_6, P_{10}, P_{14}), (P_1, P_5, P_9, P_{13}), and (P_3, P_7, P_{11},P_{15}). Each processor interacts with its neighbor along dimension 2 (the lowest dimension in each subcube). The results are given below.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
z_1																
z_2					1_1											
z_3			1_3								1_3,1_4					
z_4															1_5	
z_5										1_4						
z_6														1_3		
z_7				1_3								1_3				
z_8								1_1								

We now have eight 1-cubes (P_0, P_8), (P_4, P_{12}), (P_2, P_{10}), (P_6, P_{14}), (P_1, P_9), (P_5, P_{13}), (P_3, P_{11}), and (P_7, P_{15}). Each processor interacts with its neighbor along dimension 3 (the lowest dimension in each subcube). The results are given below.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
z_1																
z_2					1_1											
z_3			1_4													
z_4							1_5									
z_5		1_4														
z_6						1_3										
z_7																
z_8								1_1								

This is the end of Step 3. Finally, we just use our second data communication algorithm developed in Section 3 to move the final results to processor P_0.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
z_1																
z_2	1_1															
z_3	1_4															
z_4	1_5															
z_5	1_4															
z_6	1_3															
z_7																
z_8	1_1															

The final results indicate that z_2 is in polygon p_1, z_3 in p_4, z_4 in p_5, z_5 in p_4, z_6 in p_3, and z_8 in p_1,

6 Extensions of the Results to Other Networks

In this section, we show how to extend the first algorithm that locates m points in a simple polygon on hypercubes to some other interconnection networks.

It can be seen that two of the most important components in our planar point location algorithm for m query points in a simple polygon are the two communication algorithms. Therefore, if these two algorithms can be implemented on another interconnection network efficiently, it is reasonable to expect this algorithm to have a similar performance on that network as well. The new networks we have in mind are the star and pancake interconnection networks. It will be clear after this section that our discussion applies to any interconnection network that is a recursively decomposable Cayley graph [8].

The star and pancake graphs were proposed as attractive alternatives to the hypercube topology for interconnecting processors in a parallel computer (interconnection network), and compare favorably with it in several aspects [1, 2]. For example, an n-star or n-pancake has $N = n!$ nodes, but both its degree and diameter are $O(n)$, i.e., sub-logarithmic in the number of vertices, while a hypercube with $O(n!)$ vertices has a degree and diameter of $O(\log(n!))$ $= O(n \log n)$, i.e., logarithmic in the number of vertices. Other attractive properties include their symmetry properties, as well as many desirable fault tolerance characteristics. These two interconnection networks have received much attention lately [1, 2, 6, 7, 16].

Let V_n be the set of all $n!$ permutations of symbols 1, 2, ..., n. For any permutation $v \in V_n$, if we denote the i^{th} symbol of v by $v(i)$, then v can be written as $v(1)v(2)\cdots v(n)$. A *star interconnection network* on n symbols, $S_n = (V_n, E_{S_n})$, is an undirected graph with $n!$ vertices, where each vertex v is connected to $n-1$ vertices which can be obtained by interchanging the first and i^{th} symbols of v, i.e., $(v(1)v(2)\cdots v(i-1)v(i)v(i+1)\cdots v(n), v(i)v(2)\cdots v(i-1)v(1)v(i+1)\cdots v(n)) \in E_{S_n}$, for $2 \leq i \leq n$. S_n is also called an n-star. Fig. 2 shows S_4.

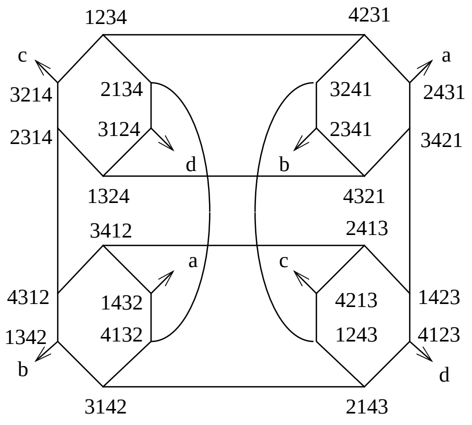

Figure 2: A 4-Star S_4

A *pancake interconnection network* on n symbols, $P_n = (V_n, E_{P_n})$, is an undirected graph with $n!$ vertices, where each vertex v is connected to $n-1$ vertices which can be obtained by flipping the first i symbols of v, i.e., $(v(1)v(2)\cdots v(i-1)v(i)v(i+1)\cdots v(n), v(i)v(i-1)\cdots v(2)v(1)v(i+1)\cdots v(n)) \in E_{P_n}$, for $2 \leq i \leq n$. P_n is also called an n-pancake. Fig. 3 shows P_4. Clearly, $S_n = P_n$, for $n \leq 3$. S_n and P_n are in the family of Cayley graphs [1].

Since our discussion and results in this section apply to both networks, we henceforth use X_n to denote either S_n or P_n.

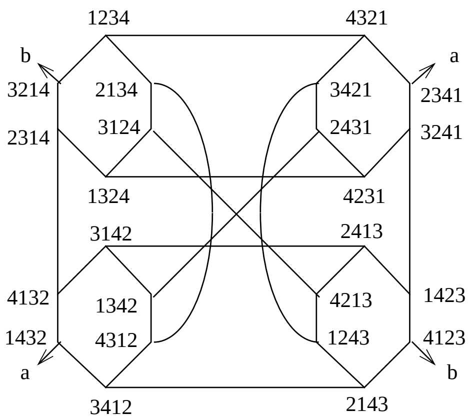

Figure 3: A 4-Pancake P_4

Let $X_{n-1}(i)$ be a sub-graph of X_n induced by all the vertices with the same last symbol i, for some $1 \leq i \leq n$. It can be seen that $S_{n-1}(i)$ is an $(n-1)$-star and that $P_{n-1}(i)$ is an $(n-1)$-pancake, both defined on symbols $\{1, 2, \cdots, n\} - \{i\}$. It follows that X_n can be decomposed into n X_{n-1}'s: $X_{n-1}(i)$, $1 \leq i \leq n$ [1, 2]. For example, S_4 in Fig. 2 contains four 3-stars, namely $S_3(1)$, $S_3(2)$, $S_3(3)$, and $S_3(4)$, by fixing the last symbol at 1, 2, 3, and 4, respectively. P_n can also be decomposed similarly.

Let I: $i_1, i_2, ..., i_l$ and J: $j_1, j_2, ..., j_l$ be two sequences from $\{1, 2, ..., n\}$ such that no two elements of I are equal, no two elements of J are equal, and $\{i_1, i_2, ..., i_l\} \cap \{j_1, j_2, ..., j_l\} = \emptyset$, i.e., $|\{i_1, i_2, \cdots, i_l, j_1, j_2, \cdots, j_l\}| = 2l$. It is desired to exchange the contents of $X_{n-1}(i_1)$, $X_{n-1}(i_2)$, ..., $X_{n-1}(i_l)$ with those of $X_{n-1}(j_1)$, $X_{n-1}(j_2)$, ..., $X_{n-1}(j_l)$ such that the contents of $X_{n-1}(i_k)$ are exchanged with $X_{n-1}(j_k)$, for $1 \leq k \leq l$. This task can be achieved in constant time by a procedure named GROUP-COPY [6].

As mentioned earlier, most hypercube algorithms are of the type ASCEND or DESCEND, or some variation thereof [14]. Those hypercube algorithms that fall in this class can be divided further into two sub-classes according to whether or not they preserve the ordering of the processors while ascending or descending, i.e., whether or not a processor ranked k^{th} in one subcube has to communicate with a processor also ranked k^{th} in another subcube. Basically, an algorithm in the first sub-class means ordering is important when processors communicate with each other while the second sub-class algorithms do not care about the ordering. Examples of algorithms in the first sub-class are Batcher's bitonic merging and sorting algorithms [9] as implemented on the hypercube. Algorithms that belong to the second sub-class include broadcasting algorithms, where it does not matter which processors receive the information first as long as eventually every processor receives a copy of the message being broadcast. Since Procedure GROUP-COPY allows one to perform recursive doubling/halving on X_n in $O(1)$ time, it is obvious that using it, we can directly implement on the star and pancake networks any algorithm for the hypercube that is in the second sub-class, using asymptotically the same number of processors and without time loss.

Using the routing scheme GROUP-COPY, various optimal broadcasting algorithms for

X_n have been found, including the one for broadcasting m messages in $O(m + \log N) = O(m + n \log n)$ time [17]. Our second data communication algorithm also belongs to the second sub-class algorithms mentioned above and therefore can also be implemented on X_n without time loss, i.e., in $O(m + \log N) = O(m + n \log n)$ time. This fact immediately implies a parallel planar point location algorithm for the star and pancake interconnection networks that locates one point in an N-gon in $O(\log N) = O(n \log n)$ time, and m points in $O(m + \log N) = O(m + n \log n)$ time.

As we see from the Step 3 of the algorithm for locating m points in a subdivision in Section 5 and the corresponding example, the order in which each processor communicates with other processors is critical in order to guarantee that the number of 1's in each processor during the entire computation is constant. Therefore, this algorithm for subdivisions can not be readily implemented on stars and pancakes without time loss.

7 Conclusion

We have presented several novel data communication algorithms for hypercubes and related interconnection networks. The performances of these algorithms match those of previous ones (whenever the latter exist). However, our algorithms are simpler in that they use the standard ASCEND/DESCEND paradigms that are common for hypercube-like networks. Moreover, the idea can be easily applied to other similar networks which are recursively decomposable [8]. This eliminates the need to find a mapping first that embeds a graph of constant degree (such as a binary tree) into the corresponding network, which is essential to using the technique of pipelining. We demonstrated the applications of our data communication algorithms by using them to solve several problems in computational geometry. The performances of these algorithms either match those of earlier algorithms or show an improvement in certain cases. Our future study includes finding more applications of the data communication algorithms developed here.

References

[1] S.B. Akers and B. Krishnamurthy, "A Group Theoretic Model for Symmetric Interconnection Networks," *IEEE Transaction on Computers*, Vol. c-38, No. 4, April 1989, pp. 555-566.

[2] S.B. Akers, D. Harel, and B. Krishnamurthy, "The Star Graph: An Attractive Alternative to the n-cube," *Proc. International Conference on Parallel Processing*, August, 1987, pp. 393-400.

[3] S.G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[4] S.G. Akl, *Parallel Computation: Models and Methods*, Prentice Hall, Upper Saddle River, NJ, 1997.

[5] S.G. Akl and K.A. Lyons, *Parallel Computational Geometry*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[6] S.G. Akl and K. Qiu, "A Novel Routing Scheme on the Star and Pancake Networks and Its Applications," *Parallel Computing*, 19, 1993, pp. 95-101.

[7] S.G. Akl, K. Qiu, and I. Stojmenović, "Fundamental Algorithms for the Star and Pancake Interconnection Networks with Applications to Computational Geometry," *Networks: An International Journal*, Vol. 23, 1993, pp. 215-225.

[8] C. Gowrisankaran, "Broadcasting on Recursively Decomposable Cayley Graphs," *Discrete Applied Mathematics*, Vol. 53, 1-3, Sept. 1994, pp. 171-182.

[9] K. Batcher, "Sorting Networks and Their Applications," *Proc. of the AFIPS Spring Joint Computing Conference*, Vol. 32, Washington, D.C., 1968, pp. 307-314.

[10] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing, Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc., Don Mills, Ontario, 1994.

[11] D.T. Lee and F.P. Preparata, "Parallel Batched Planar Point Location on the CCC," *Information Processing Letters*, 33, 1989, pp. 175-179.

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

[13] E.W. Mayr and C.G. Plaxton, "Pipelined Parallel Prefix Computations, and Sorting on a Pipelined Hypercube," *Journal of Parallel and Distributed Computing*, Vol. 17, 1993, pp. 374-380.

[14] F.P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications of the ACM*, Vol. 24, May 1981, pp. 300-309.

[15] F.P. Preparata and M.I. Shamos, *Computational Geometry, An Introduction*, Springer-Verlag, New York, 1985.

[16] K. Qiu, S.G. Akl, and H. Meijer, "On Some Properties and Algorithms for the Star and Pancake Interconnection Networks," *Journal of Parallel and Distributed Computing*, Vol. 22, 1994, pp. 16-25.

[17] K. Qiu, "Broadcasting on the Star and Pancake Interconnection Networks," *Proc. of the 9th IEEE International Parallel Processing Symposium*, Santa Barbara, California, April 1995, pp. 660-665.

[18] K. Qiu and S.G. Akl, "Parallel Point Location Algorithms on Hypercubes," *Proc. of the 10th International Conference on Parallel and Distributed Computing Systems*, New Orleans, Louisiana, Oct. 1997.