

Fundamental Algorithms for the Star and Pancake Interconnection Networks with Applications to Computational Geometry*

S. G. Akl and K. Qiu

Department of Computing & Information Science, Queen's University, Kingston, Ontario, Canada

I. Stojmenović

Computer Science Department, University of Ottawa, Ottawa, Ontario, Canada

The star and pancake networks were recently proposed as attractive alternatives to the hypercube topology for interconnecting processors in a parallel computer. However, few parallel algorithms are known for these networks. In this paper, we present several data communication schemes and basic algorithms for these two networks. These algorithms are then used to develop parallel solutions to various computational geometric problems on both networks. Computational geometry is just one area where the algorithms proposed here can be applied. Indeed, we believe that these algorithms are interesting and important in their own right and are fundamental to the design of solutions on the star and pancake networks to a host of other problems. © 1993 by John Wiley & Sons, Inc.

1. INTRODUCTION

Given a set of generators for a finite group \mathcal{G} , the *Cayley graph* with respect to \mathcal{G} is defined as follows: The vertices of the graph correspond to the elements of the group \mathcal{G} , and there is an edge (a, b) for $a, b \in \mathcal{G}$ if and only if there is a generator g such that $ag = b$ [1]. We require that the set of generators be closed under inversion so that the resulting graph can be viewed as being undirected [1].

Let \mathcal{G}_n be a symmetric group on n symbols and V_n be the set of all $n!$ permutations of symbols $1, 2, \dots, n$. A *star interconnection network* on n symbols, $S_n = (V_n, E_{S_n})$, is a Cayley graph with generators $g_i = i23 \dots (i-1)(i+1) \dots n$, $2 \leq i \leq n$. Each vertex in S_n is connected to $n-1$ vertices that can be obtained

by interchanging the first symbol (leftmost symbol) of the vertex with the i th symbol, $2 \leq i \leq n$. S_n is also called an n -star. Figure 1(a) shows S_4 .

A *pancake interconnection network* on n symbols, $P_n = (V_n, E_{P_n})$, is a Cayley graph with generators $h_i = i(i-1) \dots 321(i+1)(i+2) \dots n$, $2 \leq i \leq n$. Each vertex in P_n is connected to $n-1$ vertices that can be obtained by flipping the first i symbols, $2 \leq i \leq n$. P_n is also called an n -pancake. Figure 1(b) shows P_4 . Note that S_n and P_n have the same vertex set. Clearly, $h_i = g_i$ for $i \leq 3$, and $S_n = P_n$ for $n \leq 3$. In those cases where the discussion and results apply to both networks, we will use X_n to denote either S_n or P_n .

Both the star and pancake interconnection networks are attractive alternatives to the hypercube for interconnecting processors in a parallel computer and compare favorably with it in several aspects [1, 2]. For example, both the degree and diameter of X_n are $O(n)$, i.e., sublogarithmic in the number of vertices of X_n ,

*This work is supported by the Natural Sciences and Engineering Research Council of Canada.

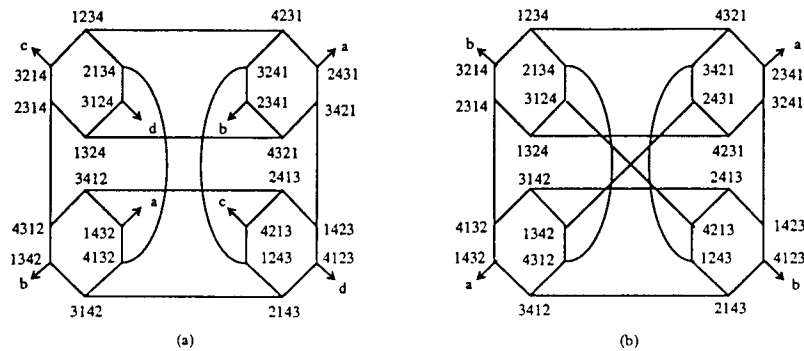


Fig. 1. (a) A 4-star S_4 and (b) a 4-pancake P_4 .

whereas 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 [3].

Definition 1. Let $X_{n-1}(i)$ be a subgraph of X_n induced by all the vertices with the same last symbol i , for some $1 \leq i \leq n$.

From this definition, X_n can be decomposed into n X_{n-1} 's: $X_{n-1}(i)$, $1 \leq i \leq n$. From the definitions of S_n and P_n , it can be seen that $S_{n-1}(i)$ is an $(n - 1)$ -star and $P_{n-1}(i)$ is an $(n - 1)$ -pancake, both of them defined on symbols $\{1, 2, \dots, n\} - \{i\}$ [1, 2]. For example, S_4 in Figure 1(a) 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.

We now present a processor ordering, first proposed by Menn and Somani [12], which is quite appropriate for the hierarchical structure of X_n .

Definition 2. In X_n , let p denote the processor associated with the vertex $a_1 a_2 \dots a_n$ and q denote the processor associated with the vertex $b_1 b_2 \dots b_n$. The ordering, $<$, on the processors is defined as follows: $p < q$ if there exists an i , $1 \leq i \leq n$, such that $a_j = b_j$ for $j > i$, and $a_i < b_i$. In other words, the processors are ordered in reverse lexicographic order (i.e., lexicographic order if we read from right to left). If $p < q$, we say that p precedes q . The rank $r(u)$ of a vertex u is the number of vertices v such that $v < u$, i.e., $r(u) = |\{v | v < u, v \in V_n\}|$. Clearly, $0 \leq r(u) \leq n! - 1$.

Throughout the paper, any vertex $u = a_1 a_2 \dots a_n$ will be referred to by $a_1 a_2 \dots a_n$ and/or $r(u)$. It is also assumed that in one step requiring unit time each processor may send or receive a constant number of data items to or from one of its neighbors. Henceforth, we use *vertex* and *processor* interchangeably.

The star and pancake interconnection networks

have received much attention recently. A general review of the properties of these two networks and their associated algorithms is provided in [17]. It is worth pointing out that most efforts to date have been directed toward deriving properties of the two networks instead of designing parallel algorithms for them. In this paper, we present a number of results in connection with the star and pancake networks. Several basic algorithms are proposed that are then used to design parallel solutions to various computational geometric problems on both networks. In particular, we describe (i) an algorithm for finding the convex hull of a set of $n!$ planar points on a star or pancake network with $n!$ vertices in $O(n^3 \log n)$ time, a performance that matches that of the best-known sorting algorithm on each of the two networks, and (ii) algorithms for solving several other geometric problems by the merging slopes technique. Computational geometry is just one area where the basic algorithms proposed here can be applied. Indeed, we believe that these algorithms are interesting and important in their own right and are fundamental to the design of solutions on the star and pancake networks to a host of other problems.

The paper is organized as follows: Section 2 presents basic algorithms on the two networks. We then use these algorithms in Section 3 to develop efficient solutions to the convex hull problem and other related computational geometry problems that can be solved by the merging slopes technique. Our concluding remarks are offered in Section 4.

2. BASIC ALGORITHMS

In what follows we describe a representation of S_n and P_n that greatly simplifies the exposition in the next section. If we arrange all the vertices in X_n into an $n \times (n - 1)!$ array in row-major order (in terms of the processor ordering), then row i becomes $X_{n-1}(i)$ [12]. The vertices in X_4 are given in Table I. From Definition

TABLE I. Vertices of X_4

4321	3421	4231	2431	3241	2341
4312	3412	4132	1432	3142	1342
4213	2413	4123	1423	2143	1243
3214	2314	3124	1324	2134	1234

2, we can see that all the vertices in the same column of the $n \times (n - 1)!$ array (Table I) have the same rank in their respective X_{n-1} 's. For example, vertices 2431, 1432, 1423, and 1324 are all ranked third in $X_3(1)$, $X_3(2)$, $X_3(3)$, and $X_3(4)$, respectively. Now if we consider these vertices as the vertices of S_n and apply generator g_n to every vertex, we get another $n \times (n - 1)!$ array (Table II) in which, by the definition of S_n , each column is connected to form a simple path, i.e., a linear array of processors [12]. This means that the contents of $S_{n-1}(i)$ can be copied to the processors of $S_{n-1}(i + 1)$, and vice versa, in a bijective and order-preserving way using three steps. For example, vertex 1432 of $S_3(2)$ sends its content to 1423 of $S_3(3)$ as follows: From 1432 (Table I) to 2431 (Table II) in one step; from 2431 to 3421 (both in a column of Table II) in a second step; and from 3421 (Table II) to 1423 (Table I) in a third step. This fact allows us to consider each column of vertices of S_n as "connected" to form a linear array when these vertices are arranged as an $n \times (n - 1)!$ array in row-major order.

To obtain the same property in P_n , we need the following routing algorithms: Let $I: i_1, i_2, \dots, i_l$ and $J: j_1, j_2, \dots, j_l$ be two sequences from $\{1, 2, \dots, n\}$ such that no two elements of I are equal, no two elements of J are equal, and $\{i_1, i_2, \dots, i_l\} \cap \{j_1, j_2, \dots, j_l\} = \emptyset$. It is desired to exchange the contents of $X_{n-1}(i_1), X_{n-1}(i_2), \dots, X_{n-1}(i_l)$ with those of $X_{n-1}(j_1), X_{n-1}(j_2), \dots, X_{n-1}(j_l)$ such that the contents of $X_{n-1}(i_m)$ are exchanged with those of $X_{n-1}(j_m)$, for $1 \leq m \leq l$. [By sending the contents of $X_{n-1}(i)$ to $X_{n-1}(j)$, we mean that the content of each processor in $X_{n-1}(i)$ is routed to a processor in $X_{n-1}(j)$ such that no two processors in $X_{n-1}(i)$ send their contents to the same processor in $X_{n-1}(j)$. By exchanging the contents of $X_{n-1}(i)$ and $X_{n-1}(j)$, we mean that the contents of $X_{n-1}(i)$ are sent to the processors of $X_{n-1}(j)$ and the contents of $X_{n-1}(j)$

are sent to the processors of $X_{n-1}(i)$.] This task can be achieved in constant time by procedure GROUP-COPY [5]. Note that GROUP-COPY does not take processor ordering into account.

Another routing scheme that uses GROUP-COPY as a subroutine allows us to exchange in constant time the contents of $P_{n-1}(i)$ (row i) with the contents of $P_{n-1}(i + 1)$ (row $i + 1$) in an order-preserving way, i.e., the content of vertex u ranked r^{th} in $P_{n-1}(i)$ is exchanged with the content of a vertex v in $P_{n-1}(i + 1)$ that is also ranked r^{th} in $P_{n-1}(i + 1)$ [namely, u and v are in the same column of the $n \times (n - 1)!$ array], for all $1 \leq i \leq n - 1$ [5]. Thus, this routing scheme "simulates" a linear array. Therefore, the vertices of each column in P_n when arranged as an $n \times (n - 1)!$ array, in row-major order, can also be viewed as connected to form a linear array.

The above discussion allows us to make the following two assumptions in the remainder of this paper:

Assumption 1. *The vertices of an X_k , $2 \leq k \leq n$, are always viewed as a $k \times (k - 1)!$ array where vertices are listed in row-major order.*

Assumption 2. *In the arrangement of Assumption 1, the vertices in each column are connected directly into a linear array of length k either in S_n or in P_n .*

Since S_n and P_n were originally proposed in 1986, few algorithms have been developed for them. This is especially true in the case of P_n . In what follows, we will see that the routing schemes that allow us to make the above assumptions about X_n , as well as procedure GROUP-COPY, enable us to view the two networks as one and, consequently, to develop parallel algorithms that work on both (while previous algorithms work on either the star or pancake, but not both). Because sorting is such a basic and important problem, we include the sorting and merging algorithms in this section.

2.1. Broadcasting

Suppose that a vertex in $X_{n-1}(i)$ wishes to broadcast a piece of information to all the processors in X_n . This problem has a lower bound of $\Omega(\log(n!)) = \Omega(n \log n)$ [2], and in [1] and [2], two separate algorithms, both requiring $O(n \log n)$ time, are given for S_n and P_n . Here, we show that broadcasting on S_n and P_n can be treated as the same operation by applying the routing algorithm GROUP-COPY. The new broadcast algorithm is simple and optimal and works on both networks.

First, the message is broadcast to all other processors in $X_{n-1}(i)$ recursively so that each processor in $X_{n-1}(i)$ has the broadcast message as its content. Then,

TABLE II. Rearranged vertices of X_4

1324	1423	1234	1432	1243	1342
2314	2413	2134	2431	2143	2341
3214	3412	3124	3421	3142	3241
4213	4312	4123	4321	4132	4231

the contents of $X_{n-1}(i)$ are copied to the rest of the X_{n-1} 's by using the constant time procedure GROUP-COPY. In other words, the message is first broadcast recursively to all the vertices in $X_{n-1}(i)$; the contents of $X_{n-1}(i)$ are then copied to the vertices in $X_{n-1}(i + 1)$ in constant time, the contents of $X_{n-1}(i)$ and $X_{n-1}(i + 1)$ are subsequently copied to the vertices in $X_{n-1}(i + 2)$ and $X_{n-1}(i + 3)$, also in constant time. After $\lceil \log n \rceil$ such stages, the message is broadcast to all the vertices in X_n . Let $t(n)$ be the time complexity of the broadcasting algorithm; then $t(n) = t(n - 1) + O(\log n) = O(n \log n)$.

Recently, Mendia and Sarkar found another broadcasting algorithm on the star graph [11]. This algorithm was then generalized by GowriSankaran to work on all recursively decomposable Cayley graphs, which include the pancake graphs [9]. It is worth pointing out that the objectives of [9] and [11] are to find broadcasting algorithms that minimize the exact number of steps. In [9], the exact number of steps for its broadcasting algorithm is $\sum_{k=2}^{n-1} \lceil \log(k) \rceil + (n - 1)$. This number has been improved to $2(\sum_{k=2}^{n-1} \log(2k)) + \lceil 3n/4 \rceil$ in [8].

2.2. Computing Prefix Sums

Given elements x_0, x_1, \dots, x_{N-1} , stored in processors A_0, A_1, \dots, A_{N-1} in a network with processors ordered such that $A_i < A_j$ if $i < j$, and an associative binary operation \oplus , the *prefix sums* problem with respect to the processor ordering is to compute all the quantities $s_j = \bigoplus_{i=0}^j x_i$, $0 \leq j \leq N - 1$. At the end of the computation, we require that processor A_j contain s_j . In what follows, we describe an algorithm that uses procedure GROUP-COPY to compute all prefix sums of an input sequence on X_n with respect to the processor ordering of Definition 2. Suppose that we have computed prefix sums for two groups of substructures as follows:

$$\text{Group 1: } X_{n-1}(i) \dots \dots \dots X_{n-1}(i + k)$$

$$\text{Group 2: } X_{n-1}(i + k + 1) \dots \dots X_{n-1}(i + 2k + 1)$$

and that each processor holds two variables s and t , for storing the partial prefix sum computed so far and the total sum of values in the group to which it belongs, respectively. Let the total sum in group 1 be t_1 and the total sum in group 2 be t_2 . We first use GROUP-COPY to send t_1 to every processor in group 2 and t_2 to every processor in group 1; then, the prefix sums in processors in group 1 remain the same, while the prefix sum s in a processor in group 2 becomes $s \oplus t_1$. The total sum for all the processors in both groups becomes $t_1 \oplus t_2$. All these steps can be accomplished in $O(1)$ time. This leads to a prefix sums algorithm that runs in $t(n) = t(n - 1) + O(\log n) = O(n \log n)$ time on X_n ,

which is optimal. For example, to compute prefix sums on X_7 , we first recursively compute prefix sums on $X_6(i)$, for $1 \leq i \leq 7$, in parallel. Then, prefix sums are computed using the aforementioned method for groups $\{X_6(1), X_6(2)\}$, $\{X_6(3), X_6(4)\}$, $\{X_6(5), X_6(6)\}$, and $\{X_6(7)\}$, in constant time. Using the same technique, we obtain prefix sums for groups $\{X_6(1), X_6(2), X_6(3), X_6(4)\}$ and $\{X_6(5), X_6(6), X_6(7)\}$, also in constant time. Finally, the two groups are combined in constant time, using the same method, to obtain all prefix sums on X_7 .

Direct applications of the broadcasting and prefix sums algorithms include computing the ranks of certain marked vertices and finding the maximum and minimum.

2.3. Sorting and Merging

Given a sequence of numbers stored in a set of processors ordered by $<$, with each processor holding one number, we say that the sequence is sorted in the *forward* direction if for any two numbers x and y held by processors p and q , respectively, $p < q$ implies $x \leq y$. The *reverse* direction is defined similarly.

A sequence of $n!$ numbers stored in X_n is given, with each processor holding one number. The $\Omega(n! \log(n!))$ lower bound on the number of steps required to sort $n!$ numbers sequentially implies an $\Omega(\log(n!)) = \Omega(n \log n)$ lower bound to sort on X_n . Sorting on S_n was studied by Menn and Somani in [12], where an $O(n^3 \log n)$ time algorithm is given. This algorithm sorts on S_n by repeatedly sorting on linear arrays of sizes 2, 3, \dots , n , using *odd-even transposition sort* [10] [recall that whenever we are considering a k -star, $2 \leq k \leq n$, we always think of it as being arranged in a $k \times (k - 1)!$ array in a row-major order and each column can be considered as a connected linear array]. This sorting algorithm implies that given two sorted sequences stored in two groups of S_{n-1} 's:

$$A: S_{n-1}(i), S_{n-1}(i + 1), \dots, S_{n-1}(j)$$

$$B: S_{n-1}(j + 1), S_{n-1}(j + 2), \dots, S_{n-1}(l)$$

(A and B do not necessarily contain the same number of S_{n-1} 's), such that one of the two sequences is in the forward direction and the other in the reverse direction, they can be merged into a sorted sequence stored in

$$C: S_{n-1}(i), S_{n-1}(i + 1), \dots, S_{n-1}(j), S_{n-1}(j + 1), \dots, S_{n-1}(l),$$

in either the forward or reverse direction in $O(n^2)$ time.

It is not hard to see that the sorting and merging algorithms for S_n also apply to P_n with the same time complexity since the vertices in columns of P_n when

arranged in an $n \times (n - 1)!$ array can also be considered as connected (Assumption 2 of Section 2) [16].

Given A , B , and C defined above, and each number in C knows the rank of the vertex in which it was originally before the merging, the problem of unmerging is to permute the list to return each number in C to its original vertex in A or B . This operation is the inverse of merging. This problem can be solved by running the merging algorithm in reverse order, using the given rank information [6]. It can also be solved by applying the operations of Concentration and Translation to be described later (concentrate the numbers of A , then those of B , then translate the numbers of B). Both approaches take $O(n^2)$ time.

Let A and B be two sorted lists stored in two groups of X_{n-1} 's. The *cousins* of $a \in A$ in B are two consecutive numbers b_1 and b_2 in B , such that a lies between b_1 and b_2 in the sorted list $A \cup B$ resulting from merging A and B (we assume that B has two dummy numbers, $-\infty$ and $+\infty$ for obvious reasons). The cousins in B of each number in A can be determined in $O(n^2)$ time by merging and interval broadcasting (see Section 2.5).

2.4. Two Classes of Parallel Algorithms

Suppose that all the vertices $u_0, u_1, \dots, u_{n!-1}$ in X_n have been ordered such that $u_k < u_j$ if $k < j$. Then, the rank $r(u_k)$ of vertex u_k is $r(u_k) = k$, $0 \leq k \leq n! - 1$. For each vertex u_k , let x_i be $\lfloor k/(i - 1)! \rfloor \bmod i$, $i = 2, 3, \dots, n$, i.e., $0 \leq x_i \leq i - 1$. Thus, each vertex u_k , $0 \leq k \leq n! - 1$, is associated with $n - 1$ unique values x_i , $2 \leq i \leq n$. These values can be viewed as the *address* $x_2 x_3 \dots x_n$ of the vertex u_k . We call x_i the *ith coordinate* of u_k . The coordinate x_i of vertex u_k is simply the number of X_{i-1} 's preceding u_k within the X_i in which u_k is found. From the two assumptions at the beginning of this section, we can see that any i vertices with coordinates $x_2 x_3 \dots x_{i-1} x_i x_{i+1} \dots x_n, x_i = 0, 1, 2, \dots, i - 1$, are "connected" in a linear array of length i in X_n (after a constant time routing), $i = 2, 3, \dots, n$, i.e., these vertices all have the same $n - i$ last symbols and they all belong to the same X_i . If this X_i is arranged as an $i \times (i - 1)!$ array, then they are in the same column. This structure allows us to generalize some algorithms on X_n to the following two classes of highly parallel algorithms: ASCEND and DESCEND. Let OPER be any computation on i numbers on a linear array of length i .

Procedure ASCEND(X_n)

for $i = 2$ to n do

OPER $(x_2 \dots x_{i-1} 0 x_{i+1} \dots x_n, x_2 \dots x_{i-1} 1 x_{i+1} \dots x_n, \dots, x_2 \dots x_{i-1} (i - 1) x_{i+1} \dots x_n)$. ■

In the dual class DESCEND, the loop is changed to run from $i = n$ down to 2.

These two types of algorithms are similar to the usual ASCEND/DESCEND algorithms for the hypercube [15]. The sorting and merging algorithms of Section 2.3 also belong to these two classes. Using the ASCEND and/or DESCEND algorithms with appropriate OPERs, a number of useful basic algorithms can be derived [6]. They are *Translation*, *Reversing*, *Concentration*, and *Distribution*. These operations on X_n are similar to the ones on the hypercube [13]. In all these algorithms, each vertex has a record that includes a destination address. This record is to be permuted to the destination. The permuting is done for all the vertices with records and nonempty destinations. At step i , if the *ith* coordinate of the destination address of a record is not the same as the *ith* coordinate of the current vertex in which this record is found; then, the record is sent to the vertex with the correct *ith* coordinate along an array of length i . By doing this, the destination address is matched coordinate by coordinate. Each OPER can be done in $O(n)$ time since the longest linear array on which OPER works is n . Therefore, all these data permutations can be done in $O(n^2)$ time.

Translation

Suppose that all the vertices $u_0, u_1, \dots, u_{n!-1}$ in X_n have been ordered such that $u_k < u_j$ if $k < j$. In the operation *translation*, given some integer s , vertex u_k has to send its datum to vertex $u_{k+s \pmod{n!}}$ simultaneously for all k , $0 \leq k \leq n! - 1$. Translation is also called *cyclic shift*.

Vertex u_k with rank $r(u_k) = k$ and address $x_2 x_3 \dots x_n$ is to be shifted to a vertex with rank $(r(u_k) + s) \bmod n!$. The address $y_2 y_3 \dots y_n$ for the new vertex can be computed accordingly. Using the new address, a translation can be accomplished by running the ASCEND or DESCEND algorithms, where OPER is sorting in the forward direction and the numbers to be sorted are coordinates of $r(u_k) + s$. If we run the DESCEND algorithm, the odd-even transposition sort will be performed on linear arrays of length, $n, n - 1, \dots, 2$, and the values in the comparisons will be y_n, y_{n-1}, \dots, y_2 , for iterations $i = n, \dots, 3, 2$. In fact, the algorithm performs a cyclic shift on a linear array of size i , $i = n, n - 1, \dots, 2$. On the other hand, if the ASCEND algorithm is run, then the values used in the comparisons are y_2, y_3, \dots, y_n , respectively, for iterations $i = 2, 3, \dots, n$.

We now give a correctness proof for the algorithm for the ASCEND case (the proof for the DESCEND case is similar). We use induction on n . For $n = 2$, the algorithm clearly works on X_2 , which is a linear array of two nodes. Suppose that the ASCEND translation

algorithm works correctly on X_n for any s in $O(n^2)$ time. For a translation of s arbitrary positions on X_{n+1} , by the algorithm and the induction hypothesis, after n steps, the algorithm correctly cyclically shifts $s \bmod n!$ positions for each $X_n(i)$, $1 \leq i \leq n + 1$. In other words, each node with address $x_2x_3 \dots x_nx_{n+1}$ and destination address $y_2y_3 \dots y_ny_{n+1}$ is now in the node with address $y_2y_3 \dots y_nx_{n+1}$. Since in a translation all records are moved the same distance, it follows that at step $n + 1$ a cyclic shift is performed on a linear array of length $n + 1$ with all the nodes on a column having the same i th coordinates, $2 \leq i \leq n$. Hence, no two records will be moved to the same vertex. Since the algorithm works on arrays of length $2 \leq k \leq n$ on X_n , the time complexity is $O(\sum_{k=2}^n k) = O(n^2)$.

In the algorithm for translation, what is done is that at iteration i the i th coordinates, x_i of u_k , and y_i of the vertex $r(u_k) + s$, are matched. Therefore, for some s such that the destination address and the original address for all data differ in only few positions, translation can be done in less time. For example, when s is a multiple of $(n - 1)!$, translation can be done in $O(n)$ time since we need only to do a cyclic shift on a linear array of length n .

Reversing

In X_n , the record in vertex u ranked $r(u)$, $0 \leq r(u) \leq n! - 1$, needs to go to the vertex ranked $n! - 1 - r(u)$. Let $y_2y_3 \dots y_n$ be the address of the vertex ranked $n! - 1 - r(u)$, i.e., the new address after reversing. Reversing is needed when two sorted sequences in the same direction are to be merged; one of them is reversed first, then the regular merging is carried out.

The reversing can also be done by running the ASCEND or DESCEND algorithms, where at iteration i the OPER is the sorting algorithm in the forward direction and the values (keys) used in the sorting algorithm are y_i 's, $i = 2, 3, \dots, n$, for ASCEND, and y_i 's, $i = n, \dots, 3, 2$, for DESCEND. The correctness of the algorithm can be shown in the same way as that for the translation.

Concentration and Distribution

Some vertices of X_n are marked as *active*. The rank $r(u)$ of an active vertex u is the number of active vertices preceding u (note the difference from Definition 2). The records in these active vertices are to be compressed (concentrated) so that they are stored in vertices $0, 1, 2, \dots$, such that the record originally in active vertex u is now in vertex $r(u)$.

Consider the following problem first: Given a linear array of length i , with vertices $0, 1, 2, \dots, i - 1$, some of the vertices are marked as "active." Each active vertex has a record with an integer key k , $0 \leq k \leq i -$

1, and no two keys contained in two different active vertices are the same. The task is to route the record in an active vertex having key k to the k th vertex in the linear array for all the active vertices. This problem can be solved as follows: We first sort the records by their keys in the forward direction (assuming that inactive vertices have keys valued at $+\infty$), i.e., vertex 0 has the smallest key, vertex 1 has the second smallest key, etc. Now the problem becomes a distribution problem (see below for definition) on the linear array that can be solved easily in $O(i)$ time.

A concentration can be done in $O(n^2)$ time by running the ASCEND algorithm. Let an active vertex u have address $x_2x_3 \dots x_n$, and let the address of the vertex $r(u)$ to which the record of u is to be moved be $y_2y_3 \dots y_n$ [computed from $r(u)$]. During iteration i , $2 \leq i \leq n$, the OPER is the same as described in the previous paragraph with keys being y_i 's. The proof of correctness is parallel to that for the hypercube [14]: Suppose that at iteration i a conflict occurs, i.e., two records are moved to the same vertex. Consider a pair of records originating from vertices u and v . Without loss of generality, assume that $u > v$. The rankings are such that $u - v \geq r(u) - r(v)$. Let the address coordinates of $r(u)$ be $y_2y_3 \dots y_n$ and the address coordinates of $r(v)$ be $y'_2y'_3 \dots y'_n$. Since at iteration i two records from u and v are moved to the same vertex, this implies that $r(u)$ and $r(v)$ have the same coordinates: $y_k = y'_k$ for $2 \leq k \leq i$. Thus, $r(u) - r(v) \geq i!$ since there exists a j , $j > i$, such that the j th coordinate of $r(u)$ is not equal to that of $r(v)$. On the other hand, $u - v < i!$. Therefore, $u - v < r(u) - r(v)$, leading to a contradiction.

A distribution is simply the inverse of a concentration and can be done by applying the concentration operation in reverse order, i.e., running the DESCEND algorithm with the same OPER.

Table III illustrates an execution of the algorithm. Initially, in X_4 , the data are stored in vertices 1, 7, 13, 14, 18, 21, and 23 (a vertex is in the range from 0 to 23 in X_4), and their addresses $x_2x_3x_4$'s are 100, 101, 102, 012, 003, 113, and 123. They are to be compressed to vertices $r(1) = 0$, $r(7) = 1$, $r(13) = 2$, $r(14) = 3$, $r(18) = 4$, $r(21) = 5$, and $r(23) = 6$ with $y_2y_3y_4$ addresses equal to 000, 100, 010, 110, 020, 120, 001.

2.5. Interval Broadcasting

In X_n , certain k vertices are marked as leaders l_1, l_2, \dots, l_k , with $l_i < l_j$ if $i < j$, and $k \leq n! - 1$; they possess data that they must share with all the higher numbered vertices (in terms of the processor ordering defined before) up to but not including the next leader, i.e., each marked vertex l_i has to broadcast its message to the interval of vertices between l_i and l_{i+1} . Interval

TABLE III. Concentration on X_4

*	A	*	*	*	*
*	B	*	*	*	*
*	C	D	*	*	*
E	*	*	F	*	G

(a) the original configuration.

A	*	*	*	*	*
*	B	*	*	*	*
C	*	*	D	*	*
E	*	*	F	G	*

(b) after iteration $i = 2$.

A	*	*	*	*	*
*	B	*	*	*	*
*	*	C	D	*	*
G	*	*	*	E	F

(c) after iteration $i = 3$.

A	B	C	D	E	F
G	*	*	*	*	*
*	*	*	*	*	*
*	*	*	*	*	*

(d) after iteration $i = 4$.

broadcasting can be done in $O(n \log n)$ time by running the prefix sums algorithm once, where each leader holds an index (processor rank) as well as its message and each nonleader has initially an index of -1 and a dummy message. Given two messages and their associated indices, the result of applying the binary associative operation of the prefix sums algorithm is as follows: The processor with the smaller index is assigned the larger of the two indices and the message associated with that index.

The interval broadcasting algorithm can be used to accomplish a translation by ± 1 position. We can do so by applying the interval broadcasting algorithm twice. For the case $s = +1$, we first let leaders be even numbered vertices $0, 2, 4, \dots, n! - 2$, then do the interval broadcasting so that their data are shifted to vertices $1, 3, 5, \dots, n! - 1$; in the second time, the leaders are $1, 3, 5, \dots, n! - 3$, and after the interval broadcasting, their data are in $2, 4, 6, \dots, n! - 2$;

and, finally, the content of vertex $n! - 1$ can be routed to vertex 0 in $O(n)$ time, so the total time for the translation is still $O(n \log n)$. For the case $s = -1$, the translation can be done in a similar way except that the ordering of the vertices is reversed, i.e., vertex i becomes vertex $n! - 1 - i$. This idea can be used to do translations by $\pm c$ positions in $O(n \log n)$ time for any constant positive integer c .

2.6. Set Difference

The set difference operation is defined as follows: Given two sorted arrays of numbers A and B , find the set difference $A - B$ (B is not necessarily a subset of A), where A and B are stored in two groups of X_{n-1} 's. This operation can be performed in $O(n^2)$ time as follows: By merging A and B , each number from A finds its cousins in B and checks whether or not it is equal to one of them; those that are not equal to any of their cousins are then compressed to give the final result. If there are repeated numbers in A , by compressing those numbers of A that are not equal to their right neighbor, repetitions are avoided.

2.7. Discussion

In this section, we have developed a number of general algorithms that are basic to designing parallel algorithms on the star and pancake networks. It should be noted that all the algorithms of this section apply not only to X_n , but also to any group of consecutively numbered X_{n-1} 's. For algorithms such as broadcasting and prefix sums that use procedure GROUP-COPY directly, it is not difficult to see this property. For the other algorithms, i.e., those in the classes ASCEND/DESCEND, the property holds because when a group of x consecutively numbered X_{n-1} 's are arranged as an x by $(n - 1)!$ array in row-major order, $1 \leq x \leq n$, the columns of the array also form linear arrays (of length x).

It is also worth pointing out that procedure GROUP-COPY makes the star and pancake networks as powerful as the hypercube when solving a certain class of problems. By this we mean that algorithms for a hypercube of $O(N)$ vertices and a star or pancake of $O(N)$ vertices will have the same time complexity for this class. The latter is characterized by the fact that all of its members can be solved on a hypercube using algorithms of type ASCEND or DESCEND [15]. Two typical such problems are broadcasting and computing prefix sums. Other examples are given in [5].

In the following section, we use these basic algorithms to develop several algorithms for solving computational geometric problems.

3. PARALLEL COMPUTATIONAL GEOMETRY ALGORITHMS

Computational geometry is the branch of computer science concerned with designing efficient algorithms for solving geometric problems, such as problems of *inclusion* (e.g., locating a point in a planar subdivision), *intersection* (e.g., determining all the points, if any, where two polygons intersect), *proximity* (e.g., finding the closest pair among a given set of planar points), and *construction* (e.g., computing the convex hull of a set of points). Over the last 10 years, a considerable amount of work has been done in developing parallel algorithms for solving computational geometric problems [4]. In this section, we use computational geometry as a vehicle for illustrating how the basic algorithms developed in the previous section can be used. Specifically, we present parallel algorithms for the star and pancake networks that solve the convex hull problem as well as other related computational geometric problems. The problem of computing the Voronoi diagram, another important problem in computational geometry, on the star and pancake networks, is addressed in [7].

3.1. Convex Hull Algorithms on the Star and Pancake Networks

A set $S = \{s_1, s_2, \dots, s_n\}$ of points in the plane is given, where each point is represented by its Cartesian coordinates [i.e., $s_i = (x_i, y_i)$]. It is required to find the convex hull of S , denoted $H(S)$, i.e., the smallest convex polygon that includes all the points of S . The corners of $H(S)$ are points of S such that every point of S is either a corner of $H(S)$ or lies inside $H(S)$. Figure 2 shows a set S of points and the convex hull of S .

Divide-and-conquer is a common strategy to find $H(S)$. Given $n!$ planar points stored in X_n , we first sort the points by their x -coordinates. Then, in procedure CONVEX-HULL below, n disjoint convex hulls of $(n - 1)!$ points each are found recursively in parallel in $X_{n-1}(i)$, $1 \leq i \leq n$. These convex hulls are then merged repeatedly until a final convex hull is obtained.

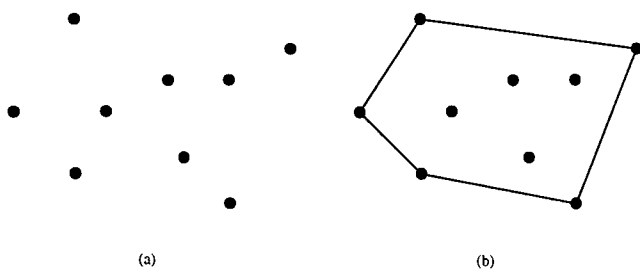


Fig. 2. (a) A set of points S ; (b) the convex hull $H(S)$ of S .

Procedure CONVEX-HULL (X_n)

- 1. **do in parallel for** $1 \leq i \leq n$:
CONVEX-HULL ($X_{n-1}(i)$)
- 2. **for** $j = 1$ **to** $\lceil \log n \rceil$ **do**
 1. Starting with row 1, arrange all rows (X_{n-1} 's) into groups of 2^j consecutively numbered rows (the last group may not have all 2^j rows).
 2. For all the groups **do in parallel**: merge two convex hulls within the group. ■

For example, when computing the convex hull on X_7 , we first recursively compute convex hulls for points stored in $X_6(i)$, for $1 \leq i \leq 7$, in parallel (Step 1 of Procedure CONVEX-HULL). After Step 2.1, there are four groups: $\{X_6(1), X_6(2)\}$, $\{X_6(3), X_6(4)\}$, $\{X_6(5), X_6(6)\}$, and $\{X_6(7)\}$. Two convex hulls in each group are merged in Step 2.2, yielding two groups: $\{X_6(1), X_6(2), X_6(3), X_6(4)\}$, and $\{X_6(5), X_6(6), X_6(7)\}$. Two convex hulls in each group are merged again and finally one group is left: $\{X_6(1), X_6(2), X_6(3), X_6(4), X_6(5), X_6(6), X_6(7)\}$. Once again, two convex hulls in this group are merged to obtain the final result.

We now describe the merge procedure to which we refer henceforth as the *merging slopes technique*. Let $H(P)$ and $H(Q)$ be two disjoint convex hulls of two sets of points P and Q . $H(P)$ and $H(Q)$ are stored in two groups of X_{n-1} 's; they are merged to form $H(P \cup Q)$ by computing two tangents common to $H(P)$ and $H(Q)$. In what follows, all angles are measured with respect to the x -axis.

Definition 3. *The distance of a point to an oriented edge p is the distance from the point to a line containing p ; if the point is to the left (alternatively, right) of p , then the distance is said to be positive (alternatively, negative). The α -distance of a point to p is its distance to the edge p' obtained by rotating p by the angle α in counterclockwise direction around a point (the results of queries discussed below do not depend on the choice of this point).*

Let A and B be two convex polygons in the plane, each containing $O(k(n - 1)!)$ edges stored in groups of $k X_{n-1}$'s, $1 \leq k \leq n - 1$, given in counterclockwise order. Given an angle α , consider the following problem [we call it the *extremal search problem* $ES(A, B, \alpha)$]: For each edge $p \in A$, find a vertex $v_p \in B$ with the smallest α -distance to p among vertices from B (v_p is called an *associated point* of p in direction α). It is easy to see that for $\alpha = 0$ ($\alpha = \pi$) v_p is the vertex with the smallest (greatest) distance from p among vertices of B . For $\alpha = \pi/2$ ($\alpha = 3\pi/2$), v_p is the easternmost (westernmost) point with respect to p .

Let $s(e)$ denote the angle of an edge e . We use the following property of associated points: The associated point $v_p \in B$ (in direction α) of an edge $p \in A$ belongs to an edge $p' \in B$ such that $|s(p) + \alpha - s(t)|$ is minimized on B for $t = p'$. In other words, the associated point of p belongs to an edge that is a cousin of p in B . We now describe the procedure $ES(A, B, \alpha)$. We first increase the angles of edges of A by α . The edges with minimal angles in A and B are recognized and by some translations they are moved to the first vertices of the corresponding groups of X_{n-1} 's. Since angles of edges of both convex polygons are then given in increasing order, the sets A and B can be merged by their angles in $O(n^2)$ time. Now sets A , B , and $A \cup B$ are sorted and each edge e of A can find its cousins in B by interval broadcasting (the last leader broadcasts its data to all the vertices preceding the first leader). We use the unmerging technique to return all edges to their initial positions.

In order to merge $H(P)$ and $H(Q)$, we decide for each of their edges whether it is external or internal edge, i.e., if it is a convex hull edge of $H(P \cup Q)$ as well. To judge if an edge is external, we need to test if $H(P)$ and $H(Q)$ are in the same half-plane bounded by the edge. However, instead of testing all the vertices of $H(Q)$ with an edge e of $H(P)$, we only test two representatives (associated points of e) such that if they are in the same half-plane bounded by e as $H(P)$, then so is every point in $H(Q)$. These two representatives of e in $H(P)$ [e in $H(Q)$] are the nearest and furthest extreme points from $H(Q)$ [$H(P)$] and are obtained by calling procedures $ES(H(P), H(Q), 0)$, $ES(H(P), H(Q), \pi)$, $ES(H(Q), H(P), 0)$, and $ES(H(Q), H(P), \pi)$. Now each edge can decide in constant time if it is external or not. Then, each extreme point of $H(P)$ or $H(Q)$ can learn if it is an extreme point of $H(S)$ (translation by 1 can be used to find the necessary data). Two extreme points in both $H(P)$ and $H(Q)$ share an external and an internal edge. These four points determine two common tangents of $H(P)$ and $H(Q)$. Then, the computation of the circular edge list of $H(S)$ can be done in $O(n^2)$ time by some translations.

The merging procedure takes $O(n^2)$ time and is repeated $O(\log n)$ times. Let $t(n)$ be the time to find the convex hull of $n!$ planar points on X_n ; then, $t(n) = t(n - 1) + O(n^2 \log n) = O(n^3 \log n)$. This performance matches that of the currently fastest known sorting algorithms on S_n and P_n .

3.2. Solving Geometric Problems by the Merging Slopes Technique

Using the merging slopes technique, several other geometric problems can also be solved. In what follows, the size of every problem is $N = O(n!)$. In this section,

we present only the algorithms for finding the smallest enclosing box of a set of points and for finding critical support lines of two convex polygons. Other problems that can be solved on X_n using the merging slopes technique are computing the diameter, width, and minimax linear fit of a set of points; computing the maximum distance between two convex polygons; and computing the vector sum of two convex polygons. Algorithms for these problems are given in [6] whose running time is either $O(n^2)$ or $O(n^3 \log n)$, depending on the problem.

3.2.1. Smallest Enclosing Box

For certain packing and layout problems, it is useful to find a minimum-area rectangle (smallest box) that encloses a set S of N points. This problem has a lower bound of $\Omega(N \log N)$ on the number of sequential steps required to solve it. Any enclosing rectangle must clearly enclose $H(S)$. A smallest enclosing box of S must have one side collinear with an edge of $H(S)$ and each of the other three sides must pass through an extreme point of S [18].

We determine for each edge e of $H(S)$ the minimum area enclosing rectangle that contains a side collinear with e . The smallest enclosing box of S is the minimal over all the obtained enclosing rectangles. To determine the minimum area rectangle for each edge e of $H(S)$, each processor containing an edge e needs to know three additional extreme points of $H(S)$: N , W , and E which are the northernmost, westernmost, and easternmost points (respectively) with respect to the edge e . First, we find the convex hull $H(S)$ of S . Then, the points W , N , and E are obtained by calling the procedures $ES(H(S), H(S), -\pi/2)$, $ES(H(S), H(S), \pi)$, and $ES(H(S), H(S), \pi/2)$, respectively. It is clear that the parallel running time is bounded by the time needed for merging and computing convex hulls and minima of some data. Thus, our algorithm runs in $O(n^3 \log n)$ time on X_n for the set S of size $O(n!)$.

3.2.2. Critical Support Lines of Two Convex Polygons

Given two disjoint convex polygons P and Q , a *critical support line* is a line $L(p(i), q(j))$ such that it is a line of support for P and Q at $p(i)$ and $q(j)$, respectively, and such that P and Q lie on opposite sides of $L(p(i), q(j))$. Critical support lines have applications in a variety of problems such as visibility and collision avoidance [19]. The algorithm to find critical support lines of two convex polygons P and Q is similar to that for merging two convex hulls and, thus, runs in $O(n^2)$ time. We find associated points for each edge from P (respectively, Q) as described in the previous section. Then, each edge of P (respectively, Q) can decide whether the polygon Q (respectively, P) lies completely on one

side of a straight line passing through that edge and P (respectively, Q) on the other side. Now $p(i)$ is determined as the point common to two edges of P , one satisfying the latter property, and the second not satisfying it. Point $q(j)$ is obtained in Q similarly. This defines $L(p(i), q(j))$. Two other points, one in P and one in Q , obtained in the same fashion, define a second critical support line.

4. CONCLUSION

In this paper, we have presented several basic algorithms on the star and pancake interconnection networks. We also introduced the concept of a coordinate system for the vertices of the two networks, in a manner similar to the use of binary strings for the vertices of a hypercube. In addition, two classes of highly parallel algorithms were presented that capture the essence of several data permutations and the general structure of the star and pancake networks. These basic algorithms can be used to obtain efficient solutions to many problems on these two networks. We demonstrate this here by presenting several algorithms for problems in the area of computational geometry. One of these is an algorithm that computes the convex hull of $n!$ planar points on star and pancake networks with $n!$ vertices in $O(n^3 \log n)$ time. This time matches that of the best-known sorting algorithms on these networks. The merging slopes technique, on which the convex hull algorithm is based, together with the basic algorithms, are then applied to obtain solutions to a number of other geometric problems.

With few exceptions, all of the basic and computational geometric algorithms on the star and pancake networks presented in this paper are, to the best of our knowledge, the first such algorithms to appear in the literature. Furthermore, the routing schemes developed here enable us to view the star and the pancake networks as one network. It is these routings that unify the two networks and, consequently, enable us to design algorithms that work on both.

The authors would like to thank the anonymous referees for their comments which improved the presentation of this paper.

REFERENCES

- [1] S. B. Akers and B. Krishnamurthy, A Group theoretic model for symmetric interconnection networks. *IEEE Trans. Comput.* **c-38**(4) (1989) 555–566.
- [2] S. B. Akers, D. Harel, and B. Krishnamurthy, The star graph: An attractive alternative to the n -cube. *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL (1987) 393–400.
- [3] S. B. Akers and B. Krishnamurthy, The fault tolerance of star graphs. (L. P. Kartashev and S. I. Kartashev, Eds.). *2nd International Conference on Supercomputing*, San Francisco (1987) Vol. III, 270–276.
- [4] S. G. Akl and K. A. Lyons, *Parallel Computational Geometry*. Prentice-Hall, Englewood Cliffs, NJ (1993).
- [5] S. G. Akl and K. Qiu, A novel routing scheme on the star and pancake networks and its applications. *Parallel Comput.* **19**(1) (1993) 95–101.
- [6] S. G. Akl, K. Qiu, and I. Stojmenović, Data communication and computational geometry on the star and pancake interconnection networks. Technical Report 91-301, Department of Computing and Information Science, Queen's University, Kingston, Canada (1991).
- [7] S. G. Akl, K. Qiu, and I. Stojmenović, Computing the Voronoi diagram on the star and pancake interconnection networks. *Proceedings of the 4th Canadian Conference on Computational Geometry*, St. John's, Newfoundland (1992) 353–358.
- [8] P. Berthome, A. G. Ferreira, and S. Perennes, Decomposing hierarchical Cayley graphs, with applications to information dissemination and algorithm design. Technical Report LIP 92-38, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Supérieure de Lyon, Lyon, France (1992).
- [9] C. GowriSankaran, Broadcasting on recursively decomposable Cayley graphs. *Discrete Appl. Math.*, to appear.
- [10] D. E. Knuth, *The Art of Computer Programming*. Addison-Wesley, Reading, MA (1973) Vol. 3.
- [11] V. E. Mendia and D. Sarkar, Optimal broadcasting on the star graph. Technical Report, Department of Mathematics and Computer Science, University of Miami (1990).
- [12] A. Menn and A. K. Somani, An efficient sorting algorithm for the star graph interconnection network. *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL (1990) Vol. III, 1–8.
- [13] D. Nassimi and S. Sahni, Data broadcasting in SIMD computers. *IEEE Trans. Comput.* **c-30**(2) (1981) 101–106.
- [14] D. Nassimi and S. Sahni, Parallel permutation and sorting algorithms and a new generalized connection network. *JACM* **29**(3) (1982) 642–667.
- [15] F. P. Preparata and J. Vuillemin, The cube-connected-cycle: A versatile network for parallel computation. *Commun. ACM* **24**(5) (1981) 300–309.
- [16] K. Qiu, H. Meijer and S. G. Akl, Parallel routing and sorting on the pancake network. *Advances in Computing and Information-ICCI '91*, Lecture Notes in Com-

- puter Science, No. 497 (F. Dehne and F. Fiala, Eds.). Springer-Verlag, Berlin (1991) 360–371.
- [17] K. Qiu, S. G. Akl and H. Meijer, On some properties and algorithms for the star and pancake interconnection networks. *J Parallel Distributed Comput.*, to appear.
- [18] I. Stojmenović, Computational geometry on a hypercube. *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL (1988) 100–103.
- [19] G. T. Toussaint, Solving geometric problems with the ‘rotating calipers.’ *Proceedings of the IEEE MELECON’83*, Athens, Greece (May 1983).