Computational Geometry on the Star and Pancake Networks

S.G. Akl K. Qiu
Department of Computing and Information Science
Queen's University, Canada

I. Stojmenovic
Computer Science Department
University of Ottawa, Canada

1. Introduction

The star and pancake networks were proposed in [1,2] as attractive alternatives to the hypercube topology for interconnecting processors in a parallel computer. Each has a rich structure, a small diameter, and many desirable fault tolerance characteristics. In this paper, we present several data communication algorithms for these two networks. These algorithms are then used to solve several computational geometric problems on the star and pancake interconnection networks. One of the results of the paper is that the convex hull of a set of $n!$ planar points can be computed on a star or pancake network with $n!$ nodes in $O(n^3\log n)$ time. This time complexity matches that of the best known sorting algorithm on each of the two networks.

Let V_n be the set of all permutations of symbols 1, 2, ..., n, a *star interconnection network* on n symbols, $S_n=(V_n, E_{S_n})$, is a graph of $n!$ nodes where each node is connected to $n-1$ nodes which can be obtained by interchanging the first symbol of the node with the i^{th} symbol, $2 \le i \le n$. Fig. 1 shows S_4. S_n is also called an n*-star.* A *pancake interconnection network* on n symbols, $P_n=(V_n, E_{P_n})$, is also a graph of $n!$ nodes where each node is connected to $n-1$ nodes which can be obtained by flipping (thus the name *pancake*) the first i symbols, $2 \le i \le n$. Fig. 2 shows P_4. P_n is also called an n*-pancake*. Clearly, $S_n = P_n$, for $n \le 3$. Both S_n and P_n have $O(n)$ diameters, and for any two arbitrary nodes u and v, it is easy to find a path from u to v of length less than $2n$ [1,2]. Since most of our discussion applies to both star and pancake networks, we use X_n to denote either S_n or P_n.

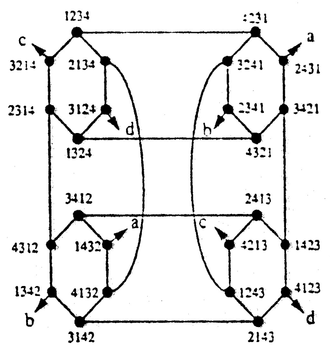

Figure 1. A 4-Star.

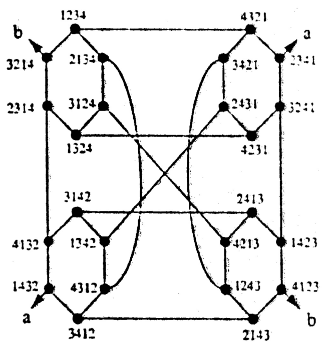

Figure 2. A 4-Pancake.

2. Data Communication algorithms

Definition 1. $X_{n-1}(i)$ is a sub-graph in X_n induced by all the nodes with the same last symbol i, $1 \le i \le n$.

It is easy to see that $S_{n-1}(i)$ is an $(n-1)$-star and $P_{n-1}(i)$ is an $(n-1)$-pancake. For example, S_4 in Fig. 1 contains four 3-stars, namely $S_3(1)$, $S_3(2)$, $S_3(3)$, and $S_3(4)$.

Definition 2. Let $a_1a_2 \cdots a_n$ and $b_1b_2 \cdots b_n$ be two processors (permutations) in X_n, the ordering, $<$, on the processors is defined as follows: $a_1a_2 \cdots a_ia_{i+1} \cdots a_n < b_1b_2 \cdots b_ib_{i+1} \cdots b_n$ if there exists an i, $1 \le i \le n$, such that $a_j = b_j$ for $j > i$, and $a_i < b_i$. In other words, the processors are ordered in reversed lexicographic order (lexicographic order if we read from right to left). In X_n, the *rank* of a node u is the number of nodes v such that $v < u$, i.e. rank $(u) = |\{v \mid v < u, v \in V_n\}|$.

If we write down all the nodes in X_n according to their ranks and arrange them into an $n \times (n-1)!$ array, then row i becomes $X_{n-1}(i)$. The nodes in X_4 are given in Table 1.

Table 1

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

Table 2

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

From the definitions, we can see that all the nodes in the same column of Table 1 have the same rank in their respective X_{n-1}'s. For example, nodes 2431, 1432, 1423, and 1324 are all ranked third in $X_3(1)$, $X_3(2)$, $X_3(3)$, and $X_3(4)$, respectively.

This work is supported by the Natural Sciences and Engineering Research Council of Canada and by Queen's University.

In S_n, if we we exchange the 1st symbol with the last one in each node, we get another $n \times (n-1)!$ array (Table 2) in which each column is connected to form a linear array by the definition of S_n. If nodes in Table 1 are considered in P_4, then we also get Table 2, in which columns are permuted, by flipping the entire nodes, i.e. node $a_1 \cdots a_n$ becomes $a_n \cdots a_1$. But these columns (as in Table 2) are not connected in P_n (for $n \geq 4$) to form linear arrays. A routing scheme developed in [8] enables us to route the data in row i ($P_{n-1}(i)$) to row $i+1$ ($P_{n-1}(i+1)$) in constant time in such a way that if node p is ranked r^{th} in $P_{n-1}(i)$ then its data is sent to node q in $P_{n-1}(i+1)$ which is also ranked r^{th} in $P_{n-1}(i+1)$; that is, p and q are in the same column as in Table 1. Henceforth, we will assume for simplicity that all the columns as in Table 1 are connected either in S_n or in P_n without performing a constant transformation or routing.

Broadcasting. Suppose that one node wants to send the same message to all the other nodes in X_n. A simple $O(n\log n)$ broadcasting algorithm is presented in [4,9] which works for both networks. References [1] and [2] also give $O(n\log n)$ algorithms for S_n and P_n separately.

Prefix Sum, Maximum, Minimum, and Ranking. An $O(n\log n)$ parallel prefix sum algorithm for X_n can be found in [4,9] with respect to the processor ordering in Definition 2. If we let the associative operation in the prefix sum algorithm be *max* or *min*, then the maximum and minimum of $n!$ elements can be found in $O(n\log n)$ time. The final result is reported in the last node $123 \cdots n$. In X_n, some nodes are marked (selected). The rank of a node u is the number of marked nodes that precedes u (compare to Definition 2). The ranks of all the marked nodes can be computed by applying the prefix sum algorithm once, with the associative operation being the usual addition +, each marked node having value 1, and others having value 0.

Sorting and Merging. Given a sequence of processors each holding one element, we say that the sequence is sorted in the F (forward) direction if $x \in p$, $y \in q$, and $p < q$, then $x \leq y$. The R (reverse) direction is defined similarly. Sorting on S_n has been considered in [6] in which an $O(n^3\log n)$ algorithm is given. For sorting on P_n, an $O(n^3\log n)$ algorithm is proposed in [8]. Given two sorted sequences stored in two groups of X_{n-1}'s:

$$A:\ X_{n-1}(i), X_{n-1}(i+1), \ldots, X_{n-1}(k),$$
$$B: X_{n-1}(k+1), X_{n-1}(k+2), \ldots, X_{n-1}(j),$$

$i \leq j$, (A and B do not necessarily contain the same number of X_{n-1}'s), such that A and B are in opposite directions, the results of [6] and [8] imply that they can be merged into sorted sequence in either direction in $O(n^2)$ time. The merging algorithm can be generalized as

```
General_Merging (D)
    1. in parallel Sort the columns in direction D
    2. in parallel Sort each row in direction D
```

which also takes time $O(n^2)$, where D is either F or R. Note that whenever a group of l consecutive X_{k-1}'s are considered, $1 \leq l \leq k \leq n$, it is always arranged into a $l \times (k-1)!$ array such that each row is a X_{k-1} in which nodes are listed in the increasing order according to their ranks. That is, sorting or merging on X_n is reduced, in this way, to sorting on the columns, and these columns are "connected". Since each column is connected as a linear array, the *odd-even transposition sort* [5] can be applied.

Unmerging. Given a sorted list stored in group $C: X_{n-1}(i), X_{n-1}(i+1), \ldots, X_{n-1}(j)$, $i < j$, such that the sorted list is obtained by merging two sorted sublists in two groups

$$A:\ X_{n-1}(i),\ X_{n-1}(i+1), \ldots,\ X_{n-1}(k),$$
$$B: X_{n-1}(k+1),\ X_{n-1}(k+2), \ldots,\ X_{n-1}(j),$$

and each element knows the rank of the node it was in originally before the merging. This rank is the rank of the node in $X_{n-1}(i), \ldots, X_{n-1}(j)$. The problem of unmerging is to permute the list to return each element in C to its original node in A or B. It is the inverse of merging. The problem can be solved by running the General_Merging (F) in reverse order using the given rank information. Let the rank of node u be $r(u)$, then when unmerging, we use the value $\lfloor r(u)/(i-1)! \rfloor \ mod\ i$, $i = 2, 3, \ldots, n$, in the comparisons in the different stages. Since merging takes time $O(n^2)$, so does the unmerging.

Translation (Cyclic Shift). If we order all the nodes in X_n by $u_0, u_1, \ldots, u_{n!-1}$, such that $u_i < u_j$ if $i < j$, in the operation *translation,* node u_i has to send its data to node $u_{i+s\ (mod\ n!)}$ concurrently for $0 \leq i \leq n!-1$, for some s, $1 \leq s \leq n!-1$. Translation is also called cyclic shift.

At the end of unmerging, we introduced values $\lfloor r(u)/(i-1)! \rfloor \ mod\ i$, $2 \leq i \leq n$, for each node. These values can be viewed as the the address (or coordinates) of a node u, $x_2x_3 \cdots x_n$, with x_i being $\lfloor r(u)/(i-1)! \rfloor \ mod\ i$, $2 \leq i \leq n$, and $0 \leq x \leq i-1$. For node u with rank $r(u)$ and address $x_2x_3 \cdots x_n$, it is to be shifted to a node with rank $(r(u)+s)\ mod\ n!$. The address $y_2y_3 \cdots y_n$ for the new node can be computed accordingly. Using the new address, a translation can be accomplished by running the General_Merging (F) algorithm or running it in reverse order with the new addresses used in different stages. If we run the General_Merging (F), the odd-even transposition sort will be performed on linear arrays of length n, $n-1$, ..., 2, the values in the comparisons will be $y_n, y_{n-1}, \ldots, y_2$. They are reversed if General_Merging (F) is run in reverse order. What is done is that at iteration i, the i^{th} coordinates of u and the node it is going to be cyclically shifted to are matched. Note that when $(n-1)! \mid s$, translation can be done in $O(n)$ time since only $y_n \neq x_n$, while $y_i = x_i$, for $2 \leq i \leq n-1$.

Concentrate and Distribute. Some nodes of X_n contain "active" elements. The rank of an active node is the number of active nodes preceding u. These active elements are to be compressed (concentrated) so that they are stored in nodes 0, 1, 2, ..., such that the active element originally in node u is now in node $r(u)$, i.e. the relative positions of active elements to each other remain unchanged. A concentration can be done in $O(n^2)$ time by running the General_Merging (F) in reverse order with values $x_2, x_3, ..., x_n, x_2x_3 \cdots x_n$ being the address of node $r(u)$. A distribution is simply the inverse of a concentration, and can be done by running the Concentrate in reverse order: running the General_Merging (F) with values $x_n, x_{n-1}, ..., x_3, x_2$.

Reversal In X_n, the element in node u ranked $r(u)$, $0 \le r(u) \le n!-1$, needs to go to node ranked $n!-1-r(u)$. This can also be done by running the General_Merging (F) or in reverse order using the new addresses (the addresses after the reversal). Reversal 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.

Interval Broadcasting. In X_n, certain nodes are marked as leaders $l_1, l_2, ..., l_k$, with $l_i < l_j$ if $i < j$, and $k \le n!$; they possess data that they must share with all the higher ranked nodes (in terms of the processor ordering defined before) up to but not including the next leader. That is, each marked node l_i has to broadcast its message to the interval of nodes between l_i and l_{i+1}. Interval broadcasting can be done in $O(n \log n)$ time by running the prefix sum algorithm once, where each leader holds an index (processor rank) as well as its message, and each non-leader has initially an index of -1, i.e. each message is associated with an index, with the associative operation being "take the largest index and the largest indexed value of the two indexed messages".

The interval broadcasting algorithm can be used to accomplish a translation of ± 1 positions. We can do so by applying the interval broadcasting algorithm twice: for the case $s = +1$, we first let leaders be even numbered nodes 0, 2, 4, ..., $n!-2$, then do the interval broadcasting so that their data are shifted to nodes 1, 3, 5, ..., $n!-1$; in the second time, the leaders are 1, 3, 5, ..., $n!-3$ and after the interval broadcasting, their data are in 2, 4, 6, ..., $n!-2$; finally the content of node $n!-1$ can be routed to node 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 the similar way except that the ordering of the nodes is reversed, i.e., node i becomes node $n!-1-i$.

Cousins. 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 elements in B so that a is in between them in sorted lists $A \cup B$. The cousins in B of each element in A can be determined in $O(n^2)$ time on X_n or a group of consecutive X_{n-1}'s by merging and interval broadcasting. The ranks of two cousins b_1 and b_2 from B for an element $a \in A$ are determined by $r(b_1, B) = r(a, A \cup B) - r(a, A)$ and $r(b_2, B) = r(b_1, B) + 1$, where $r(x, Y)$ denotes the rank of an element x in the sorted set X.

It is easy to see that all the algorithms of this section apply not only to X_n, but also to a group of consecutively numbered X_{n-1}'s.

3. Geometric Algorithms on the Star and Pancake Networks

Divide-and-conquer is a common strategy to find the convex hull $H(S)$ of a set of points S. Given $n!$ planar points stored in X_n, we first sort the points by their x-coordinates. Now n disjoint convex hulls of $(n-1)!$ points each are found recursively in parallel in $X_{n-1}(i)$, $1 \le i \le n$. These convex hulls are then merged repeatedly until a final convex hull is obtained. The algorithm is therefore as follows:

```
1. sort the n! points by their x-coordinates
2. Procedure CONVEX-HULL ($X_n$)
     do in parallel for $1 \le i \le n$
          CONVEX-HULL ($X_{n-1}(i)$)
     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. in parallel merge within each group of two sub-groups of $\le 2^{j-1}$ consecutive rows each
   end CONVEX-HULL
```

We now describe the merge procedure based on 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; $H(P)$ and $H(Q)$ are merged to form a bigger convex hull by computing two common tangents of $H(P)$ and $H(Q)$. The following technique of merging is adapted from [10].

Definition 3. The distance of a point to an oriented edge p is the distance from the point to a line containing p, with the distance of point to the left (right) of p being positive (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 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, $1 \le k \le n-1$, 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 of p.

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 $t = p'$ (where $s(e)$ denotes the angle of e; all angles are measured with respect to x-axis). In other words, the associated point of p belongs to 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 nodes 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 nodes 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 an external or internal edge, i.e. if it is a convex hull edge of $H(S)$ 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 $H(P)$, every point in $H(Q)$ is. These two representatives for 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 of them 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.

As we can see, the computation is dominated by merging and translation, which take $O(n^2)$ time, while interval broadcasting takes only $O(n \log n)$ time. This merging procedure is repeated $O(\log n)$ times. If we let $t(n)$ be the time to find the convex hull of $n!$ planar points, then $t(n) = t(n-1) + O(n^2 \log n) = O(n^3 \log n)$. Thus,

Theorem. The convex hull of $n!$ planar points can be computed in $O(n^3 \log n)$ on S_n or P_n with $n!$ processors.

Using the merging slopes technique, many other geometric problems can also be solved. They are, finding critical support lines of two convex polygons, finding the smallest enclosing box, the diameter of a set of points, the width and the minimax linear fit of a set of planar points, the maximum distance between two convex polygons, and the vector sum of two convex polygons. The details of the algorithms and their complexities can be found in [4]. Other algorithms for computational geometric problems can also be found by using our data communication algorithms.

References

[1] S.B. Akers and B. Krishnamurthy, "A Group Theoretic Model for Symmetric Interconnection Networks," *IEEE Trans. on Compu.*, Vol. 38, No. 4, 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*, 1987, pp. 393-400.

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

[4] S.G. Akl, K. Qiu, and I. Stojmenovic, Data Communication and Computational Geometry on the Star and Pancake Interconnection Networks, Tech Rep 91-301, Dept. Comp and Info Sci, Queen's University, April 1991.

[5] D.E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, Massachusetts, 1973.

[6] A. Menn and A.K. Somani, "An Efficient Sorting Algorithm for the Star Graph Interconnection Network," *Proc. International Conference on Parallel Processing*, August 1990, pp. 1-8.

[7] D. Nassimi and S. Sahni, "Data Broadcasting in SIMD Computers," *IEEE Trans. on Compu.*, Vol. c-30, No. 2, 1981, pp. 101-106.

[8] K. Qiu, H. Meijer, and S.G. Akl, "Parallel Routing and Sorting on the Pancake Network," *Proc. International Conf. on Computing and Information*, Ottawa, May 1991 (to appear).

[9] K. Qiu, S.G. Akl, and H. Meijer, The Star and Pancake Interconnection Networks: Properties and Algorithms, Tech Report 91-297, Dept. Computing and Information Sci, Queen's University, Canada, March 1991.

[10] I. Stojmenovic, "Computational Geometry on a Hypercube," *Proc. International Conference on Parallel Processing*, 1988, pp. 100-103.