Computing the Voronoi Diagram on the Star and Pancake Interconnection Networks

S.G. Akl, K. Qiu
Dept. of Computing & Info. Sci.
Queen's University
Kingston, Canada
{akl,qiu}@qucis.queensu.ca

I. Stojmenović
Computer Science Department
University of Ottawa
Ottawa, Canada
stosl@uottawa.bitnet

Abstract

The star and pancake interconnection networks are two attractive alternatives to the popular hypercube for interconnecting processors in a parallel computer. They possess many desirable properties such as small degree and diameter. In this paper, we present parallel algorithms that compute the Voronoi diagram on the star and pancake networks. For an n-star or n-pancake with $p = n!$ processors, given $n!$ planar points stored in the processors such that each processor holds one point and has a memory of constant size, the Voronoi diagram of these points can be found in $O(n^4 \log^2 n)$ time. The algorithm presented in this paper is universal in the sense that it can be used on any network that can efficiently implement the divide-and-conquer paradigm.

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]. Both the hypercube and CCC are Cayley graphs [1].

Let $\mathcal{G}$ be a permutation group and V_n be the set of all $n!$ permutations of symbols 1, 2, ..., n. A *star graph* on n symbols, $S_n = (V_n, E_{S_n})$, is a Cayley graph with generators $g_i = i23...(i-1)1(i+1)...n$, $2 \leq i \leq n$. Fig. 1 shows S_4. For a node $v \in V_n$, we define $v[i]$ to be the i^{th} symbol in v, with $v[1]$ being the leftmost one. Each vertex in S_n is connected to $n-1$ vertices which can be obtained by interchanging the first symbol of the vertex with the i^{th} symbol, $2 \leq i \leq n$. We call these $n-1$ connections *dimensions*. Thus each vertex is connected to $n-1$ vertices through dimensions 2, 3, ..., n. S_n is also called an n-star.

A *pancake interconnection network* on n symbols, $P_n = (V_n, E_{P_n})$, is a Cayley graph with generators $h_i = i(i-1)...321(i+1)(i+2)...n$, $2 \leq i \leq n$. Each vertex in P_n is connected to $n-1$ vertices which can be obtained by flipping the first i symbols, $2 \leq i \leq n$. The dimensions for P_n are defined similarly as for star networks. P_n is also called an n-pancake.

Fig. 2 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 $n \leq 3$.

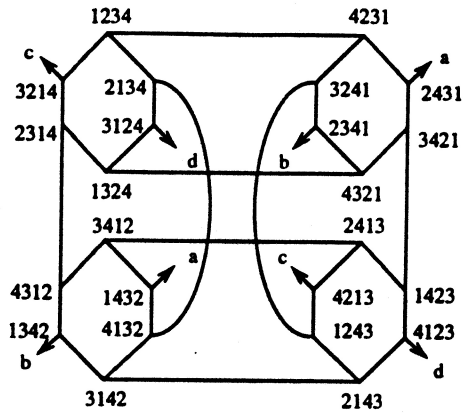

Figure 1: A 4-Star S_4

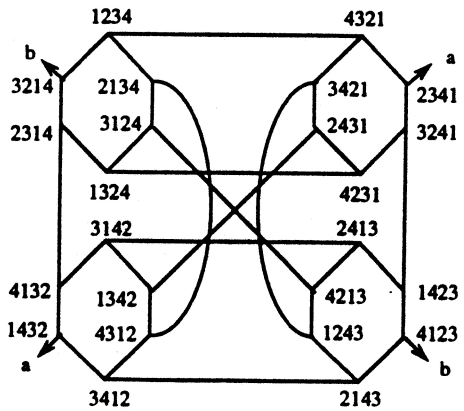

Figure 2: A 4-Pancake P_4

In those cases where our 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, the degree of X_n is $n-1$, i.e., sub-logarithmic in the number of vertices of X_n, while a hypercube with $\Theta(n!)$ vertices has degree $\Theta(\log n!) = \Theta(n \log n)$, i.e., logarithmic in the number of vertices. Similarly, the diameter of X_n is $\Theta(n)$, while a hypercube with $\Theta(n!)$ vertices has a diameter of $\Theta(\log n!) = \Theta(n \log n)$. Other attractive properties include their symmetry properties, as well as many desirable fault tolerance characteristics [1, 2, 3]. Recently, interest in the study of these two networks has grown steadily. For example, Akl, Qiu, and Stojmenovic have studied various data communication algorithms and their applications in developing parallel algorithms for solving problems in the area of computational geometry [4].

In this paper, we present parallel algorithms that compute the Voronoi diagram on the star and pancake networks. A major component and contribution of our Voronoi diagram algorithms is a parallel batched planar point location algorithm that works for the planar subdivision induced by the Voronoi diagram. All of our algorithms are motivated by the ones developed in [11] for the hypercube model. For an n-star or n-pancake with $p = n!$ processors, given $n!$ planar points stored in the processors such that each processor holds one point and has a memory of constant size, the problem of batched planar

point location for a planar subdivision induced by a Voronoi diagram can be solved in time $O(n^3 \log n)$, and the Voronoi diagram of these $n!$ points can be found in $O(n^4 \log^2 n)$ time. It is important to note that our algorithms belong to the divide-and-conquer paradigm and in that sense they can be used on any network that implements this paradigm efficiently.

The remainder of the paper is organized as follows. In Section 2 we briefly review previous results concerning a number of basic data communication algorithms used in our planar point location and Voronoi diagram algorithms. We discuss the problem of planar point location for a subdivision induced by a Voronoi diagram in Section 3. Section 4 presents Voronoi diagram algorithms for S_n as well as P_n.

2 Preliminaries

Definition 1 *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 $1 \leq i \leq n$.*

From the definitions of S_n and P_n, it can be seen easily 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. 1 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.

Definition 2 *In X_n, let p denote the processor associated with the vertex $a_1 a_2 ... a_n$ and q denote the processor associated with the vertex $b_1 b_2 ... b_n$. An ordering, $\prec$, on the processors is defined as follows: $p \prec q$ 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 $p \prec q$, we say that p precedes q.*

Definition 3 *In X_n, the rank $r(u)$ of a vertex u is the number of vertices v such that $v \prec u$, i.e., $r(u) = |\{v | v \prec u, v \in V_n\}|$. Clearly, $0 \leq r(u) \leq n! - 1$.*

Throughout this paper, we assume that in one time unit, a processor can send (receive) at most one datum of fixed length to (from) one and only one of its neighbors.

In what follows, we list a number of results to be used in our algorithms.

Sorting, Merging, and Unmerging: Given $n!$ numbers in X_n with each processor holding one number, they can be sorted in time $O(n^3 \log n)$, with respect to the processor ordering given in Definition 2 [8, 10]. Also, given two sorted sequences stored in two groups of X_{n-1}'s:

$$A: X_{n-1}(i), X_{n-1}(i+1), ..., X_{n-1}(j),$$
$$B: X_{n-1}(k), X_{n-1}(k+1), ..., X_{n-1}(l),$$

$i \leq j < k \leq l$, (A and B do not necessarily contain the same number of X_{n-1}'s), such that A and B are in opposite directions (meaning that one is non-decreasing and the other is non-increasing), they can be merged into a sorted sequence stored in C: $X_{n-1}(i)$, $X_{n-1}(i+1)$, ..., $X_{n-1}(j)$, $X_{n-1}(k)$, $X_{n-1}(k+1)$, ..., $X_{n-1}(l)$, in either direction, in $O(n^2)$ time [4]. Given sequences A, B, and C as defined above, where each element 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 such that each element in C returns to its original vertex in A or B. This problem can also be solved in $O(n^2)$ time [4].

Compression: The problem of *compression* is defined as follows. Some vertices of X_n contain "active" elements. The rank $r(u)$ of an active vertex u is the number of active vertices preceding u (note the difference between this definition and Definition 3). These active elements are to be compressed so that they are stored in vertices 0, 1, 2, ..., with the active element originally in vertex u now in vertex $r(u)$. Compression can be performed in $O(n^2)$ time on X_n [4].

Interval Broadcasting: In X_n, a certain number k of vertices are marked as *leaders* l_1, l_2, ..., l_k, with $l_i < l_j$ if $i < j$, and $k \leq n! - 1$. Each leader possesses a datum that it must share with all the higher numbered vertices (in terms of the processor ordering in Definition 2) up to but not including the next leader. That is, each marked vertex l_i has to broadcast its message to the interval of vertices between l_i and l_{i+1}. Interval broadcasting can be done in $O(n \log n)$ time on X_n [4].

3 Batched Planar Point Location

In order to locate $O(N)$ query points in a planar subdivision defined by $O(N)$ edges and induced by a Voronoi diagram, we use the chain method described by Lee and Preparata [6], a parallelization of which for mesh-connected computers is given in [7]. The two algorithms are used in [11] to obtain an algorithm for the hypercube. The latter has motivated our algorithm for the star and pancake interconnection networks, which we now describe.

A connected planar straight line graph with a finite number of vertices subdivides the plane into nonempty regions of which exactly one is infinite. These regions form a *planar subdivision*. A chain $C = (u_1, u_2, \cdots, u_p)$ is a straight line graph with vertex set $\{u_1, u_2, \cdots, u_p\}$ and edge set $\{(u_i, u_{i+1}) | 1 \leq i \leq p-1\}$. The chain is said to be *monotone* with respect to a straight line l if the orthogonal projections $\{l(u_1), l(u_2), \cdots, l(u_p)\}$ of the vertices of C on l are ordered as $(l(u_1), l(u_2), \cdots, l(u_p))$ [6].

First we sort the regions induced by the Voronoi diagram using the x-coordinates of a number of selected interior points (called *centers*). Then a monotone complete set of chains is defined as in [6, 7, 9]. These chains are nodes of a binary tree whose leaves correspond to regions of the subdivision. Each chain has its own level and index (the rank of the chain among the chains of a given level). Chains may share common edges. If an edge e belongs to more than one chain it belongs to all members of a set (an interval) of consecutive chains. We assign e to the hierarchically highest chain to which it belongs. Each edge e_i is associated with two regions represented by their centers. Since the regions constitute a Voronoi diagram for a set of points, these points are used as the centers of these regions. Because these points are sorted by their x-coordinates, we denote them by u_0, u_1, ..., such that $u_i(x) \leq u_j(x)$ if $i \leq j$. We can represent these centers (thus regions) by the binary representations of their ranks, so the centers are 00...0, 00..01, 00..10, ... Therefore, each edge is associated with two binary numbers. The level and index of each chain and edge are determined in constant time by the following rule as described in [7]: Find the "bit-wise exclusive or", say Ψ, of the two binary numbers of a pair of points associated with edge e_i. The level of e_i, call it l_i, can be obtained from $l_i = \lfloor \log \Psi \rfloor$. The index of the chain to which e_i belongs is given by $[(2's\ complement(2^{l_i}) - 2^{l_i}) \wedge (binary\ representation\ of\ e_i's\ associated\ point)]/2^{l_i+1}$. For example, if e_i is associated with points 0010 and 0101, then $\Psi = 0111$ and $\lfloor \log 0111 \rfloor = (2)_{10}$, so e_i is of level 2. Furthermore, $[(2's\ complement(2^2) - 2^2) = 1100$ - $0100 = 1000$, and $(1000 \wedge 0010)$ (or $1000 \wedge 0101) = 0$; so e_i is indexed as 0 in the chains of level 2.

All edges are now sorted by their level as the primary key, their index as the secondary key, and the y-coordinate of one of their two end-points as the ternary key (the end-point with smaller y-coordinate is chosen). Also, all query points have their level, index, and y-coordinate as their primary, secondary, and ternary keys. Initially, all query points are assigned a level of $\lceil \log N \rceil$ (the highest possible) and an index of 0, and all query points are sorted by their y-coordinates. Following [6], we call a discrimination (of point Z against edge e) the operation of deciding on which side of e the point Z lies. Then, for each level i, from $i = \lceil \log N \rceil$ to $i = 0$ the following steps are performed:

1. Merge the set of edges with the set of query points, using current indices of query points; note that all query points have the same level, equal to i;
2. Perform interval broadcasting to find, for each query point Z, the corresponding edge e against which Z should be discriminated. If the y-coordinate of Z is not between the y-coordinates of the end-points of e, then Z has been discriminated at the previous level. Depending on which side of e Z is , Z calculates the index of the chain against which it should be discriminated at the next level (we refer to this as the new index of the given query point). If Z is to the left of the corresponding edge, we call it a *left* query point, otherwise, we call it a *right* query point. If the current index of Z is k, then the new index is either $2k$ or $2k+1$, for the left and right query points, respectively;
3. Unmerge edges and query points;
4. Re-sort query points by their new indices, by compressing *left* query points and *right* query points separately, and by merging left and right query points by their new indices;
5. Assign the next level to all query points.

All query points will be located in Step 2 when $i = 0$.

In X_n, the height of the binary tree of chains is $\sum_{i=2}^{n} \lceil \log i \rceil$, which is $\Theta(n \log n)$. Thus the number of levels is also $\Theta(n \log n)$. Since all the steps can be done in $O(n^2)$ time, the time complexity of the X_n algorithm is therefore $O((n \log n)n^2) = O(n^3 \log n)$.

4 Voronoi Diagram Algorithms on X_n

An $O(n^4 \log^2 n)$ time algorithm to construct the Voronoi diagram of a set of $n!$ points on X_n with $n!$ processors and constant memory per processor can be obtained by combining the algorithm described in [5] (which solves the problem on a mesh-connected computer), the planar point location technique described in Section 3, and the data communication algorithms described in Section 2.

Given a set T of $N = n!$ planar points for which we want to find the Voronoi diagram $V(T)$, we first sort these N points by their x-coordinates. Let the set of points in $X_{n-1}(i)$ be T_i, $1 \leq i \leq n$. First, $X_{n-1}(i)$ finds the Voronoi diagram $V(T_i)$ for the points in T_i, recursively, and in parallel for $1 \leq i \leq n$. Then these n $V(T_i)$'s are merged $\lceil \log n \rceil$ times to form $V(T)$. The algorithm is given by the following procedure.

Procedure VD(X_n)

- Sort $n!$ points by their x-coordinates;
- **do in parallel for** $1 \leq i \leq n$: VD ($X_{n-1}(i)$)
- **for** $j = 1$ **to** $\lceil \log n \rceil$ **do**
  1. Starting with 1, arrange all ($X_{n-1}(i)$'s) into groups of 2^j consecutively numbered X_{n-1}'s (The last group may not have 2^j X_{n-1}'s).
  2. for all the groups **do in parallel:** merge two Voronoi Diagrams within the group. □

For example, when $n = 7$, and $j = 1$, we have four groups

$$\begin{array}{lll} Group1: & X_6(1) & X_6(2) \\ Group2: & X_6(3) & X_6(4) \\ Group3: & X_6(5) & X_6(6) \\ Group4: & X_6(7) & \end{array}$$

Let L and R be two sets of points such that $L \cap R = \emptyset$, all points in L and R are sorted by their x-coordinates, and for any $u \in L$, $v \in R$, $u(x) < v(x)$. Suppose we have found $V(L)$ and $V(R)$, respectively. $V(L)$ and $V(R)$ are merged using the algorithm of Jeong and Lee [5], except that we use our batched planar point location algorithm described in Section 3. Note that when doing the batched planar point location, for query points in R (L) in regions of $V(L)$ ($V(R)$), the points of L (R) are chosen to be centers for regions of $V(L)$ ($V(R)$).

It should be noted here that in sequential computation, the Voronoi diagram of N points can be computed optimally in $O(N \log N)$ time [9]. This directly leads to a lower bound of $\Omega(n \log n)$ on the running time of any parallel algorithm that computes the Voronoi diagram of $N = n!$ points using $n!$ processors. Whether this lower bound can be matched by an algorithm for computing the Voronoi diagram on X_n is an open question.

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*, St. Charles, Illinois, August 1987, pp. 393-400.

[3] S.B. Akers and B. Krishnamurthy, "The Fault Tolerance of Star Graphs," Ed. L.P. Kartashev and S.I. Kartashev, 2^{nd} *International Conference on Supercomputing*, Vol. III, San Francisco, May 1987, pp. 270-276.

[4] S.G. Akl, K. Qiu, and I. Stojmenović, "Data Communication and Computational Geometry on the Star and Pancake Interconnection Networks," *Proc.* 3^{rd} *IEEE Symposium on Parallel and Distributed Processing*, Dallas, Dec. 1991, pp. 415-422.

[5] C.S. Jeong and D.T. Lee, "Parallel Geometric Algorithms on Mesh-Connected Computers," *Algorithmica*, Vol. 5, No. 2, 1990, pp. 155-177.

[6] D.T. Lee and F.P. Preparata, "Location of Point in a Planar Subdivision and Its Applications," *SIAM J. Compu.*, Vol. 6, No. 3, September 1977, pp. 594-606.

[7] M. Lu, "Constructing the Voronoi Diagram on a Mesh-Connected Computer," *Proc. International Conference on Parallel Processing*, St. Charles, Illinois, August 1986, pp. 806-811.

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

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

[10] K. Qiu, H. Meijer, and S.G. Akl, "Parallel Routing and Sorting on the Pancake Network," *Proc. International Conference on Computing and Information*, Lecture Notes in Computer Science 497, Springer-Verlag, pp. 360-371, 1991.

[11] I. Stojmenović, "Computational Geometry on a Hypercube," *Proc. International Conference on Parallel Processing*, St. Charles, Illinois, August 1988 pp. 100-103.