Communication-Efficient Deterministic Parallel Algorithms for Planar Point Location and 2d Voronoi Diagram*

Mohamadou Diallo[1], Afonso Ferreira[2] and Andrew Rau-Chaplin[3]

[1] LIMOS, IFMA, Campus des Cézeaux, BP 265, F-63175 Aubière Cedex, France. E-mail: mdiallo@ifma.fr.

[2] CNRS, INRIA, Projet SLOOP, BP 93, 06902 Sophia Antipolis, France. E-mail: ferreira@sophia.inria.fr.

[3] Faculty of Computer Science, Dalhousie University, P.O. Box 1000, Halifax, Nova Scotia, Canada B3J 2X4. E-mail: arc@tuns.ca. Partially supported by NSERC.

Abstract. In this paper we describe deterministic parallel algorithms for planar point location and for building the Voronoi Diagram of n co-planar points. These algorithms are designed for BSP-like models of computation, where p processors, with $O(\frac{n}{p}) \gg O(1)$ local memory each, communicate through some arbitrary interconnection network. They are communication-efficient since they require, respectively, $O(1)$ and $O(\log p)$ communication steps and $O(\frac{n \log n}{p})$ local computation per step. Both algorithms require $O(\frac{n}{p}) = \Omega(p)$ local memory.

1 Introduction

High performance computing systems nowadays, be either multicomputers or networks of workstations, consist of a set of p *state-of-the-art* processors, each with considerable local memory, connected to some interconnection network. These systems are usually *coarse grained*, i.e. the size of each local memory is "considerably larger" than $O(1)$.

Recently, there has been a growing interest in coarse grained computational models [4, 7, 16] and the design of coarse grained algorithms [10, 5, 8, 6, 9]. The work on computational models has tended to be motivated by the observation that "fast algorithms" for fine-grained models rarely translate to fast code running on coarse grained machines. The BSP model ([16]) was proposed in order to benefit from slackness in the number of processors and memory mapping via hash functions to hide communication latency and provide for the efficient execution of fine grained PRAM algorithms on coarse grained hardware. Other coarse grained models focus more on utilizing existing sequential code and minimizing global communication operations. These include the Coarse Grained Multicomputer (CGM) from [7] used in this paper and to be described below. In this

* Part of this work was done while the second author was with the LIP at the ENS Lyon and while the authors visited each other in Lyon and in Halifax. Support from the respective Institutions is acknowledged.

mixed sequential/parallel setting, there are three important measures of any coarse grained algorithm, namely, the amount of local computation, the number and type of global communication phases required and the scalability of the algorithm, that is, the range of values for the ratio $\frac{n}{p}$ for which the algorithm is efficient and applicable.

This paper describes efficient scalable parallel algorithms for the planar point location and the 2d Voronoi diagram problems within the coarse grained multicomputer context. The planar point location algorithm requires local storage $\frac{n}{p} = \Omega(p)$ and is optimal with respect to local computation $(O(\frac{n \log n}{p}))$ and communication phases $(O(1))$. This algorithm is then used as a procedure in the Voronoi Diagram algorithm, which also requires local storage $\frac{n}{p} = \Omega(p)$, but uses $\lceil \log p \rceil$ communication phases with $O(\frac{n \log n}{p})$ local computation per phase.

The Model

The *Coarse Grained Multicomputer* model, or $CGM(n,p)$ for short, can be seen as a weak-CREW BSP machine ([7, 10]). On a $CGM(n,p)$ a problem of size n is solved using p processors each with a local memory of size $O(\frac{n}{p})$. The processors communicate through some arbitrary interconnection network or a shared memory. The term "*coarse grained*" refers to the fact that (as in practice) the number of words of each local memory $O(\frac{n}{p})$ is defined to be "considerably larger" than $O(1)$. This is clearly true for all currently available coarse grained parallel machines. In the following, when determining time complexities both local computation time and inter-processor communication time are considered in the standard way. Also note that we assume, for clarity of explanation, that $p = 2^k$ for some fixed integer k.

In this model, all global communications are performed by a small set of standard communications operations - Segmented broadcast, Segmented gather, All-to-All broadcast, Personalized All-to-All broadcast, Partial sum and Sort, which are typically efficiently realized in hardware or system level code. If a parallel machine does not provide these operations they can be, in the worst case, implemented in terms of a constant number of sorting operations [7].

Furthermore, it was shown that, given $n^{1-\frac{1}{c}} > p$ $(c \geq 1)$, sorting $O(n)$ elements distributed evenly over p processors in the CGM, BSP or LogP models can be achieved in $O(\log n / \log(h+1))$ communication rounds and $O(n \log n/p)$ local computation time, for $h = \Theta(\frac{n}{p})$, i.e. with optimal local computation and $O(1)$ h-relations, when $\frac{n}{p} = \Omega(p)$ [10]. Therefore, using this sort, the communication operations of the $CGM(s,p)$ can be realized in the BSP or LogP models in a constant number of h-relations, where $h = \Theta(\frac{s}{p})$.

Hence, finding an optimal algorithm in the CGM model is equivalent to minimizing the number of global communication rounds as well as the local computation time. It has also been shown that minimizing the number of rounds also results in improved portability across different parallel architectures [16].

Previous Work

Many algorithms (sequential or parallel) have been proposed for solving the multi-planar point location problem [1, 13], where $O(n)$ query points are located in a planar convex subdivision with n vertices. The sequential complexity of the problem is $\Theta(n \log n)$ time with $O(n)$ space. In the fine grained parallel setting, algorithms have been described for many architectures including the CREW PRAM [3], the Hypercube [15] and the Mesh [12]. Except for the PRAM, these algorithms are not work-optimal (time in $O(\sqrt{n})$ and $O(\log^2 n)$ for the Mesh and the Hypercube, respectively).

For the Voronoi diagram (see Figure 1), whose sequential complexity is $\Theta(n \log n)$, the only time-optimal parallel algorithm (although not work-optimal since it runs in $O(\sqrt{n})$ time with n processors) was proposed in [12] for the Mesh. The same technique (to be explored further in this text) was used in [15] to design a $O(\log^3 n)$ time algorithm for the Hypercube. Finally, the best existing PRAM algorithm requires $O(\log n \log\log n)$ time. With respect to the CGM, no efficient deterministic algorithm exist. The *randomized* algorithm from [5] builds the Voronoi diagram in time $O(\frac{n \log n}{p})$, *with high probability*, and requires $n/p = \Omega(p^2)$.

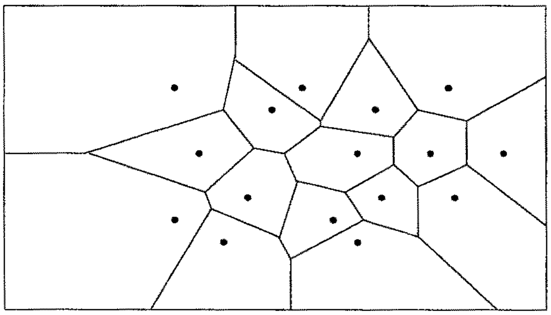

Fig. 1. Voronoi diagram of a set of points.

Our Results

In this paper we first describe a scalable coarse-grained **deterministic** algorithm for the Convex Planar Multi-Point Location problem. Our algorithm requires time $O(\frac{n \log n}{p})$ in the worst case. Furthermore, it requires only a constant number of global communication rounds and local memory space $\frac{n}{p} = \Omega(p)$ to locate $O(n)$ query points in a planar convex subdivision with n vertices.

Using this algorithm as a subprocedure, we propose an algorithm for solving the Voronoi diagram problem for points in $\mathcal{R}^2$ for the same model, with the same space complexity, $\lceil \log p \rceil$ communication rounds, and $O(\frac{n \log n}{p})$ local computation per round. Our algorithm is deterministic and is also more scalable than the algorithm given in [5] in that it is efficient and applicable for a larger range of values for the ratio n/p.

Our approach (which is very different from the one presented in [2, 11]) presents two particular strengths. First, all inter-processor communications are restricted to usages of a small set of simple communication operations. This has the effect of making the algorithms both easy to implement, in that all communications are performed by calls to a standard highly optimized communication library, and very fast in practice. Second, most of the local computation is done through well known algorithms designed for the very same problems. Therefore, costs associated with software development are largely reduced.

2 Planar Point Location

The problem of planar multi-point location on a convex subdivision is stated as follows: Locate $O(n)$ points in a planar convex subdivision defined by $O(n)$ edges. Each edge is labeled with the regions to its left and its right, and regions are defined by coordinates of one interior point (called the center of the region).

To locate a point in the planar subdivision, we design a coarse-grained algorithm based on the chain method originally described in the sequential setting ([13]) and then utilized in the fine-grained parallel setting for MCC ([12]) and hypercubes ([15]). The idea is to perform planar point location via a binary search on a balanced binary tree whose nodes represent a chain of edges of the planar subdivision. The tree is built as follows.

First the regions are sorted by x-coordinate of their centers. There is a chain of edges which share half regions to left and half to right (left and right regions correspond to centers lying to left or right of the chain). The same is applied to left and right half of regions recursively and a monotone complete set of chains is obtained (i.e. the set of chains so that for any two chains c_1 and c_2 the vertices of c_1 that are not on c_2 are on the same side of c_2). These chains are the nodes of the balanced binary tree mentioned above.

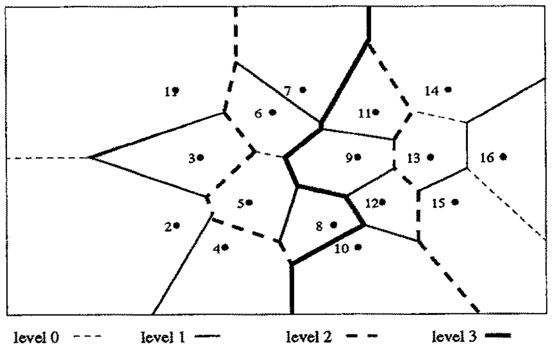

Fig. 2. Construction of the chains.

The leaves of this tree correspond to regions of the subdivision (see Figure 2). Chains may share common edges. If an edge e belongs to more than one chain then it belongs to all members of a set of consecutive chains. There is a unique member c of this set which, in the binary search tree, is a common ascendant of all the other members of the set (the highest chain, in the hierarchy, to which e belongs). In order to avoid duplication of edges, we assign e to such a member c. By $O(\log n)$ discriminations (deciding on which side of chain c a query point lies) each query point can be located.

Each chain has a level and an index. The level of a chain is the height were the chain is located in the tree (the root has the highest level). The index is the rank of the chain in the chains of a given level, ranked from left to right. And as described above, each edge is assigned to exactly one chain. The level and the index of an edge are those of the chain to which it belongs to. The levels and indices of the edges can be determined in constant time using the rules described in [12]: for a given edge e, find the "bit exclusive or", say ψ, of the binary indices of centers of e. The level of e, say l_e, is $l_e = \lfloor \log \psi \rfloor$. The index of e is $((2's\ complement(2^{l_e}) - 2^{l_e}) \wedge (index\ of\ center\ of\ e))/2^{l_e+1}$.

2.1 Coarse-Grained Planar Multi-Point Location

We describe in this subsection a planar multi-point location algorithm that requires a constant number of communication rounds. The entire data for a given problem is assumed to be initially distributed across the local memories and remains there until the problem is solved. Given a set Q of n query points, a planar convex subdivision of the plane into n regions (e.g. a Voronoi diagram) and a p processor coarse grained multicomputer we show how to locate the query points into the subdivision. The basic approach is as follows:

1. Divide the plane into the p regions or vertical slabs (Figure. 3) $V_1, V_2, \ldots V_p$ defined by the $p-1$ highest level chains.
2. For each point $q \in Q$ determine $vs(q) \in \{V_1, V_2, \ldots V_p\}$ the vertical slab q is located in. This is done by forming horizontal slabs (Figure. 3) from the chains computed in Step 1 and performing a point location within these horizontal slabs after first having load balanced the points and slabs.
3. Finally, load balance the vertical slabs and the points such that each processor stores $O(1)$ vertical slabs of total size $O(n/p)$ and $O(n/p)$ points that must be located in them. Locally execute planar multi-point location on all processors.

The main challenge lies in computing for each point which vertical slab it is in (Step 2) in a constant number of communication phases and under the constraint given by the memory size. The idea will be to partition the vertical slabs into p horizontal slabs that are bounded by lines rather than polygonal chains. Our Planar Multi-Point Location algorithm is described in detail below.

CGM's Planar Multi-Point Location(Q, $Vor(S)$)

Input: A set Q of $O(n)$ query points and a planar subdivision defined by $O(n)$ edges.
Output: The $O(n)$ query points labeled by the center of the region to which they belong.

1. For each edge, determine to which chain it belongs using the method by Jeong [12]. The method involves sorting the regions' centers by their x-coordinate. Recall that using this method, each edge belongs to only one chain. Note that we are only interested in the $p-1$ higher level chains, these chains partition the plane into p "vertical" slabs $V_1, V_2, \dots V_p$ (Figure. 3). Let C denotes the set of the edges that define the $p-1$ chains.
2. Sort the edges in C by their largest y-coordinates. Each processor i receives $O(n/p)$ edges denoted H_i and can determine a horizontal line that defines its upper boundary by looking for the largest received y-coordinate (Figure. 3). Perform an all-to-all broadcast of these horizontal lines so that every processor stores a copy of H, the set of these p horizontal lines.
3. Each processor determines for each edge $c \in C$ it stores the elements of H it intersects, denoted $range(c)$. Note that, because the chains are y-monotonic, $range(c)$ is a (contiguous) interval that can be computed by binary search in H, each edge is intersected by at most p horizontal lines and each element of H intersects at most p elements of C. Perform a personalized all-to-all broadcast such that each edge c, for which $range(c) = [i, j]$ is not empty, is broadcast to processors i through j.
4. For each point $q \in Q$ determine $hs(q) \in \{H_1, H_2, \dots H_p\}$ the horizontal slab q is located in and for each horizontal slab H_i, compute $C(H_i) = \lceil \frac{|\{q \in Q : hs(q) = H_i\}|}{\frac{n}{p}} \rceil$, for $1 \leq i \leq p$. Create $C(H_i)$ copies of H_i and distribute them such that each processor stores at most two horizontal slabs. Redistribute Q such that each point $q \in Q$ is stored on a processor that also stores a copy of $hs(q)$.
5. Each processor locally executes Kirkpatrick's planar multi-point location algorithm ([14]). When a point is located to the right or the left of an edge, the vertical slab to which it belongs, $vs(q)$ is obtained by consulting the rank of the center of the region associated to the edge, in the sorted list.
6. For each vertical slab V_i, compute $C(V_i) = \lceil \frac{|\{q \in Q : vs(q) = V_i\}|}{\frac{n}{p}} \rceil$, for $1 \leq i \leq p$. Create $C(V_i)$ copies of V_i and distribute them such that each processor stores at most two vertical slabs. Redistribute Q such that each point $q \in Q$ is stored on a processor that also stores a copy of $vs(q)$
7. All processors now locally execute Kirkpatrick's planar multi-point location ([14]). The location is done in the vertical slab into which the points are located and each point is now precisely located.

Theorem 1. *Algorithm* **CGM's Planar Multi-Point Location()** *locates $O(n)$ query points in a planar convex subdivision defined by $O(n)$ edges in $O(\frac{n \log n}{p})$ time. It requires $\frac{n}{p} = \Omega(p)$ local memory space and a constant number of communication rounds.*

Proof. The correctness follows from the correctness of the chain method described in [12], the correctness of Kirkpatrick's sequential planar multi-point

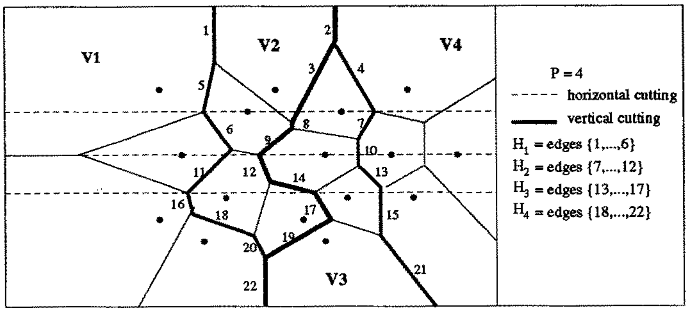

Fig. 3. Horizontal and vertical cuttings.

location method [14, pages 56-58], and the following observations. (1) Both the vertical and horizontal slabs have a size of $O(n/p)$. (2) The total number of slabs created in Steps 4 and 6 is $O(p)$. (3) The total number of queries moved in steps 4 and 6 is $O(n/p)$. The space requirement is thus $O(\frac{n}{p}+p) = O(\frac{n}{p})$ per processor. In each step, the local computation time is at most $O(\frac{n}{p}\log n)$. The global communication in each step reduces to a constant number global sorts and communications operations.

3 Building a 2D-Voronoi Diagram on a CGM

In this section, we first present an algorithm for merging two Voronoi diagrams on a $CGM(n,p)$ which requires only $O(1)$ communication phases and then show how this algorithm can be used to help build the Voronoi diagram of a set of 2d-points through a divide-and-conquer approach. The merge algorithm in turn uses the planar multi-point location algorithm described in the previous section as a basic subprocedure.

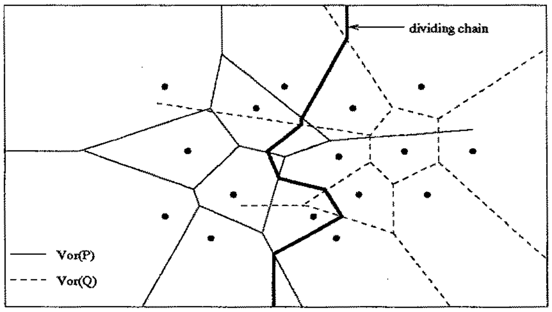

Fig. 4. The dividing chain.

Let a set S of n points (the center of each region) in the plane be given and

P and Q be two disjoint subsets of S, of size $\frac{n}{2}$ each, such that all points of P are located to the left of all points of Q. Suppose that the Voronoi diagrams of P and Q are known and denoted by Vor(P) and Vor(Q) , respectively. Finally, suppose that Vor(P) and Vor(Q) are each represented by a set of edges distributed evenly over $p/2$ processors.

Our merging algorithm follows the scheme described in [12] in which a dividing chain between two Voronoi diagrams is computed (see Figure 4). Since the problem is analogous with respect to P or Q, we will describe the details of the merging from only the point of view of P. The following are the main steps of the algorithm.

```
Two-Way Merging(Vor(P) ,Vor(Q) )

Input: A distributed representation of Vor(P) and Vor(Q) , each over $\frac{p}{2}$ processors.
Output: A distributed representation of Vor($P \cup Q$) over $p$ processors.

 1. Partition the edges of Vor(P) into three sets:
    (a) $PP$, those that have both their endpoints closer to $P$ than to $Q$,
    (b) $PQ$, those that have one of their endpoints closer to $P$ than to $Q$, and the
        other one closer to $Q$ than to $P$.
    (c) $QQ$ those that have both their endpoints closer to $Q$ than to $P$.
 2. For each of the sets found above, decide which edges are intersected by the di-
    viding chain (actually the problem is just for $QQ$).
 3. Compute the new endpoints for the edges that are intersected by the dividing
    chain (intersection point with the dividing chain) and discard the portion of the
    edge laying in the wrong side.
 4. Globally sort all the newly generated endpoints (of the edges of Vor(P) and
    Vor(Q)) in order to obtain the edges of the dividing chain (for the infinite rays,
    it suffices to look at the two points, one in P and the other one in Q, that are
    closer to their finite endpoint to find their slope).
 5. Perform Steps 1 through 4, analogously, with respect to Vor(Q) .
 6. All the current edges form Vor(S) . Distribute them over the p processors.
```

Theorem 2. *Given two sets P and Q of $\frac{n}{p}$ points in the plane, $P \cup Q = S$, such that all points in P are on the left of all points in Q, and a distributed representation of the two Voronoi diagrams Vor(P) and Vor(Q) , each distributed over $\frac{p}{2}$ processors, then algorithm* **Two-Way Merging()** *merges Vor(P) and Vor(Q) to form Vor(S) in $O(\frac{n \log n}{p})$ time. It requires $\frac{n}{p} = \Omega(p)$ local memory space and a constant number of communication rounds.*

Proof. We now consider the correctness and complexity of each step of the algorithm **Two-Way Merging(Vor(P) ,Vor(Q))**.

- Step 1: Partitioning of the edges into the sets PP, PQ and QQ can be computed for the finite edges by performing a planar point location of the endpoints of the edges. For the semi-infinite edges, it has been established [12] that, if all the semi-infinite edges of Vor(Q) are sorted by their slope δ, then for the infinite endpoint v_i and the semi-infinite edges e_i of Vor(P) , and two consecutive semi-infinite edges e_j and e_{j+1} of Vor(Q) , v_i is laying in the unbounded region bordered by e_j and e_{j+1} if and only if $\delta_{e_j} \leq \delta_{e_i} \leq \delta_{e_{j+1}}$.

Using this result, we can find the center of the region, in Vor(Q) , containing the endpoint at infinity and thus see to which set it is closer to by just computing the bisector between the closest point in P and the closest one in Q and then see if the semi-infinite edge crosses this bisector.

Hence, the time complexity of this step is also dominated by calls to the planar point location algorithm, that is $O(\frac{n \log n}{p})$.

- Step 2: In [12] it was also shown that the edges in PP do not cross the dividing chain, the edges in PQ cross it once, and for the edges in QQ we have two cases: if they cross the dividing chain they cross it twice, or else they do not cross it at all (see Figure 5). A simple technique to distinguish these two cases involves again a planar point location. The point location concerns, for each edge of QQ, a unique and precise point X on the concerned edge. Each edge which is determined to be intersected twice is split into two edges of type PQ at the point X. For and edge e in QQ, X is the intersection point between e and the horizontal line passing through one of the centers of the two regions associated to e. The chosen center is the one with the greatest x-coordinate. Here again, the time complexity of this step is also dominated by calls to the planar point location algorithm, that is $O(\frac{n \log n}{p})$.
- Step 3: Compute one intersection point per edge since the edges that are intersected twice are now split into two edges of type PQ. The computation of the intersection point can be done in constant time by computing the bisector between the point in P (the one with the greatest x-coordinate) closest to the first endpoint and the point in Q closest to the second endpoint, and then compute the intersection of the edge with this bisector.
- Step 4: A global sort. Once the new endpoints are sorted (using their y-coordinate as principal key), the dividing chain is built. Recall that this chain is y-monotonic that it is crossed at most once by all horizontal lines. The time complexity of this step is thus $T_s(n,p)$.
- Step 5: A communication phase in which the newly built dividing chain is distributed over the appropriate processors.

Note that all of the steps consist of at most $O(\frac{n \log n}{p})$ local computation and a constant number of calls to the planar point location algorithm, therefore the time complexity follows. The correctness follows from [12].

Using **Two-Way Merging()** we can now easily describe a CGM algorithm for building the Voronoi diagram. Recall that $p = 2^k$ for some integer k.

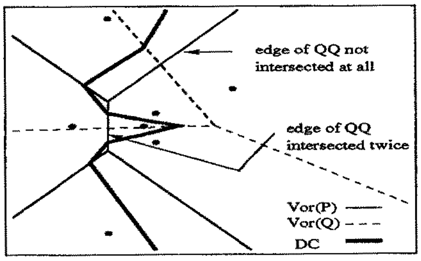
edge of QQ not intersected at all
edge of QQ intersected twice
Vor(P)
Vor(Q)
DC

Fig. 5. Example of an edge that is intersected twice.

Voronoi diagram(S)

Input: Each processor stores a set of $\frac{n}{p}$ points drawn arbitrarily from S.
Output: A distributed representation of the Voronoi diagram of S.

```
1. Globally sort the points in $S$ by x-coordinate. Let $S_i$ denote the set of $\frac{n}{p}$ sorted
   points now stored on processor $i$.
2. Independently and in parallel, each processor $i$ computes the Voronoi diagram of
   the set $S_i$. Let Vor($S_i^1$) denote the result on processor $i$.
3. For $j = 1$ to $\log p$ in parallel do
     Vor($S_i^{j+1}$) $\leftarrow$ Two-Way Merging(Vor($S_{2i}^j$), Vor($S_{2i+1}^j$)), ($i$ from 0 to $\frac{p}{2^j} - 1$).
```

We have therefore proved the second main result of this paper:

Theorem 3. *Algorithm* **Voronoi diagram()** *computes the Voronoi diagram of a set S of n points in the plane, Vor(S) , on a CGM(n,p). It requires $\frac{n}{p} = \Omega(p)$ local memory space, $\lceil \log p \rceil$ communication rounds, and $O(\frac{n \log n}{p})$ local computation time per round.*

References

1. S. Akl and K. Lyons. *Parallel Computational Geometry.* Prentice Hall, 1993.
2. M. Atallah and J. Tsay. On the parallel-decomposability of geometric problems. In *Proceedings of the 5th Annual ACM Symposium on Computational Geometry*, pages 104–113, 1989.
3. R. Cole, M. Goodrich, and C. Dunlaing. Merging free trees in parallel for efficient voronoï diagram construction. In *17th ICALP, England*, July 1990.
4. D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subrarnonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Fifth ACM SIGPLAN Symposium on the Principles and Practice of Parallel Programming*, 1993.
5. F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. Khokhar. A randomized parallel 3d convex hull algorithm for coarse grained multicomputers. In *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, pages 27–33, 1995.
6. F. Dehne, A. Fabri, and C. Kenyon. Scalable and archtecture independent parallel geometric algorithms with high probability optimal time. In *Proceedings of the 6th IEEE SPDP*, pages 586–593. IEEE Press, 1994.

7. F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. In *ACM 9th Symposium on Computational Geometry*, pages 298–307, 1993.
8. X. Deng and N. Gu. Good algorithm design style for multiprocessors. In *Proc. of the 6th IEEE Symposium on Parallel and Distributed Processing, Dallas, USA*, pages 538–543, October 1994.
9. A. Ferreira, A. Rau-Chaplin, and S. Ubéda. Scalable 2d convex hull and triangulation for coarse grained multicomputers. In *Proc. of the 6th IEEE Symposium on Parallel and Distributed Processing, San Antonio, USA*, pages 561–569. IEEE Press, October 1995.
10. M. Goodrich. Communication-efficient parallel sorting. In *Proc. of the 28th annual ACM Symposium on Theory of Computing Philadephia, USA*, May 1996.
11. M. Goodrich, J. Tsay, D. Vengroff, and J. Vitter. External-memory computational geometry. *Proceedings of the Symposium on Foundations of Computer Science*, 1993.
12. C. Jeong. An improved parallel algorithm for constructing voronoï diagram on a mesh-connected computer. *Parallel Computing*, 17:505–514, 1991.
13. D.T. Lee and F. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal on Computing*, 6(3):594–606, 1977.
14. F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985.
15. I. Stojmenovic. Computational geometry on a hypercube. Technical report, Computer Science Dpt., Washington State University, Pullman, Washington 99164–1210, 1987.
16. L. Valiant. A bridging model for parallel computation. *Communication of ACM*, 38(8):103–111, 1990.