Parallel Construction of Subdivision Hierarchies†

N. Dadoun
D.G. Kirkpatrick

Technical Report 87-13
May 1987

Abstract

A direct, simple and general parallel algorithm is described for the preprocessing of a planar subdivision for fast (sequential) search. In essence, the hierarchical subdivision search structure described by Kirkpatrick [K] is constructed in parallel. The method relies on an efficient parallel algorithm for constructing large independent sets in planar graphs. This is accomplished by a simple reduction to the same problem for lists.

Applications to the manipulation of convex polyhedra are described including an $O(log^2 n\ log^* n)$ parallel time algorithm for constructing the convex hull of n points in R^3 and an $O(log\ n\ log^* n)$ parallel time algorithm for detecting the separation of convex polyhedra.

† A preliminary version of some of the results in this report was presented in [DaK1].

I Introduction

The study of parallelism in computational geometry has been largely confined to individual case studies and isolated results with the exception of the recent comprehensive papers of Aggarwal *et al* [ACGOY1] [ACGOY2]. Aggarwal *et al* present a number of techniques and tools which lay the foundation for the study of parallelism in computational geometry. Among their results are parallel solutions to such familiar geometric problems as convex hull construction (in 2 and 3-d), Voronoi diagram construction (in 2-d) and closest point search, and segment intersection. More recently, Atallah and Goodrich [AG] elaborate on one technique - parallel plane sweep - which was proposed by Aggarwal *et al* .

A major - in many cases dominant - component of a number of geometric algorithms is a (possibly constrained) subdivision search problem. A *(planar) subdivision* is a partition of the plane into regions bounded by straight edges. A bounded subdivision is implicitly given by an embedded planar graph describing the face boundaries. (Even unbounded subdivisions can be described this way by including a point at infinity.) *Subdivision search* involves identifying the face of a given subdivision occupied by a given point.

In this paper, we focus on parallelism and subdivision search. We provide a direct, simple and general parallel solution to the problem of preprocessing a subdivision for fast sequential search. This is achieved by giving a parallel construction of the general hierarchical subdivision search structure presented by Kirkpatrick [K]. This approach was proposed by Aggarwal *et al* but abandoned in favour of a less general solution.

Subdivision hierarchies [K] are constructed by identifying and removing large independent sets of low-degree vertices to produce a sequence of progressively simpler subdivisions. Thus, much of our attention in this paper is devoted to parallel algorithms for the identification of large independent sets in certain restricted graphs.

We deal exclusively with graphs (including planar graphs) whose edge set is linear in the size of the vertex set. Thus we refer to the *size* of a graph by the single parameter n, the number of its vertices. Each of our algorithms assumes as input a graph G represented as an array of vertices V along with an array of edges E. Each vertex v in V has a pointer to a ring of edges incident with v. When the graph G is presented as a planar embedding, each edge ring will be ordered clockwise about its incident vertex.

For ease of exposition, we assume a Single Instruction/Multiple Data (SIMD) shared memory PRAM for our algorithms. Like Aggarwal *et al*, we assume a concurrent read/exclusive write (CREW) memory model for most of our parallel algorithms. The CREW model seems natural for our geometric applications in which many processors cooperate to construct a data structure and then access that data structure individually. In some cases, an exclusive read/exclusive write (EREW) memory model suffices to implement our algorithms.

Again like Aggarwal *et al*, we assume the availability of a number of processors linear in the input size n and make no attempt to optimize the utilization of processors. So, without loss of generality, we assume that each vertex and edge has associated with it a dedicated processor. Each processor has a distinct identifier which can be used to make local decisions. The processor identifier of a processor assigned to a vertex (respectively edge) may also be referred to as the vertex (respectively edge) number.

We will say that a problem of size n has parallel complexity $f(n)$ if it can be solved in $O(f(n))$ parallel time using $O(n)$ processors as described above. Our interest lies in understanding the asymptotic complexity of problems; we make no attempt here to optimize the constants involved.

Section II is concerned with the Subdivision Hierarchy Construction problem. Through a series of problem reductions, we examine the relative complexities of different instantiations of

the Subdivision Hierarchy Construction problem and relate it to the problem of identifying a Fractional Independent Set within a planar graph.

In Section III, we apply the concept of Subdivision Hierarchies to the construction of hierarchical representations of polyhedra. This, in turn, is applied to certain geometric intersection and separation problems. Among the results is an $O(log^2 n\ log^* n)$ parallel time† algorithm for the construction of 3-dimensional convex hulls.

II Subdivision Hierarchies and Fractional Independent Sets

Suppose S is a planar subdivision. We denote by G_S the associated embedded planar graph. A subdivision hierarchy representation of S is a sequence of increasingly coarse descriptions of S. The first element of the sequence, S_1, is a triangulation of S. Each subsequent element of the sequence, S_i, is a triangulated subdivision whose size is some fixed fraction less than the size of its predecessor S_{i-1} and each of whose regions intersects at most a constant number of regions of its predecessor S_{i-1}. The last element of the sequence, S_k, is a subdivision with at most some fixed number of vertices. Note that since the sizes of successive subdivisions form a geometrically decreasing sequence, the number of elements in the sequence is logarithmic in the size of the original graph.

More formally, the sequence of triangulated subdivisions $H(S) = S_1,...,S_k$ is said to be a *subdivision hierarchy* of S if there are positive constants c and d such that

i) $S_1 = S$ or some triangular refinement of S;

ii) $|S_k| = 3$;

iii) $|S_{i+1}| \leq (1 - 1/c)\ |S_i|$; and

†All logarithms in this paper are base 2. $log^* n$ is defined to be the number of applications of the *log* function required to reduce n to a constant value.

iv) each region R of S_{i+1} has associated with it at most d regions of S_i whose union includes R.

We first review the sequential algorithm for constructing a subdivision hierarchy from a given *n*-vertex subdivision presented by Kirkpatrick [K]. The original subdivision S is fully triangulated (in $O(n \log n)$ time) to produce the first element of the sequence, S_1. Each subsequent element S_i is derived from its predecessor S_{i-1} by identifying, and removing a set of low-degree independent vertices and retriangulating the resulting subdivision. This continues until a subdivision S_k with three vertices has been produced.

It follows from Euler's theorem that every planar subdivision has an average degree of less than 6 which implies that less than half the vertices have degree exceeding 11. From the set of vertices V of degree at most 11, an independent set of size at least $|V| / 12 \geq n / 24$ can be identified quickly. Using this, it is shown in [K] that every subdivision has an associated subdivision hierarchy with d = 11 and c = 24 which can be constructed in linear time exclusive of the initial triangulation. In another context, Lipton and Miller [LM] (and subsequently Edahiro *et al* [EKA]) showed that large independent sets of vertices of degree at most 6 can be easily identified. Accordingly, we say that a vertex has *low-degree* if it has degree less than or equal to 6.

Given a query point q and a subdivision hierarchy H(S) of a subdivision S on *n* vertices, the subdivision search algorithm is straightforward. Since S_k is of constant size, the region of S_k containing q is identified in constant time. For each face f of S_{i+1} either f is a face of S_i or f was produced when a low-degree vertex of S_i was removed. Thus given the face of S_{i+1} containing q, the face of S_i containing q can be determined in constant time. In this way the face of S_1 (hence S) containing q can be determined in $O(\log n)$ time.

For completeness, we include some implementation details relevant for both the sequential and parallel implementations. The data structure we use to represent a subdivision is the

Doubly Connected Edge List (DCEL) of Muller and Preparata [MP]. Within the DCEL, there are data elements for each vertex, edge, and face of the subdivision. Each vertex has an ordered ring of its incident edges. Each face has an ordered ring of its delimiting edges. Each edge has pointers to its two end vertices and its two incident faces.

This is augmented slightly to represent the subdivision hierarchy. Each face **f** in $\mathbf{S}_{i+1}$ has a pointer to the (low-degree) vertex **v** in $\mathbf{S}_i$ (if one exists) whose removal caused the formation of **f**. In the construction, when a vertex **v** is removed its neighbourhood is retriangulated and each of the resulting faces points to **v**. An edge (**v**, **w**) has its corresponding pointer in **w**'s edge ring removed. In the search, if **f** is the face of $\mathbf{S}_{i+1}$ containing a query point q, there are only a constant number of (triangular) faces to search in order to locate q in $\mathbf{S}_i$.

We define the *Subdivision Hierarchy Construction Problem* SHCP(n) as the problem of constructing a subdivision hierarchy for an arbitrary subdivision on n vertices. We will also use SHCP(n) to refer to the parallel complexity of solving the Subdivision Hierarchy Construction Problem. The special cases when **S** is a *Convex Subdivision* (it is bounded by a convex polygon and each of its interior faces is convex) - which we denote C-SHCP(n) - or when **S** is a *Triangular Subdivision* (it is bounded by a triangle and each of its interior faces is a triangle) - which we denote T-SHCP(n) - are of independent interest.

As we have seen, it suffices at each stage to identify and remove a set of low-degree independent vertices which constitutes a fixed fraction of the entire vertex set. Therefore, we define the *Fractional Independent Set Problem* FISP(n) as the problem of identifying for an arbitrary planar graph **G** with n vertices an independent set **I** of low-degree vertices in **G** such that $|\mathbf{I}| > n / c$ for some fixed constant c. Special cases of this problem, BD-FISP(n) and L-FISP(n), concern the restriction to bounded degree graphs, and list graphs (digraphs whose vertices have in-degree and out-degree bounded by 1) respectively.

It is possible to relate the parallel complexities of variants of the SHCP and FISP by means of some straightforward reductions:

Lemma 1:

(i) SHCP(n) ≤ O($log^2 n$) + T-SHCP(n)

(ii) C-SHCP(n) ≤ O($log\ n$) + T-SHCP(n)

(iii) T-SHCP(n) ≤ O($log\ n$) FISP(n)

(iv) FISP(n) ≤ O(1) + BD-FISP(n)

Proof:

(i) This involves embedding the given planar subdivision within a triangular subdivision. A bounding rectangle can be determined for the given subdivision in O($log\ n$) time by finding the minimum and maximum x and y values among the vertices of the subdivision. From the bounding rectangle, a bounding triangle can be determined in constant time. The techniques of [ACGOY2] or [AG] can then be used to triangulate the interior polygons in O($log^2 n$) parallel time using n processors.

(ii) This involves embedding the given convex subdivision within a triangular subdivision. First, a containing triangle is determined in O($log\ n$) time as in (i) above.

Next, the vertices on the boundary of the convex polygon containing the subdivision must be connected to the vertices of the containing triangle such that the region between the bounding polygon and the containing triangle is triangulated. In general this can be done in any of several different ways. The ring of edges defining the bounding polygon will be available in the DCEL; it is the edge ring corresponding to the external face. Using this ring, a vertex on the boundary can determine in constant time which (and how many) of the containing triangle

vertices are visible by using its incident edges. Hereafter, it is straightforward to construct the desired triangulation using only local information.

Finally, the interior convex regions must be triangulated. Within a convex polygon, each vertex is visible from every other vertex (by the definition of convexity). Thus, the only problem to be solved in triangulating a convex polygon in parallel is making sure the inserted edges don't cross. With a processor for each edge, each edge can learn its rank (relative to a lowest numbered edge) in both of its incident faces by standard list ranking techniques [W]. Since this also ranks the vertices, it is straightforward to form a triangulation by repeatedly connecting alternate vertices. Each convex region bounded by t edges can be triangulated in $O(\log t)$ time using t processors.

(iii) This can be demonstrated by simply mimicking the sequential algorithm. For each element S_i in the sequence, construct S_{i+1} by identifying a fractional independent set of S_i, removing this low degree independent set and retriangulating in constant time. The triangulation guarantees that any two consecutive edges incident on a given vertex v are two sides of a triangle, hence the three vertices involved are not independent. Therefore, although some vertex w incident to v may be high-degree, no consecutive edges in w's edge ring will be removed in any particular iteration (ie. all updates are local and can be performed in constant time). The dominant cost of each of the $O(\log n)$ iterations is the identification of the fractional independent set.

(iv) Again we mimic the sequential algorithm. Since the low degree vertices form a fixed fraction of the vertices of a planar graph, it suffices to identify a subgraph of the input graph such that all vertices have degree at most b. In the following description, the edge 'near end' and 'far end' locations are used to avoid read/write conflicts.

Each vertex processor marks its vertex high degree (as a default). It then counts its incident edges up to a maximum of b + 1. Any processor which counted up to b + 1 edges sits

out and the others mark themselves low-degree. Each low-degree vertex marks the 'far end' of its incident edges High Degree. It then marks the 'near end' of its incident edges Low Degree. If then reads the 'far end' of its incident edges and removes from its edge list the ones which are marked High Degree. In this way, all low degree vertices identify themselves, their low degree neighbours and the resulting induced graph in O(*1*) parallel time.

The following pseudo-code procedure, which is executed in parallel by each of the vertex processors, describes in more detail the procedure to identify the graph induced on the low degree vertices.

```
procedure LowDegreeSubgraph;

begin
    Mark vertex high-degree;
    degree := 0;
    test_edge := first_edge;
    counted_all := false;

    (* Count edges to identify low-degree vertices. *)

    for k := 1 to (b + 1) do
        if (not counted_all) then
            begin
                degree := degree + 1;
                test_edge := test_edge.next;
                counted_all := (test_edge = first_edge)
            end;

    (* Remove edges which are incident on high-degree vertices. *)

    if degree ≤ b then
        begin
            Mark vertex low-degree;
            for j := 1 to b do if (edge j <> null) then Mark 'far end' of edge j high-degree;
            for j := 1 to b do if (edge j <> null) then Mark 'near end' of edge j low-degree;
            for j := 1 to b do if (edge j <> null) then
                if 'far end' of edge j is marked high-degree
                    then remove edge j from edge list
        end
end.
```

Lemma 2: BD-FISP(n) $\leq$ O(1) + c L-FISP(n)

Proof: Assume we are given as input a graph G each of whose vertices has degree at most b. Each vertex is assigned a processor. We first show how the edge set E of the graph G can be decomposed, in constant time, into a constant number $t < 2b$ of sets E_i ($1 \leq i \leq t$) where each set E_i defines a list graph.

The sets E_i ($1 \leq i \leq t$) will be formed in t rounds. A set of chains is identified in parallel by allowing each vertex to determine at most one incoming edge and at most one outgoing edge from its remaining incident edges. As an edge is chosen, it is marked as belonging to chain i and is removed from consideration. Each round is divided into b subrounds. In a subround, a vertex which has not yet had a proposal accepted proposes to a new neighbour (if one is available). If a vertex receives any proposals, it will accept exactly one and ignore all others. A vertex has all of its proposals ignored in a round only if all of its neighbours have accepted other proposals in this round. Hence after 2b rounds, all of a vertex's neighbours must have accepted its proposal.

We then find a Fractional Independent set in E_1 and mark them as survivors. Among the set of survivors in E_i we find a Fractional Independent set in E_{i+1}. The set of survivors in E_t will form a fixed fraction of the vertices of G. Therefore a constant number of iterations of the Fractional Independent Set Problem for list graphs suffices to solve the Fractional Independent Set Problem for bounded degree graphs.

The following pseudo-code procedure, which is executed in parallel by each of the vertex processors, describes in more detail the procedure to decompose a degree bounded graph into a set of list graphs.

```
procedure DecomposeIntoChains;
begin

    (* Each iteration identifies 1 outgoing edge and 1 incoming edge. *)

    for i := 1 to t do
        begin
            in_mated := false;
            out_mated := false;

            for j := 1 to b do
                begin

                    (* Propose to a neighbour *)

                    if (not out_mated) and (edge j <> null) then
                        Mark 'Far End' of edge j propose;

                    (* Check proposals from neighbours *)

                    for k:= 1 to b do
                        if(not in_mated) and(edge k <> null) and
                            ('near end' of edge k is marked propose) then
                            begin
                                Mark 'near end' of edge k accept;
                                in_mated := true;
                                in_edge := k
                            end;

                    (* See if proposal was accepted *)

                    if(not out_mated) and (edge j <> null) and
                        ('far end' of edge j is marked accept) then
                        begin
                            out_mated := true;
                            out_edge := j
                        end
                end;

            (* Record incident edges for chain i *)

            if in_mated then
                begin
                    Mark 'near end' of edge in_edge in-chain (i);
                    Remove edge in_edge from current edge ring
                end;

            if out_mated then
                begin
                    Mark 'near end' of edge out_edge out-chain (i);
                    Remove edge out_edge from current edge ring
                end;
        end
end.
```

Note that the Fractional Independent Set Problem for n vertex list graphs can be solved by standard list ranking in $O(\log n)$ parallel time using $O(n)$ EREW processors [W]. However, full list ranking is unnecessary.

Lemma 3: L-FISP(n) can be solved in $O(\log^* n)$ parallel time using a deterministic algorithm.

Proof: Cole and Vishkin [CV] define an *r - ruling set* on a list graph L on n vertices to be a subset U of the vertices of L such that: i) No two vertices of U are adjacent; and ii) For each vertex v in L there is a directed path from v to some vertex in U whose edge length is at most r. Cole and Vishkin show how to find a 2-ruling set in $O(\log^* n)$ time using a technique which they call deterministic coin tossing. Note that a 2-ruling set is a fractional independent set for its list. •

Lemma 4: L-FISP(n) can be solved with probability $1 - O(c^n)$, for some $c < 1$, in $O(1)$ parallel time using a randomized algorithm.

Proof: Assign a processor to each vertex of the list. Each processor flips a 0 or a 1with equal probability. A vertex is chosen if its processor flips a 1 and either it has no successor or its successor flips a 0. With probability at least 1/4 an arbitrary vertex is chosen. However, these probabilities are not independent. Nevertheless, every second list element is chosen independently with a probability of at least 1/4. Thus applying the Chernoff bound (cf. [PB, p. 464]), we find that the probability that fewer than 1/8 of the even positioned list elements are chosen is at most c^n, where $c < 0.98$. •

Theorem 1: The Subdivision Hierarchy Construction Problem for a convex subdivision on n vertices can be solved in $O(\log n)$ expected parallel time using a randomized algorithm or $O(\log n \log^* n)$ parallel time using a deterministic algorithm.

Proof: The deterministic result is immediate from Lemmas 1, 2 and 3. The randomized algorithm exploits Lemma 4 to find (and remove) low degree independent sets for O($log\ n$) phases. At this point, the resulting subdivision has O($log\ n$) vertices, with overwhelming probability, and O($log\ n$) additional steps of the sequential algorithm suffice to complete the subdivision hierarchy. If this is not the case then the entire computation can be restarted and the expected time remains O($log\ n$). •

Corollary 1: SHCP(n) $\leq$ O($log^2 n$)

Once the subdivision hierarchy structure is constructed, the algorithm for subdivision search is identical to that presented in [K]. It is worth noting that Atallah and Goodrich [AG] use the parallel plane sweep technique to perform planar point location with O($log\ n\ log\ log\ n$) parallel preprocessing, O($n\ log\ n$) space and O($log\ n$) sequential query time. The subdivision hierarchy technique uses O($log^2 n$) parallel preprocessing (O($log\ n\ log^* n$) for convex subdivisions), O(n) space and O($log\ n$) sequential query time. Furthermore, as originally presented in [DK1], the subdivision hierarchy can be used for 3-dimensional applications which seem to be beyond the scope of the parallel plane sweep technique. We expand on this in the next section.

III Applications

As in [DK1] and [DK3], we exploit the fact that the surface of a convex polyhedron is topologically equivalent to a bounded planar subdivision and define an hierarchical representation for convex polytopes similar to the hierarchical representation for planar subdivisions:

Let **P** be a convex polytope on n vertices with vertex set V(**P**). A sequence of polytopes H(**P**) = $\mathbf{P}_1,...,\mathbf{P}_k$ is said to be a hierarchical representation of **P** if

i) $\mathbf{P}_1 = \mathbf{P}$;

ii) $|V(P_k)|$ is bounded by a constant;

iii) $V(P_{i+1}) \subset V(P_i)$; and

iv) the vertices of $V(P_{i+1}) - V(P_i)$ form an independent set (i.e., are non-adjacent) in P_i.

The Doubly Connected Edge List can be used to implement the hierarchical representation of convex polyhedra in the same way as for Subdivision Hierarchies.

Corollary 2: Given a convex polyhedron on n vertices, a hierarchical representation with $O(\log n)$ elements can be constructed in $O(\log n)$ expected parallel time using a randomized algorithm or $O(\log n \log^* n)$ parallel time using a deterministic algorithm.

Proof: This follows from Theorem 1 and the fact that the faces of a convex polyhedron can be triangulated in $O(\log n)$ time.

This hierarchical representation can be used to answer many different intersection and separation queries involving polyhedra. Convex polyhedron intersection and separation queries with points, lines and planes can be answered in $O(\log n)$ sequential time [DK1, DK2, DK3, DK4].

A line query, as defined by Aggarwal *et al* [ACGOY1], poses the following problem: Given a convex polyhedron P and a line L in 3-space, determine whether or not L intersects P and, if not, give the two planes through L tangent to P.

Lemma 5: Given the hierarchical representation of a convex polyhedron H(P) and a line L in 3-space, a line query can be answered in $O(\log n)$ sequential time.

Proof: In [DK1, DK4] an $O(\log n)$ sequential algorithm is described for detecting the intersection of a line L and an hierarchically described polyhedron P and for constructing the intersection when it is non-empty. By a straightforward modification of the same techniques, it is possible to construct the tangent planes through L when the intersection is empty. •

Corollary 3: Given the hierarchical representations of two separated convex polyhedra H(**P**) and H(**Q**), their convex union can be constructed in O(*log n*) parallel time using O(n) CREW processors.

Proof: By Lemma 5, given the subdivision hierarchy of a convex object with n vertices O and a line L, the hierarchy can be used to answer a line query in O(*log n*) time with a single processor. Thus, to construct the convex union in logarithmic time a processor is assigned to each edge in both **P** and **Q**. The subdivision hierarchy is used to determine the two supporting planes of the opposite convex polyhedron through each edge. As a result each edge (and incident faces) can be classified as being in or out of the convex union. It remains to update the edge rings of all vertices to reflect adjacencies on the convex union. The details, which involve list ranking and merging, are described by [ACGOY2]. •

Corollary 4: The 3-d Convex Hull of n vertices can be constructed in O($log^2 n$) expected parallel time using a randomized algorithm or O($log^2 n\ log^* n$) parallel time using a deterministic algorithm.

Proof: The algorithm proceeds in a manner similar to the divide and conquer algorithm of Preparata and Hong [PH] and Aggarwal *et al* [ACGOY2]. The vertex set is lexicographically sorted and is recursively divided into separable sets. The hull of each of the sets is found recursively and the Polyhedral Hierarchy is constructed for each. The separable Convex Union algorithm of Corollary 3 is used as the merge step. Constructing the subdivision hierarchy is the dominant cost of the recursive step. •

The 3-d hull construction algorithm described in Aggarwal *et al* [ACGOY1] runs in O($log^4 n$) parallel time. This was improved to O($log^3 n$) parallel time in [ACGOY2]. The use of the hierarchical representation makes our approach significantly simpler as well as more efficient.

Corollary 5: The 3-d Convex Polyhedron Separation problem (determining the separation of two convex objects in R^3) can be solved in O(*log n*) expected parallel time using a randomized algorithm or O(*log n log* n*) parallel time using a deterministic algorithm.

Proof: The separation of two convex polyhedra will be realized by a vertex-vertex pair, an edge-vertex pair or a edge-edge pair. Once the Polyhedral Hierarchy for both convex objects is constructed, a processor is assigned to each vertex and edge. Each will determine its separation from the opposite convex polyhedron in O(*log n*) time [DK3]. Then a minimum operation can be performed to determine the polyhedral separation in O(*log n*) time. Constructing the polyhedral hierarchy is the dominant cost. •

As another application of the fast parallel independent set technique, we note the following:

Corollary 6: A planar graph can be 7-coloured in O(*log n*) expected parallel time using a randomized algorithm or O(*log n log* n*) parallel time using a deterministic algorithm.

Proof: This follows by a straightforward parallel implementation of the sequential 7-colouring algorithm of Lipton and Miller [LM]. •

This result has been substantially improved by the recent work of Chrobak *et al* [CDH] who demonstrate that the same complexity bounds hold for the problem of 5-colouring a planar graph.

As a final observation, we note that the solution to the Fractional Independent Set Problem can be extended to find a Maximal Independent Set in an *n*-vertex planar graph in O(*log n log* n*) (respectively O(*log n*) expected) parallel time using a deterministic (respectively randomized) algorithm on O(*n*) EREW processors as described in [DaK1] and [DaK2].

IV Discussion

We have drawn together some of the fundamental techniques of parallel computation in linear data structures (notably [CV]) with the hierarchical approach to the representation and manipulation of planar subdivisions and polyhedra ([K], [DK1], [DK2], [DK3], and [DK4]) to produce simple and efficient parallel algorithms for a variety of problems in computational geometry.

Many of our results were inspired by the original work of Aggarwal *et al* ([ACGOY1], which was recently updated in [ACGOY2]).Very recently we have become aware of the related work of Chrobak, Diks, and Hagerup [CDH]. Though their focus is on parallel algorithms for graph colouring, their techniques are very similar to those presented here and we expect that they could be applied equally successfully in our setting. In fact, Chrobak *et al* pay more attention than do we to issues of processor utilization which is a natural direction to seek improvements of our results.

Acknowledgement

This work was supported in part by the National Sciences and Engineering Research Council of Canada, grant A3583.

References

[ACGOY1] Aggarwal, A., Chazelle, B., Guibas, L., O'Dunlaing, C., and Yap, C., "Parallel Computational Geometry (extended abstract)", *Proc. of the 26th IEEE Symposium on Foundations of Computer Science* (1985), pp. 468-477.

[ACGOY2] Aggarwal, A., Chazelle, B., Guibas, L., O'Dunlaing, C., and Yap, C., "Parallel Computational Geometry ", To Appear in *Algorithmica* (1987).

[AG] Atallah, M.J. and Goodrich, M.T., "Efficient Plane Sweeping in Parallel (Preliminary Version)", *Proc. of the 2nd ACM Symposium on Computational Geometry* (1986), pp. 216-225.

[CDH] Chrobak, M., Diks, K., and Hagerup, T., "Parallel 5-Colouring of Planar Graphs", Preprint (1987).

[CV] Cole, R. and Vishkin, U., "Deterministic Coin Tossing and Accelerating Cascades: Micro and Macro Techniques for Designing Parallel Algorithms", *Proc. of the 18th ACM Symposium on Theory of Computing* (1986), pp. 206-219.

[DaK1] Dadoun, N. and Kirkpatrick, D.G., "Parallel Processing for Efficient Subdivision Search", *Proc. of the 3rd ACM Symposium on Computational Geometry* (1987).

[DaK2] Dadoun, N. and Kirkpatrick, D.G., "A Parallel Algorithm for Finding Maximal Independent Sets in Planar Graphs", In Preparation.

[DK1] Dobkin, D.P. and Kirkpatrick, D.G., "Fast Detection of Polyhedral Intersections", *Proc. International Colloquium on Automata, Languages and Programming* (1982), pp. 154-165.

[DK2] Dobkin, D.P. and Kirkpatrick, D.G., "Fast Detection of Polyhedral Intersection", *Theoretical Computer Science* 27 (1983), pp. 241-253.

[DK3] Dobkin, D.P. and Kirkpatrick, D.G., "A Linear Time Algorithm for Determining the Separation of Convex Polyhedra", *Journal of Algorithms* 6, 3 (1985), pp. 381-392.

[DK4] Dobkin, D.P. and Kirkpatrick, D.G., "Fast Algorithms for Preprocessed Polyhedral Intersection Detection", In Preparation.

[EKA] Edahiro, M., Kokubo, I. and Asano, T., "A New Point-Location Algorithm and Its Practical Efficiency - Comparison with Existing Algorithms", *ACM Transactions on Graphics* 3, 2 (1984), pp. 86-109.

[K] Kirkpatrick, D.G., "Optimal Search In Planar Subdivisions" *SIAM Journal of Computing* 12,1 (1983), pp. 28-35.

[LM] Lipton, R.J., and Miller, R.E., "A Batching Method for Coloring Planar Graphs", *Information Processing Letters* 7,4 (1978), pp. 185-188.

[PB] Purdom, P.W., and Brown, C.A., *The Analysis of Algorithms*, Holt, Rinehart and Winston, New York (1985).

[PH] Preparata, F., and Hong, S. J., "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", *Communications of the ACM* 20 (1978), pp. 87-93.

[W] Wylie, J.C., "The Complexity of Parallel Computation", Technical Report TR 79-387, Dept. of Computer Science, Cornell University, Ithaca, New York, 1979.