Scalable Algorithm Design Techniques for Discrete Problems that Lack Obvious Structure

Andrew Rau-Chaplin *

Abstract

To be relevant in practice parallel algorithms must be developed for realistic models of parallel computing, such as the BSP, LogP, and Coarse Grained Multicomputer (CGM) models. This paper surveys three algorithm design techniques - Spatial Partitioning, Sampling, and Data Structure Partitioning - that have led to efficient and practical algorithms for a variety of problems on such realistic models. One particularly noteworthy feature of these techniques is that they often led to algorithms involving only $O(1)$ global communications steps, even for problems that may appear highly unstructured.

1 Introduction

Parallel algorithms for problems involving discrete objects, such as those found in geometric, graph and string problems have been studied extensively [1, 16]. These studies have been motivated by important application areas including computational biology, computational geometry, geographic information systems, image processing, finite element mesh generation and AI/knowledge representation. Until recently, these studies focused almost exclusively on parallel algorithms for highly abstract PRAM and distributed memory models. Typically, a given problem of size n has been solved on a parallel computer with p processors (e.g., a PRAM, mesh, or hypercube multiprocessor) in time $T_{parallel}$. The goal has been to develop *optimal* solutions where $T_{parallel} = O(\frac{T_{sequential}}{p})$, $T_{sequential}$ being the sequential time complexity of the problem, and the focus has been on the case $\frac{n}{p} = O(1)$, also referred to as the *fine grained* case.

However, to be relevant in practice, parallel algorithms must be developed for more realistic models that better reflect existing parallel machines, such as the BSP, LogP, C^3, and Coarse Grained Multicomputer (CGM) models and

*School of Computer Science, Technical University of Nova Scotia, P.O. Box 1000, Halifax, Nova Scotia, Canada B3J 2X4. E-mail: arc@tuns.ca. Research partially supported by Natural Sciences and Engineering Research Council of Canada.

these algorithms must be *scalable*, that is, they must be applicable and efficient for a wide range of ratios $\frac{n}{p}$ [12].

Recently there has been a growing interest in coarse grained computational models [20, 4, 13] and the design of coarse gained geometric algorithms [7, 11, 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, described by Valiant [20], uses 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. Culler et. al. introduced the LogP model which, using Valiant's BSP model as a starting point, focuses on the technological trend from fine grained parallel machines towards coarse grained systems and advocates portable parallel algorithm design [4]. Other coarse grained models focus more on utilizing local computation and minimizing global operations. These include the C^3 model [13], and the Coarse Grained Multicomputer (CGM) model used in this paper [7]. All of these models differ somewhat in their focus, operations and accounting models, but share as a principle tenant that parallel algorithms must minimize the number of global communication steps (H-relations, Supersteps) to obtain efficiency on "real" machines.

The ultimate realization of the desire to minimize communications is the development of deterministic algorithms which involve only $O(1)$ global communications steps. Perhaps surprisingly such algorithms can be developed, even for problems that may appear highly unstructured. In this paper we survey three algorithmic design techniques - Spatial Partitioning, Sampling, and Data Structure Partitioning - that have led to deterministic scalable algorithms involving only $O(1)$ global communications steps.

Throughout this paper we will use the coarse grained multicomputer model, CGM(n, p), although any of the other realistic models mentioned above would suffice. The advantage of this model for our purposes is its simplicity and natural mapping onto existing parallel machines.

Most existing multicomputers (e.g. the Intel Paragon, Intel iPSC/860, and Thinking Machines Corp. CM-5) consist of a set of p *state-of-the-art* processors (e.g. SPARC proc.), each with considerable local memory, connected to some interconnection network (e.g. mesh, hypercube, fat tree). These machines are usually *coarse grained*, i.e. the size of each local memory is "considerably larger" than $O(1)$. The coarse grained multicomputer, CGM(n, p), considered in this paper is a set of p processors numbered from 1 to p with $O(\frac{n}{p})$ local memory each, connected via some arbitrary interconnection network or a shared memory. Commonly used interconnection networks for a CGM include the 2D-mesh (e.g. Intel Paragon), hypercube (e.g. Intel iPSC/860) and the fat tree (e.g. Thinking Machines CM-5). Each processor may exchange messages of $O(\log n)$ bits with any one of its immediate neighbors in constant time. For determining time complexities both, local computation time and interprocessor communication time are considered, in the standard way. The term "coarse grained" refers to the fact that the size $O(\frac{n}{p})$ of each local memory is assumed to be "considerably larger" than $O(1)$. In reporting results in this model, we are interested in three important resource measures:

1. The amount of local computation required;

2. The number and type of global communication phases required;

3. The scalability assumption of the algorithm, that is the minimum value for the ratio $\frac{n}{p}$ for which the algorithm is applicable.

Ideally, our coarse grained algorithms would be "completely scalable", i.e. have scalability assumptions of $\frac{n}{p} \geq 1$, however in practice $\frac{n}{p} \geq p^{\epsilon}$ ($\epsilon > 0$) or $\frac{n}{p} \geq p$ will suffice.

All of the CGM algorithms referred to in this paper consist of local computation plus calls to a small number of standard communication operations including parallel prefix, segmented broadcast, multinode broadcast, total exchange, circular rotation, and sorting. Since all of these communication operations can be implemented in terms of sorting we will often summarize the complexity of a constant number of them as $O(T_s(n,p))$, which represents the time to sort n data on a p-processor CGM. For a more detailed description of the model and its associated operations, see [7].

2 Technique 1: Spatial Partitioning

Spatial partitioning is perhaps the simplest technique for developing efficient CGM algorithms. In a nutshell, the basic idea is as follows: Try to combine optimal sequential algorithms for a given problem with an efficient global routing and partitioning mechanism. Devise a constant number of partitioning schemes of the global problem (on the entire data set of n data items) into p subproblems of size $O(\frac{n}{p})$. Have each processor solve sequentially a constant number of such $O(\frac{n}{p})$ size subproblems, and then use a constant number of global routing operations to permute the subproblems between the processors. Eventually, by combining the $O(1)$ solutions of its $O(\frac{n}{p})$ size subproblems, each processor determines its $O(\frac{n}{p})$ size portion of the *global* solution.

The above is necessarily an oversimplification. Most actual algorithms will do more than just those permutations. The main challenge lies in devising the above mentioned partitioning schemes. Note that, each processor will solve only a constant number of $O(\frac{n}{p})$ size subproblems, but eventually will have to determine its part of the entire $O(n)$ size problem (without having seen all of the n data items). The most complicated part of the algorithm is to ensure that at most $O(1)$ global communication rounds are required.

Not surprisingly this technique has been most effective in geometric problems where the data items (typically, points, line or polygons) tend to interact more with other spatially close data items than with distant ones.

As an example of the spatial partitioning technique, consider the following algorithm from [7] for computing the lower envelope of non-intersecting line segments in the plane. The problem is defined as follows: Given a set S of n non-intersecting line segments in the plane, the *lower envelope* problem consists of computing the set $LE(S)$ of segment portions visible from the point $(0, -\infty)$.

Observation 1 *The lower envelope of n non-intersecting line segments is x-monotone and has size $O(n)$.*

Algorithm 1
Architecture: A p-processor coarse grained multicomputer, CGM(n, p), with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$, $\frac{n}{p} \geq p$.
Input: Each processor p_i stores a set S_i of $\frac{n}{p}$ line segments of S.
Output: Each processor stores $O(\frac{n}{p})$ segment portions of $LE(S)$.

(1) Each processor p_i computes sequentially $LE(S_i)$ for its subset S_i of line segments (ignoring all other segments).[15]

(2) Globally sort the segments in $\bigcup_{i=1}^{p} LE(S_i)$ by the x-coordinate of their right endpoints, which moves to each processor p_i a new set V_i of $O(\frac{n}{p})$ segments. Note that, each processor p_i also keeps the set $LE(S_i)$.

(3) Each processor p_i determines the vertical line l_i through the rightmost vertex of a segment of V_i. Perform a multinode broadcast where processor p_i sends l_i to all other processors. Hence, each processor stores all p vertical lines $l_1, \ldots, l_p$.

(4) Perform a total exchange, with processor p_i sending segment $s \in LE(S_i)$ to processor p_j iff s intersects the vertical line l_j. Let R_j be the set of segments received by processor p_j.

(5) Each processor p_i computes sequentially $LE(V_i \cup R_i)$.

— End of Algorithm —

This algorithm works by first reducing the amount of data pertinent to the global solution (Step 1) and then spatially partitioning the plane into p vertical slabs (Step 2 and 3). Each slab consisting of $\frac{n}{p}$ line segments is stored on a single processor which needs only p pieces of "global information" (Step 4) to complete the computation of the lower envelope for its slab (Step 5). The algorithm solves the lower envelope problem for a set of n non-intersecting line segments in the plane on a p-processor coarse grained multicomputer with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$, $\frac{n}{p} \geq p$, in time $O(\frac{n \log n}{p} + T_s(n,p))$.

Scalable CGM algorithms based on spatial partitioning have been developed for the following problems:

(1) All $2D$-nearest neighbors in a point set,

(2) $2D$-weighted dominance counting in point sets,

(3) $3D$-maxima in point sets,

(4) Area/Intersection of the union of rectangles,

(5) Lower envelope of non-intersecting line segments in the plane (and, with slightly more memory, for possibly intersecting line segments),

(6) Lower envelope of fixed degree polynomial functions,

(7) Rectangle finding problems: all isonormal rectangles, all rectangles, all isonormal squares, all squares

(8) Minimization of Hausdorff distances between point sets

(9) and a variety of dynamic computational geometry problems concerning geometric properties of moving point-objects.

The algorithms for Problems 1-5 appeared in [7] and have a running time of $O(\frac{T_{sequential}}{p} + T_s(n,p))$ on a p-processor coarse grained multicomputer, $\text{CGM}(n,p)$, with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$ where $\frac{n}{p} \geq p$. $T_s(n,p)$ refers to the time to sort globally n data items stored on a $\text{CGM}(n,p)$, $\frac{n}{p}$ data items on each processor. Since $T_{sequential} = \Theta(n \log n)$ for Problems 1-5, the algorithms either run in optimal time $\Theta(\frac{n \log n}{p})$ or in sort time $T_s(n,p)$ for the respective architecture.

The algorithms for Problems 6-9 are described in [3] and also require $\frac{n}{p} \geq p$. Problem 6 is to find the lower envelope or minimum of S, an n element set of polynomial functions of degree at most k, and is fundamental to the solution of a variety of interesting problems. The lower envelope of S can be computed on a $CGM(p\ \lambda(\lambda(\frac{n}{p}, k), k),\ p)$ in $T_s(p\ \lambda(\lambda(\frac{n}{p}, k), k),\ p)$ time, where $\lambda(n,s)$ is the maximal length of a *Davenport-Schinzel sequence* [5] defined by parameters (n,s) and is, at worst, slightly more than linear in n.

The Hausdorff distance [18] is a measure of how well two sets A and B resemble each other with respect to their positions; if A and B are finite sets regarded as statistical populations, this measure is an alternative to more common statistical measures of population similarity. When A is subjected to a translation T so that $h = H(T(A), B)$ is minimized, h may be regarded as a measure of how well an image A matches a template B. Problem 7 is the following: compute a translation T of A that minimizes the Hausdorff distance $H(T(A), B)$, where $A \cup B \subset R^1$, $|A| = m$, $|B| = n$. It can be solved using spatial partitioning on a $CGM(m+n,p)$ in $O(\frac{T_{sequential}}{p} + T_s(m+n,p))$ time.

Scalable CGM algorithms for a variety of problems concerned with geometric properties of moving objects have also been developed based on spatial partitioning. These problems all assume that k is a fixed positive integer, and that $S = \{s_0, s_1, \ldots, s_{n-1}\}$ is a set of point-objects moving in the Euclidean space R^d so that for each $s \in S$, the location of s at time t is described by a vector-valued function, each of whose Cartesian coordinates is a polynomial function of t of degree at most k. Scalable algorithms using spatial partitioning are given in [3] to solve the following problems.

- What is the nearest $s \in S \setminus \{s_0\}$ to s_0?
- When is S contained in a given fixed rectilinear, iso-oriented hyperrectangle?
- What is the edge-length of the smallest rectilinear, iso-oriented hypercube that contains S at time t?
- Assume $d = 2$. When is s_0 a vertex of the convex hull of S?

3 Technique 2: Sampling

Consider the problem of constructing the Convex Hull of a set S of n points in the plane. The obvious approach in the CGM setting is as follows: 1) Sort the points in S by x-coordinate and let S_i denote the set of $\frac{n}{p}$ sorted points now stored on processor i. 2) Independently and in parallel, have each processor i compute the convex hull of the set S_i and let X_i denote the result on processor i. 3) Merge the p convex hulls, X_i, into a convex hull using $O(1)$ global communication rounds.

Step 1 of the algorithm above can be completed by using a global sort operation and Step 2 is a totally sequential step and can be completed in time $O(\frac{n \log n}{p})$ using well known sequential methods [19]. The problem that now remains is how to merge p convex hulls, stored one per processor on a p processor CGM, into a single convex hull using a constant number of global communication rounds. We will focus on how to merge upper hulls, lower hulls and therefore the complete hull, can be computed analogously. Sequentially, two upper hulls of size $O(n)$ can be merged by finding their upper common tangent using a $\log n$ time binary search algorithm [19], but straight-forward application of this algorithms in the CGM setting yields $O(\log p \log n)$ communications steps!

The first trick to performing this merge in fewer communication steps is not to do pair-wise merging of upper hulls, but rather to find *all* p^2 upper tangent lines between the upper hulls. Clearly this is more tangent lines than we strictly need but not so many that we can't store them and computing them this way avoids the $\log p$ communications rounds that come from pairwise merging.

So how can we compute the upper common tangent between an upper hull X_i and an upper hull X_j (to its right) in only a constant number of communications rounds? The answer is to work with a *sample* of many points from X_i, rather than just the one point that is used in each step of the $\log n$ step sequential binary search. The following simple algorithm assumes $\frac{n}{p} \geq p^2$, but can be extended to scale over a larger range of values of n and p, assuming only that $\frac{n}{p} \geq p^{\epsilon}$ ($\epsilon > 0$) (See [11]).

Algorithm 2
Architecture: A p-processor coarse grained multicomputer, CGM(n, p), with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$, $\frac{n}{p} \geq p^2$.
Input: The set of p upper hulls X_i consisting of a total of at most n points from S, where X_i is stored on processor q_i, $1 \leq i \leq p$.
Output: A distributed representation of the upper hull of S.

(1) Each processor q_i sequentially identifies a sample set G_i composed of every p^{th} point from X_i. Note $|G_i| = \frac{n}{p^2}$.

(2) Perform an all-to-all broadcast of G_i and associate with each point $a \in G_i$ its two neighbours in X_i. Each processor q_j receives $O(\frac{n}{p})$ points and can compute for each received point a if the upper tangent line between X_i and X_j is rooted in X_i before, at or after a in $O(\frac{n}{p} \log \frac{n}{p})$ [17]. Perform an all-to-all broadcast to return these results.

(3) Each processor q_i can now identify with respect to each X_j a region $R_{i,j}$ of p points from X_j which is guaranteed to contain the point that roots the upper tangent line between X_i and X_j.

(4) Repeats Steps 2 and 3 using a personalized all-to-all broadcast with the point sets $R_{i,j}$ being sent to q_i, rather than G_i. Note that every processor receives p^2 points and therefore $\frac{n}{p} \geq p^2$ must hold.

(5) Each processor has now identified the upper common tangent between the upper hull X_i and all upper hulls X_j, $j > i$, and can perform an all-to-all broadcast to distribute this information globally. Using this infomation the the part of the upper hull of S that resides on earch processor can be locally computed.

— End of Algorithm —

The algorithm given above is based on ideas developed in [17] and appeared in [11]. It requires time $O(\frac{n \log n}{p} + T_s(n,p))$ on a p-processor coarse grained multicomputer, CGM(n,p), with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$ where $\frac{n}{p} \geq p^2$. Since computing 2d Convex Hull requires time $T_{sequential} = \Theta(n \log n)$ this algorithm either run in optimal time, $\Theta(\frac{n \log n}{p})$, or in sort time, $T_s(n,p)$, for the interconnection network in question. These results become optimal when $\frac{T_{sequential}}{p}$ dominates $T_s(n,p)$ or for interconnection networks like the mesh for which optimal sorting algorithms exist. Furthermore, this convex hull algorithm can be extended to scale over a larger range of values of n and p, assuming only that $\frac{n}{p} \geq p^{\epsilon}$ $(\epsilon > 0)$ (See [11]). The same technique can also be used to compute the triangulation of n points in the plane for the same model and with the same space and time complexities [11].

The sampling technique demonstrated in the CGM convex hull algorithm appears to be a powerful technique for designing scalable algorithms requiring only $O(1)$ communication rounds. Attempting to applying it to other problems involving discrete data sets is an interesting avenue for further research.

4 Technique 3: Data Structure Partitioning

The idea behind data structure partitioning is the following: If you can't find a way to spatially partition or sample your data, find a data structure that represents the data and partition it instead. Often data structures have a higher degree of regularity than the data they represent.

Consider for example, the problem of determining for a given set S of r pairwise disjoint m-vertex polygons of simple polygons all directions d such that S is separable by a sequence of r translations in direction d (one for each polygon). This is called the uni-directional translation ordering problem.

Most sets S of simple polygons defy useful spatial partitioning. Any straightforward spatial partition of S into p rectangularly bounded regions leaves the m polygons cut into basically unrelated fragments. However, there is a data structure, the segment tree, which can ably represents the problem and can be partitioned and searched efficiently (i.e. in $O(1)$ communication phases) on a CGM using a distribute data structuring technique called Multisearch. The Multisearch paradigm was first described in [8] for hypercubes and has since been extended in other parallel models [2, 7]. To illustrate this approach we will describe a simple version of CGM multisearch for balanced k-ary trees, that first appreared in [7].

Let $T = (V,E)$ be a balanced k-ary tree of size n and height $h = O(\log_k n)$, where k is a fixed constant. The definition of the multisearch problem for T and a set $Q = \{q_1, \ldots, q_m\}$ of $m = O(n)$ search queries on T is as follows:

Each query $q \in Q$ has a *search path*, $path(q) = (v_1(q), \ldots, v_h(q))$, of h vertices of T (from the root to a leaf of T) which is a sequence defined by a successor function $f : (V \cup start) \times Q \rightarrow V$ with the following properties: $f(start, q) = v_1$, $f(v_i, q) = v_{i+1}$ where $(v_i, v_{i+1}) \in E$ and $f(v_i, q)$ can be computed by a single processor in time $O(1)$. We say that query q *visits* node $v_t(q)$ at time t. The *multisearch problem* for Q on T consists of executing (in parallel) all m search processes induced by the m search queries. It is important to note that the m search processes may overlap arbitrarily. That is, at any time t, any node of T may be visited by an arbitrary number of queries. See [2] and [8] for more details.

Define as T_0 the subtree of T induced by the root and all nodes of T which have a distance from the root of at most $\log_k p$. Subtree T_0 has $p' \leq p$ leaves. To simplify exposition, assume w.l.o.g. that $p' = p$. Let T_i be the subtree of T rooted at the i-th leaf of T_0, $1 \leq i \leq p$.

Algorithm 3
Architecture: A p-processor coarse grained multicomputer, $\text{CGM}(n,p)$, with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$, $\frac{n}{p} \geq p$.
Input: Each processor stores $\frac{n}{p}$ nodes of T and $\frac{m}{p} = O(\frac{n}{p})$ queries $q \in Q$.
Result: Each $q \in Q$ visits its entire search path $path(q)$.

(1) Using a total exchange operation, create p copies of T_0 and distribute them such that each processor has one copy of T_0.

(2) Using its copy of T_0, each processor performs the first $\log_k p$ multisearch steps for its $O(\frac{n}{p})$ search queries.

(3) For each tree T_i compute $c(T_i) = \left\lceil \frac{|\{q \in Q : v_{\log_k p}(q) \in T_i\}|}{\frac{m}{p}} \right\rceil$, $1 \leq i \leq p$.

(4) Create $c(T_i)$ copies of each subtree T_i and and distribute them such that each processor stores at most two subtrees.

(5) Redistribute Q such that every query $q \in Q$ is stored at a processor that also stores a copy of the subtree T_i $(1 \leq i \leq p)$ containing $v_{\log_k p}(q)$.

(6) Each processor performs the remaining $h - \log_k p$ multisearch steps for its $O(\frac{n}{p})$ search queries.

— End of Algorithm —

Using the algorithm given above, the multisearch problem for a balanced search tree of size $O(n)$ and fixed degree k, and a set of $m = O(n)$ search queries, can be solved on a p-processor coarse grained multicomputer with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$, $\frac{n}{p} \geq p$, in time $O(\frac{n \log n}{p} + T_s(n,p))$ [7].

Extensions to this basic technique allow queries to move both up and down the data structure and to both read and write values to the nodes. To use this technique one needs only to describe how to construct in parallel the tree T to be used in a particular application, and give the application specific function $f : (V \cup start) \times Q \to V$ with the appropriate properties, before calling Multisearch to advance all of the queries down their search paths. Scalable algorithms have been developed, based on this CGM Multisearch algorithm, for the following problems:

(1) Uni-directional separability problem for simple polygons

(2) Multi-directional separability problem for simple polygons

(3) Trapezoidal decomposition of a set of line segments

(4) One-Dimensional Range Search Query Reporting

(5) Bichromatic Segment Intersection reporting problem

(6) d-Dimensional Range Search Query Problems

Let S be a set of r pairwise disjoint m-vertex polygons. The *uni-directional separability* problem consists of determining all directions d such that S is separable by a sequence of r translations in direction d (one for each polygon). The *multi-directional separability* problem asks if S is separable by a sequence of r translations in different directions. These problems (Problems 1-2) were solved in [7] on a p-processor coarse grained multicomputer with arbitrary interconnection network and local memories of size $O(\frac{n}{p})$, $n = O(r^2 + rm)$ and $\frac{n}{p} \geq p$, in time $O(\frac{r^2(m+\log r)}{p} + T_s(r^2, p))$.

Problems 3-5 were solved in [9] using data structure partitioning via an extended version of Multisearch. Let k denote the size of the output. Problems 3 was solved on a $CGM(n \log p, p)$ with local memories of size $O(\frac{n \log p}{p})$, $\frac{n}{p} \geq p$, in time $O(\frac{n \log n \log p}{p} + T_s(n \log p, p))$. Problems 4 was solved on a $CGM(\max(n,k), p)$ with local memories of size $O(\frac{\max(n,k)}{p})$, $\frac{n}{p} \geq p$, in time $O(\frac{n \log n}{p} + \frac{k}{p} + T_s(n,p))$. Problems 5 was solved on a $CGM(\max(k, n \log p), p)$ with local memories of size $O(\frac{\max(k, n \log p)}{p})$, $\frac{n}{p} \geq p$, in time $O(\frac{n \log n \log p}{p} + \frac{k}{p} + T_s(n \log p, p))$.

Problem 6 is a basic geometric and database problem. Consider a collection L of n records, where each record l has a value $key(l)$ and is identified by an ordered d-tuple $(x_1(l), ..., x_d(l)) \in E^d$, the d-dimensional Cartesian space. In the range search problem, the query specifies a domain q in E^d, and the outcome of the search, depending on the application, may be either the subset L_q of the points of L contained in q, or the number of such points, or more generally a function $\bigotimes_{l \in L_q} f(l)$, where $f(l)$ is an element of a commutative semigroup with operation $\otimes$. Problems 6 can be solved using data structure partitioning [10] for a range tree T of size $s = O(n \log^{d-1} n)$ on a $CGM(\max(k,s), p)$ with local memories of size $O(\frac{s}{p})$, $\frac{s}{p} \geq p$, in time O$(\frac{s}{p} + T_s(s,p) + \frac{k}{p})$, where k is the size of the output.

5 Summary and Conclusions

In this paper we have surveyed three algorithmic design techniques - Spatial Partitioning, Sampling, and Data Structure Partitioning - that have led to scalable CGM algorithms involving only $O(1)$ global communications steps for problems that appeared to be highly unstructured. Implementations of algorithms using these techniques [7, 11] have indeed verified that they do result in fast practical codes on real parallel machines. For the most part, we have described geometric applications, but many interesting open discrete problems remain in this setting. In particular those problems dealing with graphs seem important and very challenging.

References

[1] S.G. Akl and K.A. Lyons, *Parallel Computational Geometry*, Prentice-Hall, New York, 1993.

[2] M.J. Atallah, F. Dehne, R. Miller, A. Rau-Chaplin, and J.-J. Tsay Multisearch techniques for implementing data structures on a mesh-connected computer. *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pp. 204–214, 1991.

[3] L. Boxer, R. Miller and A. Rau-Chaplin, Some Scalable Parallel Geometric Algorithms, In Preparation.

[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos R. Subramonian, and T. von Eicken, LogP: Towards a Realistic Model of Parallel Computation. *Proc. 4th ACM SIGPLAN Sym. on Principles of Parallel Programming*, 1993.

[5] H. Davenport and A. Schinzel, A combinatorial problem connected with differential equations. *Amer. J. Math.* 87 (1965), 684-694.

[6] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. Khokhar, A randomized parallel 3D convex hull algorithm for coarse grained multicomputers, *Proc. 7th IEEE Symp. on Parallel and Distributed Processing*, 1995.

[7] F. Dehne, A. Fabri, and A. Rau-Chaplin, Scalable parallel geometric algorithms for multicomputers, *Proc. 7th ACM Symp. on Computational Geometry*, 1993.

[8] F. Dehne and A. Rau-Chaplin. Implementing data structures on a hypercube multiprocessor and applications in parallel computational geometry. *Journal of Parallel and Distributed Computing*, Vol. 8, No. 4, pp. 367–375, 1990.

[9] A. Fabri, and O. Devillers, Scalable Algorithms for Bichromatic Line Segment Intersection Problems on Coarse Grained Multicomputers, *Proc. 3rd Workshop on Algorithms and Data Structures*, 1993.

[10] A. Ferreira, C. Kenyon, A. Rau-Chaplin, and S. Ubeda, Scalable Algorithms for the d-Dimensional Range Search on Coarse Grained Multicomputers,In Preparation.

[11] A. Ferreira, A. Rau-Chaplin, and S. Ubeda, Scalable 2d convex hull and triangulation algorithms for coarse grained multicomputers, *Proc. 7th IEEE Symp. on Parallel and Distributed Processing*, 1995.

[12] *Grand Challenges: High Performance Computing and Communications*. The FY 1992 U.S. Research and Development Program. A Report by the Committee on Physical, Mathematical, and Engineering Sciences. Federal Council for Science, Engineering, and Technology. To Supplement the U.S. President's Fiscal Year 1992 Budget.

[13] S. Hambrusch, and A. Khokhar, C3: An Architecture-Independent Model For Coarse-Grained Parallel Machines, Purdue University Computer Sciences Technical Report CSD-TR-93-080 (1993).

[14] S. Hart and M. Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes, *Combinatorica* 6 (1986), 151-177.

[15] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters* 33, pp. 169–174, 1989.

[16] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[17] R. Miller and Q. Stout. Efficient Convex Hull Algorithms *IEEE Trans. on Computers*, 37, pp. 1605–1618, 1988.

[18] S.B. Nadler, Jr., *Hyperspaces of Sets*, Marcel Dekker, Inc., New York, 1978.

[19] F.P. Preparata and M.I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[20] L.G. Valiant, A Bridging Model for Parallel Computation, *Communications of the ACM* 33 (1990), 103-111.