Some Scaleable Parallel Algorithms for Geometric Problems

Laurence Boxer [1] Russ Miller [2] Andrew Rau-Chaplin [3]

[1]Department of Computer and Information Sciences, Niagara University, NY 14109, USA. E-mail: boxer@niagara.edu. Research partially supported by a grant from the Niagara University Research Council.

[2]Department of Computer Science, State University of New York at Buffalo, Buffalo, New York 14260, USA. E-mail: miller@cs.buffalo.edu. Research partially supported by NSF grant IRI9412415.

[3]Faculty of Computer Science, Dalhousie University, 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.

Abstract

This paper considers a variety of geometric problems on input sets of size n using a *coarse grained multicomputer* model consisting of p processors with $\Omega(\frac{n}{p})$ local memory each (*i.e.*, $\Omega(\frac{n}{p})$ memory cells of $\Theta(\log n)$ bits apiece), where the processors are connected to an arbitrary interconnection network. It introduces efficient scaleable parallel algorithms for a number of geometric problems including the *rectangle finding problem*, a variety of *lower envelope problems*, the *maximal equally-spaced collinear points problem*, and the *point set pattern matching problem.* All of the algorithms presented are *scaleable* in that they are applicable and efficient over a very wide range of ratios of problem size to number of processors. In addition to the practicality imparted by scaleability, these algorithms are easy to implement in that all required communications can be achieved by a small number of calls to standard global routing operations.

Key words and phrases: parallel algorithms, computational geometry, scaleable algorithms, coarse grained multicomputer, lower envelope

1 Introduction

Computational geometry is an important area of research with applications in computer image processing, pattern matching, manufacturing, robotics, VLSI design, and so forth. A typical problem in parallel computational geometry calls for an efficient solution to a query involving n geometric objects on a parallel computer with p processors. Most previous theoretical work in parallel computational geometry has assumed fine grained parallelism, *i.e.*, $\frac{n}{p} = \Theta(1)$, for machine models including the PRAM, mesh, hypercube, and pyramid computer [A&L93, M&S96]. However, since most commercial parallel computers are coarse grained, it is desirable that parallel algorithms be *scaleable*, *i.e.*, implementable and efficient over a wide range of ratios of $\frac{n}{p}$. Recently, there has been growing interest in developing scaleable parallel algorithms for solving geometric problems on coarse grained machines (see [Boxe97, De&Dy95, DFR93, FRU95, DDDFK95]). This paper continues this effort by describing new scaleable algorithms for a variety of problems in pattern recognition, a variety of lower envelope problems (including a nearly-optimal algorithm for determining the lower envelope of polynomials), and other problems in computational geometry.

The paper is organized as follows.

- Section 2: We define the model of computation and discuss fundamental data movement operations.
- Section 3: We give a scaleable parallel algorithm to find all rectangles determined by a set of planar points, and we discuss straightforward solutions to related problems.
- Section 4: We give a nearly-optimal scaleable algorithm that yields a description of the lower envelope of polynomials of bounded degree, and we discuss a variety of scaleable parallel algorithms for problems whose solutions depend on such descriptions.
- Section 5: We give a scaleable parallel algorithm to find all maximal equally spaced collinear subsets of a finite point set in a Euclidean space.
- Section 6: We give scaleable parallel algorithms to find all subsets of a finite set in a Euclidean space that match, in the sense of geometric congruence, a given pattern.
- Section 7: We give some concluding remarks.

Some researchers feel that a "good" parallel algorithm is one with *work* (product of running time and number of processors) the same as, or perhaps only slightly more than, that of the best serial algorithm for the same problem. We note this goal is often impossible on certain parallel architectures; *e.g.*, on a fine grained mesh of n processors, optimal semigroup operations and sorting of evenly distributed data take $\Theta(n^{3/2})$ work [M&S96]; while, for problems of size n, serial semigroup operations require $\Theta(n)$ work and sorting requires $\Theta(n \log n)$. We feel that in the world of applications, users of parallel computers are often more interested in speed than in conservation of work efficiency. We feel that all algorithms presented in this paper are efficient, in that their running times are typically bounded above by an expression no larger than the sorting time for the volume of output (or, in some cases, the sorting time of a slightly larger volume of seemingly crucial intermediate data, *e.g.*, in our lower envelope problems). In section 2.3, we comment further on why we feel this is an appropriate standard of efficiency. All our algorithms show significant speedup as compared with the best serial solutions to their respective problems.

Preliminary versions of this paper appear in [BMR96a, BMR96b]. Some of the results presented in the current paper improve (in some cases, by correcting errors; in others, by demonstrating faster running times) results of [BMR96a, BMR96b].

2 Preliminaries

2.1 Model of Computation

The *Coarse Grained Multicomputer* model, or *CGM*(n,p) for short, considered in this paper consists of a set of p processors with $\Omega(\frac{n}{p})$ local memory each (*i.e.*, $\Omega(\frac{n}{p})$ memory cells of $\Theta(\log n)$ bits apiece in every processor), either connected to some arbitrary interconnection network or sharing global memory. Commonly used interconnection networks for a CGM include the 2D-mesh (*e.g.*, Intel Paragon), 3D-mesh (*e.g.*, Cray T30), 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, we will consider both local computation time and interprocessor communication time, in the standard way. The term "coarse grained" refers to the fact that the size $\Omega(\frac{n}{p})$ of each local memory is assumed to be "considerably larger" than $\Theta(1)$. Our definition of "considerably larger" will be that $\frac{n}{p} \geq p$. This implies that each processor has at least enough local memory to store the ID number of every other processor. Typically, commercial Coarse Grained Multicomputers like the IBM SP2, Cray T30, Intel Paragon, or TMC CM-5 have local memories ≥ 32 Mbytes. For a more detailed description of the model and its associated operations, see [DFR93].

Recently, there has been a growing interest in coarse grained computational models [Vali90, CKPSSSSE, H&K93] and the design of coarse grained geometric algorithms [DFR93, FRU95, DDDFK95]. The work on computational models has tended to be motivated by the observation that "fast algorithms" for fine-grained models rarely translate into fast code running on coarse grained machines. The BSP model, described by Valiant [Vali90], 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.* [CKPSSSSE] 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. Other coarse grained models, including the C^3 [H&K93] and the Coarse Grained Multicomputer (CGM) model used in this paper [DFR93], focus more on utilizing local computation and minimizing global operations.

The assumption $\frac{n}{p} \geq p$ (equivalently, $n \geq p^2$) implies, for example, that for a machine to process 100,000 data items over 100 processors, each processor must have a capacity of at least 1,000 data items. This is in contrast to the fine-grained model, in which each processor is expected to store only a small (*e.g.*, less than 10) number of data items.

2.2 Terminology, Notation, Assumptions

Throughout the paper, we use R^d to denote Euclidean $d-$dimensional space.

Sorting is used in most of the algorithms presented in this paper. We therefore assume that our data sets may be linearly ordered in some fashion that should be clear from context.

A set of $k-$tuples $X = \{(x_1, x_2, \ldots, x_k)\}$ is in *lexicographic order* if $(x_1, \ldots, x_k) < (x'_1, \ldots, x'_k)$ means

- $x_1 < x'_1$; or
- for some integer j, $1 \leq j < k$, $x_1 = x'_1$ and $x_2 = x'_2$ and $\ldots$ and $x_j = x'_j$ and $x_{j+1} < x'_{j+1}$.

2.3 Fundamental Operations

For a given problem, suppose T_{seq} and T_{par} are, respectively, the running times of the problem's best sequential and best parallel solutions. If $T_{par} = \Theta(\frac{T_{seq}}{p})$, then the parallel algorithm is optimal, to within a constant factor. In practice, analysis of a CGM algorithm usually must account for the time necessary for interprocessor communications and/or data exchanges (*e.g.*, in global sorting operations) in order to evaluate T_{par}. The time for these communications may be asymptotically greater than $\Theta(\frac{T_{seq}}{p})$.

We denote by $T_{sort}(n,p)$ the time required by the most efficient algorithm to sort $\Theta(n)$ data on a $CGM(n,p)$. Sorting is a fundamental operation that has been implemented efficiently on all models of parallel machines (theoretical and existing). Sorting is important not only in its own right, but also as a basis for a variety of parallel communications operations. In particular, each of the following data movement operations can be implemented via sorting.

- *Multinode broadcast:* Every processor sends the same $\Theta(1)$ data to every other processor.
- *Total exchange:* Every processor sends (not necessarily the same) $\Theta(1)$ data to every other processor.
- *Permutation exchange:* Let $\sigma : \{1, 2, \ldots, p\} \to \{1, 2, \ldots, p\}$ be a permutation (a function that is one-to-one and onto). Every processor P_i sends a list of $\frac{n}{p}$ data items to processor $P_{\sigma(i)}$ (*e.g.*, this operation could be used to rotate data circularly among sets of processors).
- *Semigroup operation:* Let $X = \{x_1, \ldots, x_n\}$ be data distributed evenly among the processors and let $\circ$ be a binary operation on X that is associative and that may be computed in $\Theta(1)$ serial time. Compute $x_1 \circ x_2 \circ \ldots \circ x_n$. Examples of such operations include *total, product, minimum, maximum, and,* and *or.*
- *Parallel prefix:* Let $X = \{x_1, \ldots, x_n\}$ be data distributed evenly among the processors and let $\circ$ be a binary operation on X that is associative and that may be computed in $\Theta(1)$ serial time. Compute all n members of $\{x_1,\ x_1 \circ x_2,\ \ldots,\ x_1 \circ x_2 \circ \ldots \circ x_n\}$.
- *Merge:* Let X and Y be lists of ordered data, each evenly distributed among the processors,

with $|X| + |Y| = \Theta(n)$. Combine these lists so that $X \cup Y$ is ordered and evenly distributed among the processors.

- *Parallel search:* Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be ordered lists (if necessary, we sort X and Y separately), each distributed evenly among the processors. Each $x_i \in X$ searches Y for a value y'_i or a range of values (in the latter case, we mean x_i "learns" the first and last indices of those members of Y with sort key in a given interval I_i).
- *Formation of combinations:* Let $X = \{x_1, \ldots, x_n\}$ and let k be a fixed positive integer, $1 < k < n$. Form the set of $\Theta(n^k)$ combinations of members of X that have exactly k members, $\{\{x_{i_1}, \ldots, x_{i_k}\} \mid 1 \le i_1 < i_2 < \ldots < i_k \le n\}$.
- *Formation of pairs from lists:* Let $X = \{x_1, \ldots, x_k\}$ and let $Y = \{y_1, \ldots, y_n\}$. Form all pairs (x_i, y_j), where $x_i \in X$, $y_j \in Y$.

The following result will be useful in comparing the resources required by problems of different sizes.

Lemma 2.1 *For positive integers k, n, p, we have*

$$k \cdot T_{sort}(n,p) = O(T_{sort}(kn,p)) \text{ on a } CGM(kn,p).$$

Proof: This follows from the fact that the work in sorting is superlinear in the amount of data being sorted. ■

Since $p^2 \le n$, the running times in the next result improve the $T_{sort}(n,p)$ times of [DFR93] for the same operations.

Proposition 2.2 *The following operations may be performed on a $CGM(n,p)$ in $T_{sort}(p^2,p)$ time.*

- *Multinode broadcast;*
- *Total exchange.*

Proof: We give a proof for multinode broadcast. A proof for total exchange is similar and is left to the reader. We give the following algorithm.

1. Let x_i be the data value to be sent by processor P_i, $i \in \{1, \ldots, p\}$, to all processors. In parallel, every processor P_i creates records $(x_i, 1), (x_i, 2), \ldots, (x_i, p)$. This takes $\Theta(p)$ time.

2. Sort the p^2 records created above by the second component, so that (x_i, j) ends up in P_j. This takes $T_{sort}(p^2, p)$ time.

The assertion follows. ■

Proposition 2.3 *A permutation exchange operation may be implemented in time $T_{sort}(n,p)$ on a $CGM(n,p)$.*

Proof: The following algorithm suffices.

1. Let σ be the permutation function of the operation. In parallel, each processor P_i sequentially assigns the tag value $\sigma(i)$ to each of its $\frac{n}{p}$ data items. This takes $\Theta(\frac{n}{p})$ time.

2. Sort the data by the tag values. This takes $T_{sort}(n,p)$ time.

Since the algorithm's running time is dominated by the sort step, the assertion follows. ■

The following improves the $T_{sort}(n,p)$ running time of [BMR96a].

Proposition 2.4 *A semigroup operation on evenly distributed data $x_1, \ldots, x_n$ may be implemented in time $\Theta(\frac{n}{p}) + T_{sort}(p^2, p)$ on a $CGM(n,p)$. At the end of this operation, all processors have the value of $X = x_1 \circ \ldots \circ x_n$.*

Proof: We give the following algorithm.

1. Without loss of generality, processor P_k, $k \in \{1, \ldots, p\}$, has the data values

$$x_{\frac{(k-1)n}{p}+1}, x_{\frac{(k-1)n}{p}+2}, \ldots, x_{\frac{kn}{p}}.$$

In parallel, each processor P_k computes its partial product

$$g_k = x_{\frac{(k-1)n}{p}+1} \circ x_{\frac{(k-1)n}{p}+2} \circ \ldots \circ x_{\frac{kn}{p}}.$$

This takes $\Theta(\frac{n}{p})$ time.

2. Perform a multinode broadcast, in which processor P_k sends g_k to all processors. By Proposition 2.2, this takes $T_{sort}(p^2, p)$ time.

3. Each processor computes $g_1 \circ g_2 \circ \ldots \circ g_p$ in $\Theta(p)$ time.

Since $p \leq n/p$, the assertion follows. ■

The following improves the $T_{sort}(n, p)$ running time of [BMR96a].

Proposition 2.5 *A parallel prefix operation may be implemented in time $\Theta(\frac{n}{p}) + T_{sort}(p^2, p)$ on a $CGM(n, p)$. At the end of the operation, the prefix $x_1 \circ x_2 \circ \ldots \circ x_i$ is in the same processor as x_i, $i \in \{1, 2, \ldots, n\}$.*

Proof: We give the following algorithm.

1. Without loss of generality, processor P_k, $k \in \{1, \ldots, p\}$, has the data values

$$x_{\frac{(k-1)n}{p}+1},\ x_{\frac{(k-1)n}{p}+2},\ \ldots,\ x_{\frac{kn}{p}}.$$

   In parallel, each processor P_k computes its prefixes $r_{\frac{(k-1)n}{p}+i}$, $i \in \{1, 2, \ldots, \frac{n}{p}\}$, defined by

$$r_{\frac{(k-1)n}{p}+1} = x_{\frac{(k-1)n}{p}+1},$$

$$r_{\frac{(k-1)n}{p}+i} = r_{\frac{(k-1)n}{p}+i-1} \circ x_{\frac{(k-1)p}{n}+i},\ i \in \{2, \ldots, \frac{n}{p}\}.$$

   This takes $\Theta(\frac{n}{p})$ time. Observe processor P_1 now has its desired prefixes, $r_1, r_2, \ldots, r_{\frac{n}{p}}$.

2. Perform a multinode broadcast operation in which processor P_k sends $r_{\frac{kn}{p}}$ to all processors. By Proposition 2.2, this takes $T_{sort}(p^2, p)$ time.

3. In parallel, each processor P_k, $2 \leq k \leq p$, computes the prefixes s_k and $t_{k,i}$, $i \in \{1, 2, \ldots, \frac{n}{p}\}$ given by

$$s_k = r_{\frac{n}{p}} \circ r_{\frac{2n}{p}} \circ \ldots \circ r_{\frac{(k-1)n}{p}},$$

$$t_{k,i} = s_k \circ r_{\frac{(k-1)n}{p}+i},\ i \in \{1, 2, \ldots, \frac{n}{p}\}.$$

   This is done in $\Theta(\frac{n}{p})$ time. The prefixes $t_{k,i}$ are the results desired of the algorithm that weren't already computed in the first step.

The assertion follows. ■

Proposition 2.6 *Let X and Y each be lists of ordered data, evenly distributed among the processors of a $CGM(n,p)$, where $|X| + |Y| = \Theta(n)$. Then X and Y may be merged in $T_{sort}(n,p)$ time.*

Proof: Sort the list $Z = X \cup Y$ in $T_{sort}(n,p)$ time. ■

Proposition 2.7 *Let X and Y each be lists of data, evenly distributed among the processors of a $CGM(k+n,p)$, where $|X| = k$ and $|Y| = n$. Then a parallel search, in which each member of X searches Y for a value or range of values, may be performed in $T_{sort}(k+n,p)$ time.*

Proof: We give the following algorithm for a search in which every member of X searches Y for a single value. Minor modifications give an algorithm in which every member of X searches Y for a range of values.

1. Sort X in $T_{sort}(k,p)$ time.
2. Let x'_i be the value sought by x_i. Let $X' = \{x'_1, \ldots, x'_k\}$. For each x_i, create a record r_i with components x_i, x'_i, and *report*. Let $R = \{r_1, \ldots, r_k\}$. This takes $\Theta(\frac{k}{p})$ time.
3. Sort $R \cup Y$, using the x'_i component of members of R as the key field. This takes $T_{sort}(k+n,p)$ time.
4. Use parallel prefix and postfix operations so every member of R learns whether or not its nearest member of Y in the sorted $R \cup Y$ has the desired x'_i value. If so, set the *report* field equal to the corresponding member of Y; otherwise, set the *report* field to $fail$. This takes $\Theta(\frac{k+n}{p}) + T_{sort}(p^2,p)$ time.
5. Sort the members of R (found in $R \cup Y$) by the x_i component. This takes $O(T_{sort}(k+n,p))$ time.
6. Each member of R is now in the processor in which it was created, and "reports" its *report* component to the corresponding x_i. This takes $\Theta(\frac{k}{p})$ time.

Thus, the algorithm takes $T_{sort}(k+n,p)$ time. ■

Proposition 2.8 *Let $X = \{x_1, \ldots, x_n\}$. Let $k > 1$ be a fixed integer. Then the set of all combinations of members of X with k members apiece, $\{\{x_{i_1}, \ldots, x_{i_k}\} \mid 1 \leq i_1 < i_2 < \ldots < i_k \leq n\}$ can be formed in*

$$\Theta(\frac{n^k}{p}) \; + \; p \cdot T_{sort}(n,p) \; = \; O(T_{sort}(n^k,p))$$

time on a $CGM(n^k,p)$. If $p^2 = O(\frac{n^{k-1}}{\log n})$ (which must happen when $k > 2$), the running time is $\Theta(\frac{n^k}{p})$, which is optimal.

Proof: The algorithm follows.

1. Use $p-1$ circular rotation operations of $\Theta(\frac{n}{p})$ data per processor so that each processor has the entire list X. This takes $p \cdot T_{sort}(n,p)$ time.

2. In parallel, each processor P_i computes one-p^{th} of all the $\Theta(n^k)$ combinations of k members of X. This takes $\Theta(\frac{n^k}{p})$ time.

Thus, the time required is $\Theta(\frac{n^k}{p}) \; + \; p \cdot T_{sort}(n,p)$. From Lemma 2.1, we have $p \cdot T_{sort}(n,p) = O(T_{sort}(np,p))$, which is (since $p < n$ and $k \geq 2$) $O(T_{sort}(n^k,p))$. Thus, the running time is $O(T_{sort}(n^k,p))$.

If we consider the sorting term in the running time, we have, since parallel sorting is faster than serial,

$$p \cdot T_{sort}(n,p) \; = \; O(np \log n) \; = \; O(\frac{np^2 \log n}{p}). \tag{1}$$

If $p^2 = O(\frac{n^{k-1}}{\log n})$ (which must happen if $k > 2$ since $p^2 \leq n$), it follows from statement (1) that

$$p \cdot T_{sort}(n,p) \; = \; O(\frac{n^k}{p}).$$

Thus, if $p^2 = O(\frac{n^{k-1}}{\log n})$, the running time is $\Theta(\frac{n^k}{p})$, which is optimal, since there is $\Theta(n^k)$ output. ■

Proposition 2.9 *Let $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_n\}$ be two lists evenly distributed among the processors of a $CGM(kn,p)$, with $p^{1/2} \leq k \leq n$. Then the set*

$$X \times Y \; = \; \{(x_i, y_j) \mid 1 \leq i \leq k, \; 1 \leq j \leq n\}$$

may be computed in

$$\Theta(\frac{kn}{p}) \ + \ p \cdot T_{sort}(k,p) \ = \ O(T_{sort}(kn,p))$$

time. If $p^2 \log k = O(n)$, *the running time reduces to* $\Theta(\frac{kn}{p})$, *which is optimal.*

Proof: Let $z_{ij} \ = \ (x_i, y_j)$, $1 \leq i \leq k$, $1 \leq j \leq n$. The following algorithm suffices.

1. Allocate space for the array

$$Z \ = \ \{z_{ij} \mid 1 \leq i \leq k, 1 \leq j \leq n\},$$

   its entries uninitialized, in $O(\frac{kn}{p})$ time.

2. Use $p-1$ circular rotations of X so that every processor has a copy of the entire list X. This takes $p \cdot T_{sort}(k,p)$ time, which, by Lemma 2.1 is $O(T_{sort}(kp,p)) \ = \ O(T_{sort}(kn,p))$.

3. Now every processor has all of X and its original share of Y. In parallel, every processor computes its share of $X \times Y$ corresponding to its share of Y in $\Theta(\frac{kn}{p})$ time.

Thus, the algorithm requires

$$\Theta(\frac{kn}{p}) \ + \ p \cdot T_{sort}(k,p) \ = \ O(T_{sort}(kn,p))$$

time.

Since parallel sorting is faster than serial, the sorting term in the running time is

$$p \cdot T_{sort}(k,p) \ = \ O(kp \log k) \ = \ O(\frac{kp^2 \log k}{p}).$$

If $p^2 \log k = O(n)$, it follows that this sorting term is

$$p \cdot T_{sort}(k,p) \ = \ O(\frac{kn}{p}),$$

and the running time is therefore $\Theta(\frac{kn}{p})$, which is optimal, since there is $\Theta(kn)$ output. ■

3 Rectangle Problems

In this section, we give a scaleable parallel algorithm to solve the *rectangle finding* or *all rectangles (AR)* problem. We say a polygon P is *from* $S \subset R^2$ if all vertices of P belong to S. The AR problem is to find all rectangles from S. A serial solution to this problem is given in [VK&D91].

Proposition 3.1 [VK&D91] *Let $S \subset R^2$, $|S| = n$. Then a solution to the AR problem has $\Theta(n^2 \log n)$ output in the worst case. Therefore, $\Omega(n^2 \log n)$ time is required for any serial algorithm that solves the AR problem.* ■

We have the following.

Theorem 3.2 *Let $S = \{v_0, v_1, \ldots, v_{n-1}\}$ be given as input. Then the AR problem can be solved in $T_{sort}(n^2 \log n, p)$ time on a $CGM(n^2 \log n, p)$.*

Proof: Note that a rectangle in R^2 may be determined by a pair of opposite sides with nonnegative slope. This observation allows us to avoid duplicate construction of rectangles. We give an algorithm with the following steps.

1. Form the set L of all line segments with endpoints in S and with nonnegative slopes, where each member of L is represented as a pair (v_i, v_j) of members of S such that $v_i < v_j$ with respect to lexicographic order. This may be done in $O(T_{sort}(n^2, p))$ time by a trivial modification to the algorithm associated with Proposition 2.8.

2. Define the order of the elements ℓ of L, in decreasing order of significance, by

   (a) slope;

   (b) length;

   (c) equation $ax+by+c=0$ (with first non-zero coefficient equal to 1) of the line perpendicular to ℓ at its first endpoint (the order of equations is the lexicographic order of triples (a, b, c)); and

   (d) the first endpoint of ℓ.

Note that in this order, if $\ell_0 < \ell_1 < \ell_2$ and (ℓ_0, ℓ_2) is a pair of opposite sides of a rectangle, then (ℓ_0, ℓ_1) and (ℓ_1, ℓ_2) are pairs of opposite sides of rectangles. Sort the members ℓ of L. This takes $T_{sort}(n^2, p)$ time.

3. Use parallel prefix operations to do the following. For each $\ell \in L$, determine the unique (if they exist) $\ell_0, \ell_1 \in L$ such that

   - $\ell_0 \leq \ell \leq \ell_1$, and
   - if $\ell_0 \leq \ell' \leq \ell_1$ and $\ell' \neq \ell$ then ℓ and ℓ' are opposite sides of a rectangle.

   Also determine, for each $\ell \in L$,
   $$r_\ell = ord(\ell_1) - ord(\ell),$$
   the number of rectangles for which ℓ is the first side, and
   $$P_\ell = \Sigma_{\ell' < \ell}\, r_{\ell'} \ ,$$
   the number of rectangles whose first sides precede ℓ. By Proposition 2.5, these operations require $\Theta(\frac{n^2}{p}) + T_{sort}(p^2, p)$ time.

4. Assign the first side of each of the $O(n^2 \log n)$ rectangles as follows. The i^{th} rectangle, $P_\ell < i \leq P_\ell + r_\ell$, gets ℓ as its first side. Since the values of the P_ℓ and r_ℓ may be assumed associated with the corresponding ℓ in the ordered set L, the first side of every rectangle can be found via parallel search operations in $T_{sort}(n^2 \log n, p)$ time.

5. Assign the second side (the one opposite the first side) of each of the $O(n^2 \log n)$ rectangles as follows. The i^{th} rectangle, $P_\ell < i \leq P_\ell + r_\ell$, has for its second side the member of L whose index in L is $ord(\ell) + (i - P_\ell)$. Thus, the second side of all rectangles may be determined via parallel search operations in $T_{sort}(n^2 \log n, p)$ time.

Thus, the running time of the algorithm is $T_{sort}(n^2 \log n, p)$. ■

Straightforward modifications to the algorithm of Theorem 3.2 yield the following (the output estimates are in [VK&D91, P&Sh92]):

Problem	Worst Case Output	T_{par}
All Isonormal Rectangles	$\Theta(n^2)$	$T_{sort}(n^2, p)$ on $CGM(n^2, p)$
All Squares	$\Theta(n^2)$	$T_{sort}(n^2, p)$ on $CGM(n^2, p)$

4 Lower Envelope Problems

4.1 Describing the Lower Envelope of Polynomials

Let S be a set of polynomial functions. Finding the *lower envelope* or minimum of S (equivalently, the *upper envelope* or maximum) is fundamental to the solution of a variety of interesting problems. The lower envelope of $S = \{f_i : R^1 \to R^1 \mid i = 1, \ldots, n\}$ is the function $LE : R^1 \to R^1$ defined by

$$LE(x) \;=\; \min\{f_i(x) \mid i = 1, \ldots, n\}.$$

We will abuse notation and refer to this function as $LE(S)$. We say a *piece of* $LE(S)$ is a pair (f_i, I), where $f_i \in S$ and I is a maximal interval on which $LE(x) = f_i(x)$ identically. Thus, the problem of describing $LE(x)$ is that of determining an ordered (by intervals) list of the pieces of $LE(x)$.

Let k be a fixed positive integer. Suppose the members of S are all polynomial functions of degree at most k. Then the maximal number of pieces of $LE(S)$ is denoted by $\lambda(n, k)$. It was shown in [Atal85a] that $\lambda(n, s)$ is the maximal length of a *Davenport-Schinzel sequence* [D&S65] defined by parameters n and s as follows.

Definition 4.1 [Atal85a] *Let n and s be positive integers. Let $C_n = \{c_1, \ldots, c_n\}$ be an alphabet of n distinct symbols. Let $L_{n,s}$ be the set of strings over C_n that do not contain any $c_i c_i$ as a substring and that do not contain as a subsequence of their characters any of the following "forbidden sequences" $E^s_{ij}, i \neq j$, defined for positive integer p by*

$$E^s_{ij} = \begin{cases} c_i c_j c_i & \text{if } s = 1 \\ E^{s-1}_{ij} c_j & \text{if } s = 2p \\ E^{s-1}_{ij} c_i & \text{if } s = 2p + 1. \end{cases}$$

The strings in $L_{n,s}$ are called Davenport-Schinzel sequences. ■

Notice that the presence of some E^s_{ij} as a *subsequence* of the characters of a string z, not necessarily as a *substring* of z, is sufficient to disqualify z from membership in $L_{n,s}$. For example,

$z = c_1c_2c_3c_1c_2 \notin L_{3,2}$, since z contains as a subsequence of its characters the sequence $E_{12}^2 = c_1c_2c_1c_2$, which is forbidden to members of $L_{3,2}$.

The following is a generalization of Lemma 2.4 of [B&M89a].

Lemma 4.2 *For all positive integers k, n, p, if p is a factor of n then*

$$p\lambda(n/p, k) \leq \lambda(n, k).$$

Proof: The lemma is stated in the form in which it will be used later in the paper. We note it suffices to prove the equivalent statement,

$$p\lambda(n, k) \leq \lambda(np, k), \text{ for all positive integers } k, n, p. \tag{2}$$

The proof of statement (2) is given by induction on p. For $p = 1$, the truth of statement (2) is obvious.

Now suppose statement (2) is true for $p \leq u$, for some positive integer u. Recall the alphabet of $L_{n,k}$ is $C_n = \{c_1, \ldots, c_n\}$. Let $m = \lambda(nu, k)$. Let $a \in L_{nu,k}$ be such that $|a| = m$. Let $a = a_1 \ldots a_m$ where $a_i \in C_{nu}, 1 \leq i \leq m$. Let $m' = \lambda(n, k)$. Let $z \in L_{n,k}$ be such that $|z| = m'$. Let $z = z_1 \ldots z_{m'}$ where $z_i \in C_n, 1 \leq i \leq m'$. Let $z' = z'_1 \ldots z'_{m'}$ where

$$z'_i = c_{nu+j} \text{ if } z_i = c_j, \ 1 \leq i \leq m'.$$

Then $z' \in L_{n,k}$ is defined over the alphabet $C'_n = \{c_{nu+1}, c_{nu+2}, \ldots, c_{n(u+1)}\}$, which is disjoint from the alphabet C_{nu} on which z is defined, and $|z'| = m'$. Hence $z'' = az' \in L_{n(u+1),k}$ and

$$\lambda(n(u+1), k) \geq |z''| = |a| + |z'| = m + m' = \lambda(nu, k) + \lambda(n, k)$$

$$\geq \text{ (by inductive hypothesis) } u\lambda(n, k) + \lambda(n, k) = (u+1)\lambda(n, k),$$

as desired. This completes the proof. ■

The function $\lambda(n, k)$ is, at worst, slightly more than linear in n. In the following, $\alpha(n)$ is the extremely slowly growing inverse Ackermann function (*c.f.*, [H&Sh86]). We have the following.

Theorem 4.3 *The following results concerning the function $\lambda(n, k)$ are known.*

- $\lambda(n,1) = n$ *and* $\lambda(n,2) = 2n-1$ [D&S65].
- $\lambda(n,3) = \Theta(n\,\alpha(n))$ [H&Sh86].
- $\lambda(n,4) = \Theta(n\,2^{\alpha(n)})$ [Agar91].
- *For* $s > 4$,

$$\lambda(n,s) \;=\; \begin{cases} O(n \cdot 2^{O([\alpha(n)]^{(s-2)/2})}) & \textit{if } s \textit{ is even;} \\ O(n \cdot 2^{O([\alpha(n)]^{(s-3)/2}\log(\alpha(n)))}) & \textit{if } s \textit{ is odd} \end{cases}$$

[AShSh89]. ■

In the following, we assume that k is a positive integer and that S is a set of functions, polynomials of degree at most k (or more generally, *k-intersecting* [Hersh89], *i.e.*, each pair of members of S has graphs that intersect in at most k points). As was done in [Atal85a, B&M89a, B&M89b, Hersh89], we also assume that for $\{f_i, f_j\} \subset S$, $i \neq j$, all solutions of the equation

$$f_i(x) \;=\; f_j(x)$$

may be determined in $\Theta(1)$ serial time. We note that somewhat different resources are required for the case of functions with a common interval domain than for the more general case.

Theorem 4.4 [Atal85a] *Let k be a fixed positive integer and let S be a set of polynomial functions, each of degree at most k. If all members of S are defined on the same interval, then the lower envelope of S has at most $\lambda(n,k)$ pieces generated by members of S and may be described in $O(\lambda(n,k)\log n)$ serial time.* ■

Theorem 4.5 [H&Sh86, Wi&Sh88] *Let k be a fixed positive integer and let S be a set of polynomial functions, each of degree at most k. If the domain of each member of S is an interval of R^1 (not necessarily the same interval for each member of S), then the lower envelope of S has at most $\lambda(n,k+2)$ pieces generated by members of S.* ■

Theorem 4.6 [Hersh89] *Let k be a fixed positive integer and let S be a set of polynomial functions, each of degree at most k. If the domain of each member of S is an interval of R^1 (not necessarily the same interval for each member of S), then a description of the lower envelope of S may be computed in $O(\lambda(n, k+1)\log n)$ serial time.* ■

Theorem 4.6 is perhaps surprising, in light of Theorem 4.5. One might expect that the serial time needed to produce the ordered list of $O(\lambda(n, k+2))$ pieces that describe the lower envelope would be $O(\lambda(n, k+2)\log n)$. However, the algorithm of [Hersh89] uses a clever insight to reduce, slightly, the running time to $O(\lambda(n, k+1)\log n)$.

For the following, it is useful to observe that $1 < p \leq n^{1/2}$ implies $log\frac{n}{p} = \Theta(\log n)$. We therefore will use the simpler $\log n$ in asymptotic expressions. We have the following (compare [DFR93]).

Theorem 4.7 *Let k be a fixed positive integer and let S be a set of polynomial functions, each of degree at most k. Assume that, initially, descriptions of the members of S are stored $O(\frac{n}{p})$ per processor. Then the lower envelope of S may be described using the following resources.*

- *$O[\lambda(\lambda(\frac{n}{p}, k), k+1)\log n + T_{sort}(p\lambda(\frac{n}{p}, k), p)]$ time on a $CGM(p\ \lambda(\lambda(\frac{n}{p}, k), k+2),\ p)$, if there is an interval $J \subset R^1$ such that J is the domain of each member of S.*
- *$O[\lambda(\lambda(\frac{n}{p}, k+2), k+1)\log n + T_{sort}(p\lambda(\frac{n}{p}, k+2), p)]$ time on a $CGM(p\ \lambda(\lambda(\frac{n}{p}, k+2), k+2),\ p))$, if the domain of each member of S is an interval in R^1 (not necessarily the same interval for each member of S).*

Proof: The following algorithm solves the problem for both cases. We analyze the cases separately.

1. Let S_j be the subset of S whose members are stored initially in processor P_j. In parallel, each processor P_j computes sequentially $LE(S_j)$.
   - If all members of S have the same interval domain, there are $O(\lambda(\frac{n}{p},\ k))$ pieces of $LE(S_j)$ stored in P_j. The time required is $O(\lambda(\frac{n}{p},\ k)\ \log n)$, by Theorem 4.4.
   - If all members of S have some (not necessarily the same) interval for domain, there are $O(\lambda(\frac{n}{p},\ k+2))$ pieces of $LE(S_j)$ stored in P_j. The time required is $O(\lambda(\frac{n}{p},\ k+1)\ \log n)$, by Theorem 4.6.

2. Globally sort the collection of pieces of $\cup_{j=1}^{p} LE(S_j)$ by the left endpoints of their intervals. In the case of a common interval domain for members of S, each processor P_j now has a new set V_j of $O(\lambda(\frac{n}{p}, k))$ pairs (f_i, I) as described above, where each pair is a piece of some $LE(S_{j'})$; in the more general case under consideration, $O(\lambda(\frac{n}{p}, k+2))$ such pairs. As a result of the sort, if $j < j'$, $(f_i, I_j) \in V_j$ and $(f_{i'}, I_{j'}) \in V_{j'}$, then the left endpoint of I_j is less than or equal to the left endpoint of $I_{j'}$.

   - For members of S having common interval domain, this step requires $T_{sort}(p\ \lambda(\frac{n}{p}, k),\ p)$ time.
   - For the more general case, this step requires $T_{sort}(p\ \lambda(\frac{n}{p}, k+2),\ p)$ time.

3. In parallel, each processor P_j computes sequentially $LE(V_j)$. Since the members of V_j need not have the same domain, we use the algorithm of Theorem 4.6 for both cases under consideration.

   - If the members of S have a common interval domain, the time required for this step is $O(\lambda(\lambda(\frac{n}{p}, k),\ k+1)\ \log n)$, and $LE(V_j)$ has $O(\lambda(\lambda(\frac{n}{p}, k),\ k+2))$ pieces.
   - In the more general case, this step takes $O(\lambda(\lambda(\frac{n}{p}, k+2),\ k+1)\ \log n)$ time, and $LE(V_j)$ has $O(\lambda(\lambda(\frac{n}{p}, k+2),\ k+2))$ pieces.

4. Let $R_j = (f_{i_j}, I_j)$ be the rightmost piece of $LE(V_j)$. This is the only piece of $LE(V_j)$ whose interval can intersect with the interval of a piece of $LE(V_{j'})$ for $j' > j$. Perform a multinode broadcast so that processor P_j sends R_j to all other processors. Hence, each processor now stores all of $R_1, \ldots, R_p$. By Proposition 2.2, this step requires $T_{sort}(p^2, p)$ time.

5. In parallel, each processor describes $LE(\{R_1, \ldots, R_p\})$ in $O(\lambda(p, k+1) \log p)$ time, using the algorithm of Theorem 4.6. The number of pieces of $LE(\{R_1, \ldots, R_p\})$ is $O(\lambda(p, k+2))$.

6. By our choice of the R_j, we can describe pieces of $LE(S)$ as follows. In parallel, each processor P_j merges the pieces of $LE(V_j)$ with those of $LE(\{R_1, \ldots, R_p\})$.

   - If the members of S have a common interval domain, this step takes $O(\lambda(\lambda(\frac{n}{p}, k),\ k+2))$ time, which, by Theorem 4.3, is $O(\lambda(\lambda(\frac{n}{p}, k), k+1) \log n)$.

- In the more general case we consider, this step takes $O(\lambda(\lambda(\frac{n}{p}, k+2),\ k+2))$ time, which, by Theorem 4.3, is $O(\lambda(\lambda(\frac{n}{p}, k+2), k+1)\log n)$.

7. There may be adjacent pieces with the same function, *i.e.*, the previous step may have created pieces (f_i, I) and (f_j, I') such that $i = j$ and the right endpoint of I coincides with the left endpoint of I'. Wherever this happens, we wish to combine the pairs into a single piece $(f_i, I \cup I')$. This can be done via a parallel prefix operation.

   - If the members of S have a common interval domain, the time required for this step is
$$O(\frac{p\ \lambda(\lambda(\frac{n}{p}, k),\ k+2)}{p}) \ + \ T_{sort}(p^2, p) \ = \ O(\ \lambda(\lambda(\frac{n}{p}, k),\ k+2)) \ + \ T_{sort}(p^2, p)$$
$$= \text{ (as above) } O[\lambda(\lambda(\frac{n}{p}, k), k+1)\log n \ + \ T_{sort}(p\lambda(\frac{n}{p}, k), p)].$$
   - In the more general case we consider, the time required for this step is
$$O(\frac{p\ \lambda(\lambda(\frac{n}{p}, k+2),\ k+2)}{p}) \ + \ T_{sort}(p^2, p) \ = \ O(\ \lambda(\lambda(\frac{n}{p}, k+2),\ k+2)) \ + \ T_{sort}(p^2, p)$$
$$= \text{ (as above) } O[\lambda(\lambda(\frac{n}{p}, k+2), k+1)\log n \ + \ T_{sort}(p\lambda(\frac{n}{p}, k+2), p)].$$

Thus, the required resources are as follows.

- For the case in which the members of S have the same interval domain, the algorithm uses $O[\ \lambda(\lambda(\frac{n}{p}, k), k+1)\log n \ + \ T_{sort}(p\lambda(\frac{n}{p}, k), p)]$ time on a $CGM(p\lambda(\lambda(\frac{n}{p}, k),\ k+2), p)$.
- For the case in which we assume that the members of S have (not necessarily the same) interval domains, the algorithm uses $O[\ \lambda(\lambda(\frac{n}{p}, k+2), k+1)\log n \ + \ T_{sort}(p\lambda(\frac{n}{p}, k+2), p)]$ time on a $CGM(p\lambda(\lambda(\frac{n}{p}, k+2),\ k+2), p)$.

■

In the algorithm of Theorem 4.7, we produce a sorted (by intervals) list of

- $O(\lambda(n, k))$ pieces in the case of all members of S having the same interval domain;
- $O(\lambda(n, k+2))$ pieces in the case of all members of S having (not necessarily the same) interval domains.

It follows by Theorem 4.3 and Lemma 4.2 that the time and memory resources required by our algorithm are very close to optimal.

In the following, we discuss several applications of Theorem 4.7. We will use the following abbreviations.

$$T_{env}(n,k,p) \ = \ \lambda(\lambda(\frac{n}{p},k),k+1)\log n \ + \ T_{sort}(p\lambda(\frac{n}{p},k),p);$$

$$CGM_{env}(n,k,p) \ = \ CGM(p \ \lambda(\lambda(\frac{n}{p},k),k+2), \ p);$$

$$T^{g}_{env}(n,k,p) \ = \ \lambda(\lambda(\frac{n}{p},k+2),k+1)\log n \ + \ T_{sort}(p\lambda(\frac{n}{p},k+2),p);$$

$$CGM^{g}_{env}(n,k,p) \ = \ CGM(p \ \lambda(\lambda(\frac{n}{p},k+2),k+2), \ p).$$

Following [Atal85a], we say the function $f(t)$ has a *jump discontinuity at* u if both $\lim_{t\to u^+} f(t)$ and $\lim_{t\to u^-} f(t)$ exist, and $\lim_{t\to u^+} f(t) \neq \lim_{t\to u^-} f(t)$; and the function $f(t)$ has a *transition* at t_0 if $f(t)$ switches between being defined and undefined on either side of t_0.

The next result is a generalization of the complexity bound for lower envelope functions to functions with transitions and jump discontinuities.

Lemma 4.8 [B&M89a] *Let k be a positive integer. Let $f_1,\ldots,f_n$ be real-valued functions of time, such that (a) every f_i is continuous except for at most p_i jump discontinuities, (b) every f_i has at most q_i transitions, where (c) $p_i + q_i \leq k$, and (d) no pair of distinct functions f_i and f_j intersect more than s times. Then $h(t) = \min\{f_1(t),\ldots,f_n(t)\}$ has no more than $\lambda(n,s+2k)$ pieces generated by $\{f_1,\ldots,f_n\}$.* ■

Theorem 4.9 *Let s and k be positive integers. Let $f_1,\ldots,f_n$ be as in Lemma 4.8. Assume also that the f_i satisfy*

- *each f_i has a $\Theta(1)$ storage description;*
- *each value $f_i(t)$ may be computed in $\Theta(1)$ time by a single processor; and*
- *for $i \neq j$, there are at most s distinct real solutions to the equation $f_i(t) \ = \ f_j(t)$, all of which can be found by a single processor in $\Theta(1)$ time.*

Then the function $h(t) = \min\{f_1(t), \ldots, f_n(t)\}$ *can be constructed in* $T_{env}(n, s+2k, p)$ *time by a* $CGM_{env}(n, s+2k, p)$.

Proof: The assertion may be proved by an argument that is virtually identical to that given for Theorem 4.7. ■

The next two lemmas will be useful when we combine piecewise defined functions.

Lemma 4.10 [B&M89a] *Let* $f(t)$ *and* $g(t)$ *be functions from* R^1 *to* R^1. *Let* m *and* n *be positive integers. Suppose* $f(t)$ *has* m *pieces and* $g(t)$ *has* n *pieces. Then the intervals of pieces of* $f(t)$ *have, altogether, at most* $m+n$ *nondegenerate intersections with the intervals of pieces of* $g(t)$. ■

Lemma 4.11 [B&M89a] *Let* m *and* k *be positive integers. Let* $f(t)$ *and* $g(t)$ *be functions from* R^1 *to* R^1. *Suppose that for every piece of both* $f(t)$ *and* $g(t)$, *the function of the piece is a polynomial whose degree is at most* k. *Assume that the intervals of the pieces of* $f(t)$ *have* m *nondegenerate intersections with the intervals of the pieces of* $g(t)$. *Then the function* $\min\{f(t), g(t)\}$ *has at most* $m(k+1)$ *pieces.* ■

4.2 Minimization of Hausdorff Distance

The Hausdorff distance [Nadl78] is a measure of how well two sets A and B resemble each other with respect to their locations; if A and B are nonempty finite subsets of a Euclidean space, 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.

In this section, we let $d(a, b) = |a - b|$ be the Euclidean metric for R^1. We abuse notation and write

$$d(z, A) = \min\{d(z, a) \mid a \in A\}.$$

The "nonsymmetric" or "one-way" Hausdorff measure is

$$H^*(A, B) = \max_{a \in A} d(a, B).$$

Thus, we have the following.

Proposition 4.12 *Let $A \cup B \subset R^1$, $|A| = m$, and $|B| = n$. Then*

$$H^*(A,B) = \max\{H^*(\{a\},B) \mid a \in A\}. \blacksquare$$

The Hausdorff metric $H(A,B)$ is defined [Nadl78] by

$$H(A,B) = \max\{H^*(A,B), H^*(B,A)\}.$$

For $A \subset R^1$, $t \in R^1$, let

$$A + t = \{a + t \mid a \in A\}.$$

In [Röte91], a serial algorithm is given to solve the following problem: Let A and B be finite subsets of the real line. Find a translation t of A so that the Hausdorff distance $H(A+t,B)$ is minimized. The algorithm of [Röte91] is dominated by the description of the function $H(A+t,B)$, which is an upper envelope problem. We give in Theorem 4.18 an efficient algorithm to solve this problem on a coarse grained multicomputer.

It will be useful to assume the members of A and those of B are ordered:

$$a_1 < a_2 < \ldots < a_m, \quad b_1 < b_2 < \ldots < b_n.$$

There is no loss of generality in making such an assumption, since if not initially known to be true, this state can be achieved in $T_{sort}(m,p) + T_{sort}(n,p)$ time on a $CGM(m+n,p)$. In order to prove Theorem 4.18, we use the following.

Lemma 4.13 [Röte91] *Let $A \cup B \subset R^1$, $|A| = m$, and $|B| = n$. Suppose $a_1 = b_1$. Then, for all $t \in R^1$, $H(A+t,B) \geq |t|$.* $\blacksquare$

For each $a \in A$, let $f_a : R^1 \to R^1$ be the function

$$f_a(t) = H^*(\{a+t\}, B).$$

The assertions of the following Lemma are found in [Röte91]. We give a proof to clarify our methods.

Lemma 4.14 *Let $a \in A$. Then f_a is a continuous, piecewise linear function for which the graph of each linear piece has slope in $\{-1,1\}$.*

Proof: First, we show f_a is piecewise linear, with slopes in $\{-1, 1\}$. We consider the following cases.

1. Suppose $a + t_0 \leq b_1$. Then t_0 belongs to an interval I_0 on which $f_a(t) = b_1 - (a + t)$. Thus, on I_0, f_a has slope -1.

2. Similarly, if $a + t_1 \geq b_n$, then t_1 belongs to an interval I_1 on which f_a has slope 1.

3. The only remaining possibility is that there are consecutive members b_i, b_{i+1} of B such that $b_i \leq a + t \leq b_{i+1}$. This requires consideration of two subcases.

   - If t_3 is such that $b_i \leq a + t_3 \leq (b_i + b_{i+1})/2$, then t_3 belongs to an interval I_3 on which $f_a(t) = a + t - b_i$. Hence, on I_3, f_a has slope 1.

   - If t_4 is such that $(b_i + b_{i+1})/2 \leq a + t_4 \leq b_{i+1}$, then t_4 belongs to an interval I_4 on which $f_a(t) = b_{i+1} - (a + t)$. Hence, on I_4, f_a has slope -1.

Thus, for all $t \in R^1$, t belongs to an interval on which the graph of f_a has slope -1 or 1.

That f_a is continuous follows from the fact that common endpoints of intervals discussed above must belong to one of the following cases.

- $a + t = b_i \in B$. Then the formulas for both of the pieces of f_a whose intervals intersect at such a value of t give $f_a(t) = 0$.

- $a+t = (b_i+b_{i+1})/2$ for some pair b_i, b_{i+1} of consecutive members of B. Then the formulas for both of the pieces of f_a whose intervals intersect at such a value of t give $f_a(t) = (b_{i+1} - b_i)/2$. $\blacksquare$

Lemma 4.15 *Let $a \in A$.*

- *Suppose there exists $t_0 \geq 0$ such that $f_a(t_0) \leq t_0$. Then, for all $t \geq t_0$, $f_a(t) \leq t$.*

- *Suppose there exists $T_0 \leq 0$ such that $f_a(T_0) \leq |T_0|$. Then, for all $t \leq T_0$, $f_a(t) \leq |t|$.*

Proof: Let $t_0 \geq 0$ be such that $f_a(t_0) \leq t_0$. Let I be the interval of the linear piece of f_a such that $t_0 \in I$. First, we claim that

$$\text{for all } t \in I \text{ such that } t \geq t_0,\ f_a(t) \leq t. \tag{3}$$

This claim follows from Lemma 4.14.

Suppose there is a $t_1 > t_0$ such that $f_a(t_1) > t_1$. It follows from statement (3) that t_0 and t_1 belong to intervals of distinct pieces of f_a. Let t_2 be the left endpoint of the interval of f_a containing t_1. Without loss of generality, we may assume t_2 is minimal among endpoints t of intervals I of pieces of f_a such that $t > t_0$ and such that I has a point t_* satisfying $f_a(t_*) > t_*$.

Then t_2 is a right endpoint of a piece of f_a on whose interval $f_a(t) \leq t$, by (3). On the interval $[t_2, t_1]$, f_a is continuous and differentiable, and $\frac{f_a(t_1)-f_a(t_2)}{t_1-t_2} > 1$. It follows from the Mean Value Theorem of calculus that there exists t_3 such that $t_2 < t_3 < t_1$ and $f_a'(t_3) > 1$. This contradicts Lemma 4.14. It follows that $t > t_0$ implies $f_a(t) \leq t$.

The proof of the second assertion is similar and is omitted. ■

Let $F : R^1 \to R^1$ be a piecewise-defined function and let (f, I) be a piece of F. Let $H : R^1 \to R^1$ be a piecewise-defined function. We say (f, I) *contributes to* H if there is a subinterval J of I such that $H(t) = f(t)$ for all $t \in J$.

We define the following functions:

- $id : R^1 \to R^1$ is defined by $id(t) = t$ for all $t \in R^1$.
- $-id : R^1 \to R^1$ is defined by $-id(t) = -t$ for all $t \in R^1$.
- For a fixed $a \in R^1$, $\alpha_{a,i} : R^1 \to R^1$ is defined by $\alpha_{a,i}(t) = a + t - b_i$, for all $t \in R^1$.
- For a fixed $a \in R^1$, $\beta_{a,i} : R^1 \to R^1$ is defined by $\beta_{a,i}(t) = b_{i+1} - (a + t)$, for all $t \in R^1$.

The following Proposition is essentially found in [Röte91], where it is stated in somewhat lesser detail than is given below.

Proposition 4.16 *Let $A \cup B \subset R^1$, $|A| = m$, and $|B| = n$. Suppose $a_1 = b_1$. Then we have the following.*

- *If $a < b_1$, the piece of f_a that may contribute to $H(A + t, B)$ is $(\beta_{a,0}, (-\infty, b_1 - a])$.*
- *If $a = b_1$, the pieces of f_a that may contribute to $H(A + t, B)$ are*

$$(-id,\ (-\infty, 0]) \text{ and } (id,\ [0, \frac{b_2 - b_1}{2}]).$$

- *If* $a = b_i$ *for* $i \in \{2, \ldots, n-1\}$*, the pieces of* f_a *that may contribute to* $H(A+t, B)$ *are*

$$(-id,\ [-\frac{b_i - b_{i-1}}{2}, 0]) \text{ and } (id,\ [0, \frac{b_{i+1} - b_i}{2}]).$$

- *If* $a = b_n$*, the pieces of* f_a *that may contribute to* $H(A+t, B)$ *are*

$$(-id,\ [-\frac{b_n - b_{n-1}}{2}, 0]) \text{ and } (id,\ [0, \infty)).$$

- *If* $b_i < a < b_{i+1}$*, the pieces of* f_a *that may contribute to* $H(A+t, B)$ *are*

$$(\alpha_{a,i},\ [-(a - b_i), \frac{b_i + b_{i+1}}{2} - a]) \text{ and } (\beta_{a,i},\ [\frac{b_i + b_{i+1}}{2} - a, b_{i+1} - a]).$$

- *If* $a > b_n$*, the piece of* f_a *that may contribute to* $H(a+t, B)$ *is*

$$(\alpha_{a,n}, [-(a - b_n), \infty)).$$

Proof: That the pieces claimed indeed are pieces of f_a may easily be checked by the reader. That no other pieces of f_a may contribute to $H(a+t, B)$ follows from Lemma 4.13 and Lemma 4.15. ■

Proposition 4.17 *Let* $A \cup B \subset R^1$*,* $|A| = m$*, and* $|B| = n$*. Suppose* $a_1 = b_1$*. Then a description of the function* $H(A+t, B)$ *may be computed in*

$$O[\lambda(\lambda(\frac{m}{p}, 3), 2) \log m \ + \ \lambda(\lambda(\frac{n}{p}, 3), 2) \log n \ + \ T_{sort}(\lambda(m, 3) \ + \ \lambda(n, 3), p)]$$

time on a $CGM^g_{env}(m+n, 1, p)$*, and this function has* $O(\lambda(m, 3) \ + \ \lambda(n, 3))$ *pieces.*

Proof: We give the following algorithm.

1. For each $a \in A$, compute the (at most) two pieces of f_a that can contribute to $H(A+t, B)$ described in Lemma 4.16. Let us denote these pieces by $f_{a,1}$ and $f_{a,2}$. This is done via a parallel search step in which each $a \in A$ finds the corresponding b_i and b_{i+1} discussed in Proposition 4.16, followed by sequential (within processors executing in parallel) construction of the pieces, in $T_{sort}(m+n, p)$ time.

2. Compute a description of the function $H_A : R^1 \to R^1$, defined as the upper envelope of $\{f_{a_i,1}, f_{a_i,2}\}_{i=1}^m$. By Theorem 4.7, this requires $T_{env}^g(m,1,p)$ time, and H_A has $O(\lambda(m,3))$ pieces. Note $H_A(t)$ may not be identically equal to $H^*(A+t,B)$; however, it follows from Proposition 4.12 and Proposition 4.16 that any piece of $H(A+t,B)$ that is contributed by a piece of $H^*(A+t,B)$ is contributed by a piece of $H_A(t)$.

3. Note that the function $H^*(B, A+t)$ is identical to the function $H^*(B-t, A)$. Therefore, we may similarly execute analogs of the previous steps to compute a description of the function $H_B : R^1 \to R^1$, analogous to H_A, from the (at most) two pieces of $f_{b,1}$ and $f_{b,2}$, for all $b \in B$, that may contribute to the function $H(A+t,B)$. This requires $T_{sort}(m+n,p) \; + \; T_{env}^g(n,1,p)$ time, and the function H_B has $O(\lambda(n,3))$ pieces. The function $H_B(t)$ has a similar relationship with $H^*(B,A+t)$ as that between H_A and $H^*(A+t,B)$.

4. Compute the function $H(A+t,B)$, which is the upper envelope of the functions H_A and H_B. This step can be performed by a merge-like operation of the pieces of H_A and those of H_B in $T_{sort}(\lambda(m,3) \; + \; \lambda(n,3), \; p)$ time, followed by a parallel prefix operation to combine adjacent pieces with the same function description into a single piece, in
$$O(\frac{\lambda(m,3) \; + \; \lambda(n,3)}{p}) \; + \; T_{sort}(p^2,p) \; = \; \text{(by Theorem 4.3)}$$
$$O(T_{env}^g(m,1,p) \; + \; T_{env}^g(n,1,p))$$
time. By Lemma 4.10, the function $H(A+t,B)$ has $O(\lambda(m,3) \; + \; \lambda(n,3))$ pieces.

It follows from our definition of $T_{env}^g(n,k,p)$ that the time required by our algorithm is
$$O[\lambda(\lambda(\frac{m}{p},3),2)\log m \; + \; \lambda(\lambda(\frac{n}{p},3),2)\log n \; + \; T_{sort}(\lambda(m,3) \; + \; \lambda(n,3),p)]. \; \blacksquare$$

We now prove the main result of this section.

Theorem 4.18 *Let $A \cup B \subset R^1$, $|A| \; = \; m$, and $|B| \; = \; n$. Then a translation t of A that minimizes the Hausdorff distance $H(A+t,B)$ may be described on a $CGM_{env}^g(m+n,1,p)$ in*
$$O[\lambda(\lambda(\frac{m}{p},3),2)\log m \; + \; \lambda(\lambda(\frac{n}{p},3),2)\log n \; + \; T_{sort}(\lambda(m,3) \; + \; \lambda(n,3),p)].$$
time.

Proof: We give the following algorithm.

1. Translate A by
$$t_0 = b_1 - a_1$$
to $A' = A + t_0 = \{a_i + t_0 \mid i = 1, \ldots, m\}$. Let $a'_i = a_i + t_0$, $i = 1, \ldots, m$. Note that $a'_1 = b_1$. This is done as follows.
   - Broadcast the values of a_1 and b_1 to all processors. This requires $O(p)$ time.
   - In parallel, all processors compute t_0. This takes $\Theta(1)$ time.
   - In parallel, every processor adds t_0 to each of its members of A to obtain the corresponding members of A'. This requires $\Theta(\frac{m}{p})$ time.
2. Compute a description of the function $H(A'+t, B)$ via the algorithm of Proposition 4.17. This takes
$$O[\lambda(\lambda(\frac{m}{p}, 3), 2) \log m \ + \ \lambda(\lambda(\frac{n}{p}, 3), 2) \log n \ + \ T_{sort}(\lambda(m, 3) \ + \ \lambda(n, 3), p)].$$
time, and there are $O(\lambda(m, 3) + \lambda(n, 3))$ pieces.
3. In parallel, every processor computes the minimum value attained by each of its pieces of $H(A' + t, B)$ and notes the value of t that yields the minimum value for the piece. Since every piece is a linear function on an interval, it takes $\Theta(1)$ time to determine the minimum value of a piece of $H(A' + t, B)$. Hence, this step requires $O(\frac{\lambda(m,3)+\lambda(n,3)}{p})$ time.
4. Let t_1 be a value of t that yields a minimum value for the function $H(A' + t, B)$. The value of t_1 is determined by performing a minimum operation on the piecewise minima determined in the previous step. By Proposition 2.4, this takes $O(\frac{\lambda(m,3)+\lambda(n,3)}{p}) \ + \ T_{sort}(p^2, p))$ time.
5. A translation parameter t_2 such that
$$H(A + t_2, B) = \min\{H(A + t, B) \mid t \in R^1\}$$
is now obtained via $t_2 = t_0 + t_1$. Since all processors have the values of t_0 and t_1, all processors computes t_2 in $\Theta(1)$ time.

Thus, the time required by our algorithm is

$$O[\lambda(\lambda(\frac{m}{p},3),2)\log m \ + \ \lambda(\lambda(\frac{n}{p},3),2)\log n \ + \ T_{sort}(\lambda(m,3) \ + \ \lambda(n,3),p)].$$

■

4.3 Common Intersections of Polygons

In [Reic88, B&M90], serial and fine-grained parallel algorithms are given to solve the *Common Intersection Problem* for *vertically convex* polygons (a polygon P is vertically convex if for every pair of points $\{x,y\} \in P$, if x and y are on the same vertical line segment s, then $s \subset P$). The Common Intersection Problem is that of determining whether a collection of subsets of the Euclidean plane R^2 has a common intersection, and, if so, describing the intersection. We have the following.

Theorem 4.19 *Let S be a set of vertically convex polygons in R^2 whose boundaries have a total of n line segments. Then the Common Intersection Problem for S can be solved on a $CGM^g_{env}(n,1,p)$ in*

$$O[\lambda(\lambda(\frac{n}{p},3),2)\log n \ + \ T_{sort}(\lambda(n,3),p)] \text{ time.}$$

Proof: We assume input to the problem consists of a description of n line segments representing the boundaries of k vertically convex polygons, $F_1,\ldots,F_k$, where $1 \leq k < n$, with each line segment labeled by the polygon to which it belongs, such that the line segments of the same polygon are consecutive in the input and are given in circular order. Our algorithm follows.

1. For each F_i, determine a leftmost and a rightmost vertex, l_i and r_i, respectively. Since the edges are given in circular order within polygons, associate with each edge of F_i the values of l_i and r_i. This is done via parallel prefix operations in $\Theta(\frac{n}{p}) \ + \ T_{sort}(p^2,p)$ time.

2. Use l_i and r_i to determine the upper and lower boundaries U_i and L_i of F_i. Each of U_i and L_i is a connected union of edges of F_i that forms a path from l_i to r_i. Since the edges of F_i are in circular order, this may be done in $\Theta(\frac{n}{p}) \ + \ T_{sort}(p^2,p)$ time via parallel prefix operations.

3. Compute a description of the lower envelope function $f(t)$ of the edges in $\cup_{i=1}^{k}U_i$ and a description of the upper envelope function $g(t)$ of the edges in $\cup_{i=1}^{k}L_k$. By Theorem 4.7, this

takes $T^g_{env}(n,1,p)$ time, and by Theorem 4.5, each of these envelope functions has $O(\lambda(n,3))$ pieces.

4. A point $(t_0, y) \in \cap_{i=1}^k F_i$ if and only if (t_0, y) is below (or on) the graph of $f(t)$ and above (or on) the graph of $g(t)$, with
$$B \leq t_0 \leq C, \tag{4}$$
where B and C are the abscissas of the rightmost of $\{l_i\}_{i=1}^k$ and the leftmost of $\{r_i\}_{i=1}^k$, respectively. We determine whether such a point exists, as follows.
   - Compute a description of the function $f(t) - g(t)$. This may be done by a merge-like operation on the pieces of f and the pieces of g, in $T_{sort}(\lambda(n,3),p)$ time. By Lemma 4.10, $f(t) - g(t)$ has $O(\lambda(n,3))$ pieces.
   - Compute each of B and C. This may be done in $O(\frac{n}{p}) + T_{sort}(p^2,p)$ time via semigroup operations. At the end of this step, every processor has the values of B and C.
   - Examine the $O(\lambda(n,3))$ pieces of $f(t) - g(t)$ to see if there is a piece that attains a nonnegative value at some t_0 satisfying inequalities (4). Since $f(t)-g(t)$ has linear pieces, it takes $\Theta(1)$ time for a processor to examine one piece. Hence, each processor examines its share of the pieces in $O(\frac{\lambda(n,3)}{p})$ time. A description of the common intersection points may be obtained by noting, on each piece of $f(t) - g(t)$, the subinterval J of the piece satisfying

     (a) $t \in J$ implies $B \leq t \leq C$, and

     (b) $t \in J$ implies $f(t) - g(t) \geq 0$.

It follows from Lemma 4.2 and our definition of $T^g_{env}(n,k,p)$ that the time our algorithm requires is
$$O[\lambda(\lambda(\frac{n}{p},3),2)\log n \ + \ T_{sort}(\lambda(n,3),p)].$$
■

Next, we give a slight generalization of Theorem 4.19 that may be used to solve the Common Intersection Problem for planar figures with curved boundaries, *e.g.*, circular disks or figures whose boundaries are graphs of polynomial functions. The proof is not given, as it is virtually identical with that given for Theorem 4.19.

Theorem 4.20 *Let k, m, n be integers, $0 < k < n$. Let $f_1, \ldots, f_m$ be real-valued functions of a real variable such that*

- *each f_i has a $\Theta(1)$ storage description;*
- *each value $f_i(x)$ may be computed in $\Theta(1)$ time by a single processor; and*
- *for $i \neq j$, there are at most k distinct real solutions to the equation $f_i(x) = f_j(x)$, all of which can be found by a single processor in $\Theta(1)$ time.*

Let S be a set of vertically convex subsets of R^2 such that the union of the boundaries of members of S is the union of n pieces of the graphs $y = f_i(x)$, $i \in \{1, \ldots, m\}$. Then the intersection of the members of S may be described on a $CGM^g_{env}(n, k, p)$ in

$$O[\lambda(\lambda(\frac{n}{p}, k+2), k+1) \log n \ + \ T_{sort}(\lambda(n, k+2), p)].$$

time. ■

4.4 Dynamic Computational Geometry

Problems concerning geometric properties of moving point-objects were considered in [Atal85a, B&M89a, B&M89b]. Sequential algorithms are presented in [Atal85a], while fine-grained parallel algorithms are presented in [B&M89a, B&M89b].

We have obtained efficient scaleable parallel algorithms for many of the problems discussed in the papers cited above. All have running times dominated by description of lower or upper envelopes and data movements of an envelope's pieces. In this section, we 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 $s_i \in S$, the location of s_i at time t is described by a vector-valued function

$$\mathbf{f}_i(t) \ = \ [f_i^1(t), \ldots, f_i^d(t)],$$

such that each Cartesian coordinate function f_i^j is a polynomial in t of degree at most k. We refer to such motion as $k-motion$ [B&M89a].

4.4.1 Nearest Neighbor

We have the following.

Theorem 4.21 *Let d and k be fixed positive integers. Let S be a system of n point-objects, each of which is in $k-$motion in R^d. Then, as a function of t, a nearest member of $S \setminus \{s_0\}$ to s_0 may be described in $T_{env}(n, 2k, p)$ time on a $CGM_{env}(n, 2k, p)$.*

Proof: For $j \in \{1, 2, \ldots, n-1\}$, let

$$d_j(t) \;=\; d(\mathbf{f}_0(t), \mathbf{f}_j(t)),$$

where d indicates the Euclidean distance function. Note $[d_j(t)]^2$ is a polynomial of degree at most $2k$. Since

$$d_j(t) \;=\; \min\{d_1(t), \ldots, d_{n-1}(t)\} \text{ if and only if } [d_j(t)]^2 \;=\; \min\{[d_1(t)]^2, \ldots, [d_{n-1}(t)]^2\},$$

it follows that the problem reduces to describing $LE(\{[d_1(t)]^2, \ldots, [d_{n-1}(t)]^2\})$. The assertion follows from Theorem 4.7. ∎

4.4.2 Containment in an Iso-Oriented Hyperrectangle

We have the following.

Theorem 4.22 *Let d and k be fixed positive integers. Let $X_1, \ldots, X_d$ be fixed positive numbers. Let S be a system of point-objects, each of which is in $k-$motion in R^d. Then, as a function of t, the time intervals when an iso-oriented hyperrectangle of dimensions $X_1, \ldots, X_d$ contains S may be determined in $O[\lambda(\lambda(\frac{n}{p}, k), k+1)\log n \;+\; T_{sort}(\lambda(n,k), p)]$ time on a $CGM_{env}(n, k, p)$.*

Proof: We give the following algorithm.

1. For $j = 1, \ldots, d$, let $f_i^j(t)$ be the j^{th} coordinate function of $\mathbf{f}_i(t)$. Compute descriptions of all the functions

   $$m_j(t) = \min\{f_0^j(t), \ldots, f_{n-1}^j(t)\},$$

   and

   $$M_j(t) = \max\{f_0^j(t), \ldots, f_{n-1}^j(t)\},$$

   $j = 1, \ldots, d$. It follows from Theorem 4.7 that all can be described in $T_{env}(n, k, p)$ time.

2. For $j = 1, \ldots, d$, describe all the functions

$$D_j(t) \;=\; M_j(t) - m_j(t),\; j \;=\; 1, \ldots, d.$$

   Since each of the functions M_j and m_j has $O(\lambda(n,k))$ pieces, this step can be done by merging the pieces of M_j and those of m_j in $T_{sort}(\lambda(n,k),p)$ time (Proposition 2.6). Note D_j has $O(\lambda(n,k))$ pieces, by Lemma 4.10.

3. For $j = 1, \ldots, d$, describe all the functions

$$w_j(t) \;=\; D_j(t) - X_j,$$

   as follows. Broadcast the values of $X_1, \ldots, X_d$ to all processors in $O(p)$ time. Then, in $O(\frac{\lambda(n,k)}{p})$ time, each processor sequentially computes the appropriate difference in each of its pieces of the members of $\{D_1(t), \ldots, D_d(t)\}$.

4. For $j = 1, \ldots, d$, describe all the functions

$$W_j(t) \;=\; \begin{cases} 1 & \text{if } w_j(t) \leq 0; \\ 0 & \text{otherwise.} \end{cases}$$

   Each piece of w_j generates at most $k+1$ constant-valued pieces of W_j in $\Theta(1)$ serial time; hence W_j has $O(\lambda(n,k))$ constant-valued pieces. This step requires $O(\frac{\lambda(n,k)}{p})$ time.

5. Now describe the product function $\pi(t)$ of $\{W_1, \ldots W_d\}$. As above, this function has $O(\lambda(n,k))$ pieces and may be described in $T_{sort}(\lambda(n,k),p)$ time via $\lceil \log d \rceil \;=\; \Theta(1)$ merge-like steps, starting at the lowest level with the pieces of pairs of $\{W_1, \ldots, W_d\}$. We note that S is contained in an iso-oriented hyperrectangle of the specified dimensions precisely during those intervals of time corresponding to pieces of π when $\pi(t) = 1$.

The assertion follows from Lemma 4.2 and the definition of $T_{env}(n,k,p)$. ■

4.4.3 Smallest Containing Hypercube

A problem related to that discussed in Section 4.4.2 is the description, as a function of time, of the edgelength of the smallest rectilinear, iso-oriented hypercube that contains S at time t. Here, by "hypercube" we mean a hyperrectangle in which all edges have the same size. We have the following.

Theorem 4.23 *Let d and k be fixed positive integers. Let S be a system of point-objects, each of which is in $k-$motion in R^d. Then the function $E(t)$, the edgelength of the smallest rectilinear, iso-oriented hypercube that contains S at time t, can be described in*

$$O[\lambda(\lambda(\frac{n}{p},k),k+1)\log n \ + \ T_{sort}(\lambda(n,k),p)]$$

time on a $CGM_{env}(n,k,p)$.

Proof: We give the following algorithm.

1. Compute descriptions of all the functions $D_1(t),\ldots,D_d(t)$ that represent the edgelengths of the smallest iso-oriented hyperrectangle that contains S. As described in the proof of Theorem 4.22, the union of the pieces of these functions has a cardinality of $O(\lambda(n,k))$, and this step may be performed in $O[\lambda(\lambda(\frac{n}{p},k),k+1)\log n \ + \ T_{sort}(\lambda(n,k),p)]$ time on a $CGM_{env}(n,k,p)$.

2. Note $E(t) \ = \ \max\{D_1(t),\ldots,D_d(t)\}$. Therefore, a description of $E(t)$ may now be computed by $\lceil \log d \rceil \ = \ \Theta(1)$ merge-like steps, starting at the lowest level with the pieces of pairs of $\{D_1,\ldots,D_d\}$. This step may be performed in $T_{sort}(\lambda(n,k),p)$ time.

The assertion follows. ■

4.4.4 Vertex of Convex Hull

The *convex hull* of a set of points $S = \{P_0,\ldots,P_n\}$, denoted $hull(S)$, is the smallest convex set containing S. A point $P_i \in S$ is an *extreme point* or *vertex* of $hull(S)$ if $P_i \notin hull(S \setminus \{P_i\})$. In this section, we develop a CGM algorithm for determining when a given point $P_i \in S$ is an extreme point of $hull(S)$, where S is a set of point objects in $k-$motion in R^2. In doing so, we use some results of [Atal85a, B&M89a].

Let $T_{ij}(t)$ be the angle made by rotating the positively oriented horizontal ray with endpoint P_i about P_i until the ray contains the line segment from P_i to P_j at time t. By convention, $-\pi < T_{ij}(t) \leq \pi$. Formally, if $x_i(t), x_j(t), y_i(t)$, and $y_j(t)$ are the x and y coordinates of the points P_i and P_j, respectively, at time t, then

$$T_{ij}(t) = \begin{cases} \pi/2 & \text{if } x_i(t) = x_j(t) \text{ and } y_i(t) < y_j(t); \\ -\pi/2 & \text{if } x_i(t) = x_j(t) \text{ and } y_i(t) > y_j(t); \\ \arctan\left(\frac{y_j(t)-y_i(t)}{x_j(t)-x_i(t)}\right) & \text{if } x_i(t) < x_j(t); \\ \arctan\left(\frac{y_j(t)-y_i(t)}{x_j(t)-x_i(t)}\right) + \pi & \text{if } x_i(t) > x_j(t) \text{ and } y_i(t) < y_j(t); \\ \arctan\left(\frac{y_j(t)-y_i(t)}{x_j(t)-x_i(t)}\right) - \pi & \text{if } x_i(t) > x_j(t) \text{ and } y_i(t) > y_j(t); \\ \text{undefined} & \text{if } x_i(t) = x_j(t) \text{ and } y_i(t) = y_j(t). \end{cases}$$

$$\text{Define } G_{ij}(t) = \begin{cases} T_{ij}(t) & \text{if } T_{ij}(t) \geq 0; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\text{Define } B_{ij}(t) = \begin{cases} T_{ij}(t) & \text{if } T_{ij}(t) < 0; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Define the functions a_i, b_i, c_i, and d_i as follows.

$$a_i(t) = \min\{G_{ij}(t) \mid 0 \leq j < n,\ i \neq j,\ G_{ij}(t) \text{ is defined}\}.$$

$$b_i(t) = \max\{G_{ij}(t) \mid 0 \leq j < n,\ i \neq j,\ G_{ij}(t) \text{ is defined}\}.$$

$$c_i(t) = \min\{B_{ij}(t) \mid 0 \leq j < n,\ i \neq j,\ B_{ij}(t) \text{ is defined}\}.$$

$$d_i(t) = \max\{B_{ij}(t) \mid 0 \leq j < n,\ i \neq j,\ B_{ij}(t) \text{ is defined}\}.$$

If at time t, $G_{ij}(t)$ is undefined (respectively, $B_{ij}(t)$ is undefined) for all j, then $a_i(t)$ and $b_i(t)$ (respectively, $c_i(t)$ and $d_i(t)$) are undefined.

Lemma 4.24 [B&M89a], proof of Lemma 4.4: *Let S be a set of n point-objects that are in $k-$motion, for some positive integer k, and let $H : R^1 \to R^1$ be any of the functions in*

$$\{G_{ij},\ H_{ij} \mid 0 \leq i < n,\ 0 \leq j < n,\ i \neq j\}$$

described above. Let q be the number of jump discontinuities of H and let r be the number of transitions of H. Then $q + r \leq k$. ■

$$\text{Define } T = \{T_{ij} \mid j \neq i,\ 0 \leq j < n\}.$$

Lemma 4.25 [Atal85a, B&M89a] *For a system of n point-objects in the Euclidean plane with k-motion, each of the functions a_i, b_i, c_i, and d_i has at most $\lambda(n, 4k)$ pieces generated by T.* ■

Lemma 4.26 [Atal85a] *Given a set S of n point-objects in the plane with $k-$motion, a point P_i is an extreme point of hull(S) at time t if and only if*

1. $a_i(t) - d_i(t) > \pi$, *or*

2. $b_i(t) - c_i(t) < \pi$, *or*

3. $a_i(t)$ *and* $b_i(t)$ *are undefined, or*

4. $c_i(t)$ *and* $d_i(t)$ *are undefined.* ■

The reader may find Figure 1 helpful in understanding Lemma 4.26.

In the following theorem, we give an algorithm to determine the intervals of time over which a given point $P_i \in S = \{P_0, \ldots, P_{n-1}\}$ is an extreme point of *hull*(S). We will again assume that the roots of a polynomial of bounded degree can be determined in $\Theta(1)$ time.

Theorem 4.27 *Let $S = \{P_0, \ldots, P_{n-1}\}$ be a set of points in the plane with k-motion. Then the ordered intervals of time during which a given point P_i is an extreme point of hull(S) can be determined in $O[\lambda(\lambda(\frac{n}{p}, 4k), 4k+1)\log n + \ T_{sort}(\lambda(n, 4k), p)]$ time on a $CGM_{env}(n, 4k, p)$.*

Proof: Observe that solving $T_{ij}(t) = T_{im}(t)$ means finding instants at which the directed line segment from P_i to P_j and the directed line segment from P_i to P_m are parallel and similarly oriented. Finding instants when the line segments are parallel requires solving the equation

$$[y_j(t) - y_i(t)]\,[x_m(t) - x_i(t)] = [y_m(t) - y_i(t)]\,[x_j(t) - x_i(t)] \tag{5}$$

which is a polynomial equation of degree at most $2k$. We assume such equations can be solved in $\Theta(1)$ time by a single PE. Further, determining whether or not two parallel directed line segments are similarly oriented can be accomplished in $\Theta(1)$ serial time. It follows that $T_{ij}(t) = T_{im}(t)$ can be solved by a single processor in $\Theta(1)$ time.

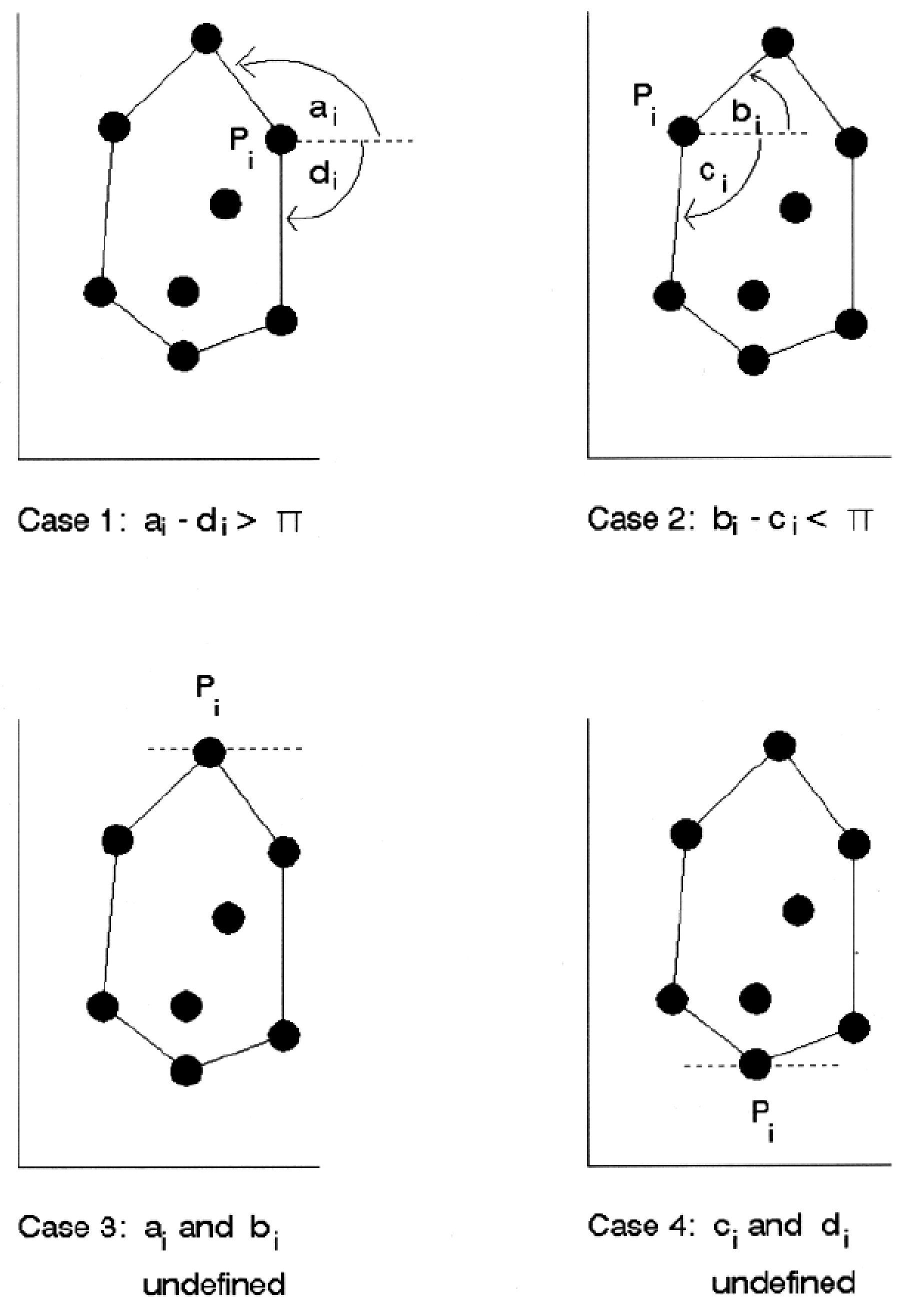

Figure 1: Extreme points of the convex hull.

By Lemma 4.24, each G_{ij} (similarly, each B_{ij}) has at most k values of t that yield jump discontinuities or transitions. It follows from Theorem 4.9 that we can construct the functions $a_i(t), b_i(t), c_i(t)$, and $d_i(t)$ in $T_{env}(n, 4k, p)$ time. It follows from Lemma 4.25 and Lemma 4.10 that each of $a_i(t) - d_i(t)$ and $b_i(t) - c_i(t)$ has $O(\lambda(n, 4k))$ pieces generated by differences of members of T. The ordered pieces of the functions $a_i(t) - d_i(t)$ and $b_i(t) - c_i(t)$ are now constructed in $T_{sort}(\lambda(n, 4k), p)$ time by merge-like operations. Similarly, ordered maximal intervals on which $a_i(t)$ and $b_i(t)$ are both undefined (respectively, on which $c_i(t)$ and $d_i(t)$ are both undefined) are determined in $T_{env}(n, 4k, p)$ time.

Define

$$A_i(t) = \begin{cases} 1 & \text{if } a_i(t) - d_i(t) \geq \pi \\ 0 & \text{otherwise} \end{cases}$$

and

$$B_i(t) = \begin{cases} 1 & \text{if } b_i(t) - c_i(t) \leq \pi \\ 0 & \text{otherwise.} \end{cases}$$

Observe that if I_1 and I_2 are intervals of pieces of a_i and d_i, respectively, where $I = I_1 \cap I_2$ is nondegenerate, then $a_i|_I(t) - d_i|_I(t) = \pi$ implies there are integers j and m determined by I_1 and I_2, respectively, such that $a_i|_I = T_{ij}$, $d_i|_I = T_{im}$, and $T_{ij}(t) - T_{im}(t) = \pi$. Solving the latter means finding instants at which the directed line segment from P_i to P_j and the directed line segment from P_i to P_m are parallel and oppositely oriented. We noted above that finding instants when the line segments are parallel may be accomplished in $\Theta(1)$ serial time, and that there are at most $2k$ such instants. Determining whether or not two parallel directed line segments are oppositely oriented may also be done in $\Theta(1)$ serial time. Every piece of $a_i(t) - d_i(t)$ generated by differences of members of T yields at most $2k + 1$ pieces of $A_i(t)$ generated by the set of constant functions $\{0, 1\}$. It follows from Lemma 4.11 that $A_i(t)$ has at most $(2k+1)\, 2\, \lambda(n, 4k) = O(\lambda(n, 4k))$ pieces generated by $\{0, 1\}$. Similarly, $B_i(t)$ has $O(\lambda(n, 4k))$ pieces generated by $\{0, 1\}$. The functions $A_i(t)$ and $B_i(t)$ may be constructed in $T_{sort}(\lambda(n, 4k), p)$ additional time, using merge-like operations.

Similarly, in $T_{sort}(\lambda(n, 4k), p)$ time we can construct the $O(\lambda(n, 4k))$ ordered pieces generated by $\{0, 1\}$ of

$$C_i(t) = \begin{cases} 1 & \text{if both } a_i(t) \text{ and } b_i(t) \text{ are undefined} \\ 0 & \text{otherwise} \end{cases}$$

and

$$D_i(t) = \begin{cases} 1 & \text{if both } c_i(t) \text{ and } d_i(t) \text{ are undefined} \\ 0 & \text{otherwise.} \end{cases}$$

It follows from Lemma 4.11 that there are $O(\lambda(n, 4k))$ pieces generated by $\{0, 1\}$ of

$$H_i(t) = \max\{A_i(t), B_i(t), C_i(t), D_i(t)\},$$

which now may be determined in $T_{sort}(\lambda(n, 4k), p)$ additional time by merge-like operations. Since Lemma 4.26 implies P_i is an extreme point at time t if and only if $H_i(t) = 1$, the algorithm is complete. The running time of the algorithm is $T_{env}(n, 4k, p) + T_{sort}(\lambda(n, 4k), p)$, which, by Lemma 4.2 and the definition of T_{env}, is $O[\lambda(\lambda(\frac{n}{p}, 4k), 4k+1)\log n + T_{sort}(\lambda(n, 4k), p)]$. ■

5 Maximal collinear sets

In this section, we give a scaleable parallel algorithm to solve the All Maximal Equally Spaced Collinear Subsets (AMESCS [K&R91]) Problem: Given a set S of n points in a Euclidean space, find all maximal equally-spaced collinear subsets of S determined by segments of any length ℓ. This problem was studied in [K&R91, B&M93]. The algorithm of [K&R91] runs in optimal $\Theta(n^2)$ serial time. It seems to be an essentially sequential algorithm. A rather different algorithm that is efficient on a fine-grained PRAM and optimal on a fine-grained mesh is presented in [B&M93].

We say $S' \subset S$ is *collinear* if $|S'| > 2$ and there is a line in R^d that contains all members of S'. A collinear set S' is *equally-spaced* if the members $\{s_1, \ldots, s_k\}$ of S' are in lexicographic order such that all of the line segments $\overline{s_i s_{i+1}}$ have the same length ℓ; such a set S' is a *maximal equally-spaced collinear subset determined by segments of length* ℓ if it is not properly contained in any other equally-spaced collinear subset determined by segments of length ℓ.

The AMESCS Problem is interesting because the regularity sought is often meaningful in a seemingly irregular environment. Collinear equally-spaced subsets might represent street lights, fence posts, land mines, etc. We have the following.

Theorem 5.1 *Let d be a fixed positive integer. Let $S \subset R^d$, $|S| = n$. Then the AMESCS Problem can be solved for S in $T_{sort}(n^2, p)$ time on a $CGM(n^2, p)$.*

Proof: We give the following algorithm.

1. Sort the members of S according to lexicographic order. This takes $T_{sort}(n, p)$ time.

2. Determine the set L of all the ordered pairs of distinct data points in S such that the first member of the pair precedes the second. This may be done by the algorithm of Proposition 2.8 in $O(T_{sort}(n^2, p))$ time.

   Since S was sorted, the ordered pair formed from the set $\{x_i, x_j\}$, $i < j$, is (x_i, x_j).

3. Sort the members (x_i, x_j) of L with respect to length as the primary key and lexicographic order of x_i and x_j as secondary and tertiary keys, respectively. This takes $T_{sort}(n^2, p)$ time.

4. In parallel, every processor determines for each of its ordered pairs $(x_i, x_j) \in L$ a third point $z_{(i,j)}$ such that $(x_i, x_j, z_{(i,j)})$ is an equally spaced collinear triple with the $x_i < x_j < z_{(i,j)}$. This is done in $\Theta(\frac{n^2}{p})$ time.

5. Perform a parallel search to determine for each pair (x_i, x_j) whether $z_{(i,j)} \in S$. If so, note the value of k such that $z_{(i,j)} = x_k$. This takes $T_{sort}(n^2, p)$ time.

6. For each $(x_i, x_j) \in L$, create a record $L_{i,j} = (x_i, x_j, i, j, k, i, j)$, where k is as determined in the previous step, if found; otherwise, $k = \infty$. This takes $\Theta(\frac{n^2}{p})$ time.

7. Now we perform a component labeling-like step. The ordering of L above allows the records $L_{i,j}$ to inherit the order of L such that

   - members of $\{L_{i,j} \mid 1 \leq i < j \leq n\}$ of the same length are consecutive, and
   - if $x_k = z_{(i,j)}$, then $L_{i,j} < L_{j,k}$.

   Let $M = \{m_s \mid s = 1, \ldots, N\}$ be an enumeration of the members of $\{L_{i,j} \mid 1 \leq i < j \leq n\}$, $m_i < m_{i+1}$, where $N = \Theta(n^2)$. Regard the third and fourth components of each $L_{i,j}$ record as representing the indices of a line segment's endpoints; the fifth component, if finite, as indicating the next vertex in a graph's component; and the sixth and seventh components as forming a component label. We now perform a parallel prefix operation, in $\Theta(\frac{n^2}{p}) + T_{sort}(p^2, p)$ time, to compute all of the members of

   $$\{m_1, m_1 \circ m_2, \ldots, m_1 \circ m_2 \circ \ldots \circ m_N\},$$

   where $u \circ v$ is defined as follows.

- Suppose $u = (x_i, x_j, i, j, k, a, b)$ and $v = (x_j, x_k, j, k, l, c, d)$. Then

$$u \circ v = (x_j, x_k, j, k, l, a, b).$$

- Otherwise, $u \circ v = v$.

8. At the end of the last step, the prefixes m_i that are identical in the last two components represent maximal equally spaced collinear subsets of S. Now, sort the m_i with respect to, in decreasing priority, the sixth, seventh, and third components of the m_i records, so that all members of a maximal equally spaced collinear set are grouped consecutively (sixth and seventh components), and, within maximally equally spaced collinear sets, the points are ordered (third components). This takes $T_{sort}(n^2, p)$ time.

The running time of the algorithm is $T_{sort}(n^2, p)$. ■

6 Point set pattern matching

In this section, we give scaleable parallel algorithms to solve the Point Set Pattern Matching (PSPM) Problem: Given a set S of points in a Euclidean space R^d and a pattern $P \subset R^d$, find all subsets $P' \subset S$ such that P and P' are congruent. Serial and fine-grained parallel solutions to this problem have been given in several papers, including [Boxe92, Boxe96, dR&L95, G&K92, L&L92, SL&Y90].

We assume that $|S| = n$, $|P| = k \leq n$, and that the coordinates of all members of $P = \{a_0, a_1, \ldots, a_{k-1}\}$ and $S = \{s_0, s_1, \ldots, s_{n-1}\}$ are given as input to the problem, with each of P and S evenly distributed among the processors of a CGM. In the following, we give rather different algorithms for solving the Point Set Pattern Matching Problem for different values of d, the dimension of the ambient Euclidean space. Roughly, this is because different dimensions produce different constraints on the complexity of the output. We also give algorithms for PSPM restricted to realization via rotation or translation in R^2.

6.1 PSPM in R^1

A serial algorithm for this case is given in [dR&L95], in which it is shown that the worst case output complexity is $\Theta(k(n-k))$. We have the following.

Theorem 6.1 *The Point Set Pattern Matching Problem in R^1 can be solved on a $CGM(k(n-k),p)$ in optimal $T_{sort}(k(n-k),p)$ time.*

Proof: We give the following algorithm.

1. Sort the members of S by their coordinates in $T_{sort}(n,p)$ time.

2. Sort the members of P by their coordinates in $T_{sort}(k,p)$ time.

3. Broadcast a_0 to all processors. This takes $O(p)$ time.

4. For $j \in \{0,1,\ldots,n-k-1\}$, compute $d_j = s_j - a_0$. This takes $\Theta(\frac{n-k}{p})$ time.

5. For $i \in \{0,1,\ldots,k-1\}$, $j \in \{0,1,\ldots,n-k-1\}$, define $A_{i,j}$ to be true if and only if $(a_i+d_j) \in S$. If $A_{i,j}$ is true, associate the index $m(i,j)$ with $A_{i,j}$, where $s_{m(i,j)} = a_i + d_j$. These values can be computed by a parallel search operation in $T_{sort}(k(n-k),p)$ time.

6. In $T_{sort}(k(n-k),p)$ time, sort the $A_{i,j}$ with respect to j as primary key and i as secondary key.

7. Observe now that P is matched in S via a translation that sends a_0 to s_j if and only if for all i, $A_{i,j}$ is true. In $\Theta(\frac{k(n-k)}{p}) + T_{sort}(p^2,p)$ time, perform a parallel prefix operation on the $A_{i,j}$ to determine which indices j yield such translations. Let L_q be the q^{th} index j such that a translation τ of P sending a_0 to s_j satisfies $\tau(P) \subset S$. We note the members of S forming the set that matches P via this translation are marked by the indices associated with the $A_{i,j}$ above.

8. Another $\Theta(\frac{k(n-k)}{p}) + T_{sort}(p^2,p)$ time parallel prefix operation can be used to produce a list of indices $M_{q,r}$ from the lists $A_{i,j}$ and L_q such that $M_{q,0} = L_q$ and $M_{q,r} = s_{m(r,L_q)}$, the index of the member of S to which a_r is translated, for $1 \leq r \leq k-1$. Thus, the list M is an ordered list of the indices of translated copies of P in S.

9. The steps above find all matches of P in S obtained by translating P. In order to find matches obtained by reflecting and translating P, we compute the set $-P = \{-p \mid p \in P\}$ and repeat the previous steps with $-P$ substituted for P. This takes $T_{sort}(k(n-k),p)$ time.

10. It may happen that the same subset of S is found more than once as a match for P. We may eliminate such duplication as follows.

    - Sort each of the lists
    $$\mathcal{M}_q \;=\; \{M_{q,r} \mid r = 0, \ldots, k-1\}.$$
    Then sort the list of lists
    $$\mathcal{M}_1, \mathcal{M}_2, \ldots$$
    by lexicographic order. This takes $T_{sort}(k(n-k), p)$ time.

    - Use a parallel prefix operation to remove any $\mathcal{M}_q$ that equals its predecessor. This takes $\Theta(\frac{k(n-k)}{p}) \;+\; T_{sort}(p^2, p)$ time.

Thus, the algorithm takes $T_{sort}(k(n-k), p)$ time. This is optimal, as we produce ordered lists with, in the worst case, $\Theta(k(n-k))$ members. ■

6.2 PSPM in R^2

Let $b > 0$ be a fixed constant. In the Euclidean plane R^2, the complexity of the output in the Point Set Pattern Matching Problem is, in part, limited by the complexity of the function $D_2(n)$, the number of line segments in R^2 of length b whose endpoints are in $S \subset R^2$. The function $D_2(n)$ was introduced in [Erd46].

Proposition 6.2 [SST84] $D_2(n) \;=\; O(n^{4/3})$. ■

We have the following, which is implicit in [G&K92].

Proposition 6.3 *The output of the Point Set Pattern Matching Problem in R^2 has complexity $O(kD_2(n))$.*

Proof: Let b be the length of the line segment from a_0 to a_1. There are at most $D_2(n)$ line segments $\ell \subset R^2$ of this length with endpoints in S. For each such ℓ, let the endpoints of ℓ be $\{s_{i_0}, s_{i_1}\} \subset S$. A necessary condition for the existence of $\{s_{i_2}, \ldots, s_{i_{k-1}}\} \subset S$ such that $\{s_{i_0}, s_{i_1}, s_{i_2}, \ldots, s_{i_{k-1}}\}$ is a match for P is the existence of i_2 such that $\{s_{i_0}, s_{i_1}, s_{i_2}\}$ is a match for

$\{a_0, a_1, a_2\}$. There are at most 2 such values of i_2, each of which determines at most 1 matching of P in S. Since every matching has complexity k, the assertion follows. ■

The sequential time necessary to find all the $O(D_2(n))$ line segments of length b with endpoints in S is denoted by $A_2(n)$. We have the following.

Proposition 6.4 [Agar90, Chaz91] *For any fixed $\delta > 0$, $A_2(n) = O(n^{\frac{4}{3}+\delta})$.* ■

Theorem 6.5 [G&K92] *The Point Set Pattern Matching Problem in R^2 can be solved sequentially in $O(A_2(n) + kD_2(n)\log n)$ time.* ■

Theorem 6.6 *The Point Set Pattern Matching Problem in R^2 can be solved in*

$$\frac{A_2(n)}{p} + pT_{sort}(n,p) + T_{sort}(kD_2(n),p) = \frac{T_{seq}}{p} + O(pT_{sort}(n,p) + T_{sort}(kD_2(n),p))$$

time on a $CGM(kD_2(n),p)$. For $p = O(\frac{kD_2(n)}{n})$, the running time is

$$\frac{A_2(n)}{p} + T_{sort}(kD_2(n),p) = O(\frac{T_{seq}}{p} + T_{sort}(kD_2(n),p)).$$

Proof: Note it follows from Theorem 6.5 that $\frac{A_2(n)}{p} = O(\frac{T_{seq}}{p})$. We give the following algorithm.

1. Broadcast $\{a_0, a_1, a_2\}$ to all processors and determine, in each processor, $b = d(a_0, a_1)$, where d is the Euclidean distance function. This takes $O(p)$ time.

2. Determine all the combinations $\{s_i, s_j\} \subset S$ such that $d(s_i, s_j) = b$. This is done as follows.

   - In parallel, each processor P_i determines all of its pairs of members of S that are at distance b from each other. Let S_i be the subset of S contained in P_i.
   - Perform $p-1$ circular rotations of S, keeping in processor P_i a copy of S_i. After each rotation operation, P_i has copies of S_i and S_j for some $j \neq i$. Processor P_i finds all combinations $\{s_q, s_r\}$, $s_q \in S_i$, $s_r \in S_j$, such that $d(s_q, s_r) = b$.

   These operations take

$$\frac{A_2(n)}{p} + (p-1)T_{sort}(n,p) = \frac{T_{seq}}{p} + O(pT_{sort}(n,p))$$

   time.

3. For each of the $O(D_2(n))$ pairs $\{s_i, s_j\}$ of members of S that are at distance b from each other, determine the two points $z_m(i,j)$, $m \in \{0,1\}$, such that (a_0, a_1, a_2) matches $(s_i, s_j, z_m(i,j))$. This takes $O(\frac{D_2(n)}{p})$ time.

4. For each of the $O(D_2(n))$ pairs $\{s_i, s_j\}$ of members of S that are at distance b from each other, determine for $m \in \{0,1\}$ whether $z_m(i,j) \in S$. This may be done via a parallel search operation in $T_{sort}(D_2(n), p)$ time.

5. For each of the $O(D_2(n))$ triples $(s_{i_0}, s_{i_1}, s_{i_2})$ such that $(s_{i_0}, s_{i_1}, s_{i_2})$ matches (a_0, a_1, a_2), determine whether there exist $s_{i_3}, \ldots, s_{i_{k-1}}$ in S such that $(s_{i_0}, s_{i_1}, s_{i_2}, s_{i_3}, \ldots, s_{i_{k-1}})$ matches P. This is done as follows.

   - For each such triple $(s_{i_0}, s_{i_1}, s_{i_2})$ and each $j \in \{3, 4, \ldots, k-1\}$, determine the unique $z_j \in R^2$ such that $(s_{i_0}, s_{i_1}, s_{i_2}, z_j)$ matches (a_0, a_1, a_2, a_j). This takes $O(\frac{kD_2(n)}{p})$ time.
   - For each such z_j, determine whether $z_j \in S$. If so, let the j^{th} component of a $k-$tuple, whose components with indices $0, 1, 2$ are, respectively, $s_{i_0}, s_{i_1}, s_{i_2}$, be z_j; otherwise, let the j^{th} component of this $k-$tuple be $fail$. This may be done a via parallel search operation in $T_{sort}(kD_2(n), p)$ time.
   - Perform a parallel prefix operation to remove those $k-$tuples constructed above that have at least one $fail$ entry. The remaining $k-$tuples represent all the matches of P in S. This step requires $\Theta(\frac{kD_2(n)}{p}) \ + \ T_{sort}(p^2, p)$ time.

6. It may happen that the same subset of S is found more than once as a match for P. We may eliminate such duplication as follows.

   - Sort each of the $k-$tuples representing a match of P in S lexicographically. Then sort the collection of such $k-$tuples by lexicographic order. This takes $T_{sort}(kD_2(n), p)$ time.
   - Use a parallel prefix operation to remove any $k-$tuple that equals its predecessor. This takes $\Theta(\frac{kD_2(n)}{p}) \ + \ T_{sort}(p^2, p)$ time.

Thus, the algorithm requires

$$\frac{A_2(n)}{p} \ + \ pT_{sort}(n,p) \ + \ T_{sort}(kD_2(n),p) \ = \frac{T_{seq}}{p} \ + \ O(pT_{sort}(n,p) \ + \ T_{sort}(kD_2(n),p))$$

time. It follows from Lemma 2.1 that $pT_{sort}(n,p) = O(T_{sort}(np,p))$, so for $p = O(\frac{kD_2(n)}{n})$, hence for $np = O(kD_2(n))$, the running time reduces to

$$\frac{A_2(n)}{p} + T_{sort}(kD_2(n),p) = O(\frac{T_{seq}}{p} + T_{sort}(kD_2(n),p)).$$

■

6.3 PSPM in R^3

In this section, we present a scaleable parallel algorithm for solving the Point Set Pattern Matching Problem in R^3.

In the following, the function λ is the same function used earlier to discuss the complexity of lower envelope functions. The following is used to construct an upper bound on the complexity of the output.

Proposition 6.7 [CEGSW90] *Let $S \subset R^3$ with $|S| = n$. The maximum number of line segments in R^3 of a given length with endpoints in S is $n^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}$.* ■

It follows from Theorem 4.3 that the expression $[\frac{\lambda(n,6)}{n}]^{1/4}$, which appears in the analysis of our algorithm, is nearly constant. We have the following, which is implicit in [Boxe96].

Proposition 6.8 *The output of the Point Set Pattern Matching Problem in R^3 has complexity*

- $O(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4})$*, if P is a collinear set;*
- $O(kn^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4})$ *in general.*

Proof: By Proposition 6.7, there are $O(n^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4})$ pairs (s_{i_0}, s_{i_1}) of members of S such that (s_{i_0}, s_{i_1}) matches (a_0, a_1).

- If P is a collinear set, then, without loss of generality, P is contained in the line segment from a_0 to a_1. A necessary condition of a matching of P by members of S is a matching of a pair of members (s_{i_0}, s_{i_1}) of S with (a_0, a_1). Since P is a collinear set, there are at most two subsets $\{s_{i_2}, \ldots, s_{i_{k-1}}\}$ of S in the line segment from s_{i_0} to s_{i_1} that match $\{a_2, \ldots, a_{k-1}\}$ and either (s_{i_0}, s_{i_2}) or (s_{i_1}, s_{i_2}) matches (a_0, a_2). It follows that the output in this case has complexity $O(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4})$.

- If P is not collinear, we may assume a_0, a_1, and a_2 are not collinear. For every pair (s_{i_0}, s_{i_1}) that matches (a_0, a_1), there are at most $n-2$ points $s_{i_2} \in S$ such that $(s_{i_0}, s_{i_1}, s_{i_2})$ matches (a_0, a_1, a_2). For each of the $O(n^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4})$ such triples $(s_{i_0}, s_{i_1}, s_{i_2})$, there is a unique rigid transformation $f_{i_0 i_1 i_2}$ of R^3 such that $f_{i_0 i_1 i_2}(a_j) = s_{i_j}$, $j \in \{0,1,2\}$. The transformation $f_{i_0 i_1 i_2}$ yields a matching of P in S if and only if $f_{i_0 i_1 i_2}(\{a_j \mid j = 3, \ldots, k-1\}) \subset S$. Since there is a one-to-one correspondence between such rigid transformations of R^3 and matchings of P in S, and every such matching has complexity $\Theta(k)$, it follows that the output in the general case has complexity $O(kn^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4})$. ■

Theorem 6.9 [Boxe96] *The Point Set Pattern Matching Problem in R^3 can be solved on a serial computer in*

- $O(n^2 + kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4} \log n)$ *time, if P is a collinear set;*
- $O(kn^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4} \log n)$ *time in the general case.* ■

We present a scaleable parallel algorithm to handle the special case of a collinear pattern.

Proposition 6.10 *Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Suppose it is known that there is a line $L \subset R^3$ such that $P \subset L$. Then every subset P' of S such that P' matches P can be identified in*

$$O[T_{sort}(n^2, p) + T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)]$$

time on a $CGM(n^2 + kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)$.

Proof: We give the following algorithm.

1. Sort P by lexicographic order. This takes $T_{sort}(k, p)$ time.
2. Sort S by lexicographic order. This takes $T_{sort}(n, p)$ time.
3. Form the set C of all the ordered pairs (s_i, s_j), $i \neq j$, of members of S, and sort C with respect to the lengths of the line segments whose endpoints are the members of the respective pairs. By Proposition 2.8, this takes $O(T_{sort}(n^2, p))$ time. Note $|C| = \Theta(n^2)$.

4. Broadcast $\{a_0, a_{k-1}\}$ to all processors. This takes $O(p)$ time.

5. Use a parallel prefix operation to determine the range C' of consecutive members of the ordered list C representing line segments whose length equals the length of the line segment from a_0 to a_{k-1}. This takes $\Theta(\frac{n^2}{p}) + T_{sort}(p^2, p)$ time. By Proposition 6.7, $|C'| = O(n^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4})$.

6. The members of C' are not (likely) evenly distributed. Use a sort step to redistribute the members of C' evenly among the processors. This takes $O(T_{sort}(n^2, p))$ time.

7. We know (s_i, s_j) matches (a_0, a_{k-1}) for each $(s_i, s_j) \in C'$. Thus, to identify a subset of S that matches P including a submatch of (s_i, s_j) with (a_0, a_{k-1}), it is necessary and sufficient to determine if there exists a $(k-2)-$tuple $(s_{i_1}, \ldots, s_{i_{k-2}})$ such that for each i_m, $m \in \{1, \ldots, k-2\}$,

   - $s_{i_m} \in S$;
   - s_{i_m} belongs to the line segment $\overline{s_i s_j}$; and
   - the length of $\overline{s_i s_{i_m}}$ equals the length of $\overline{a_0 a_m}$ for all $m \in \{1, 2, \ldots, k-2\}$, or the length of $\overline{s_i s_{i_m}}$ equals the length of $\overline{a_{k-1} a_m}$ for all $m \in \{1, 2, \ldots, k-2\}$.

   For each m, it takes $\Theta(1)$ time to compute the length of $\overline{a_0 a_m}$ and the Cartesian coordinates of s_{i_m}; we also require time for a search on S to determine if $s_{i_m} \in S$. Since these searches can be done in a parallel search operation, determination of all such matches of P in S takes $T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)$ time. The list of matches of P in S has complexity $O(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4})$. Repeat this step, substituting $\overline{a_{k-1} a_m}$ for $\overline{a_0 a_m}$.

8. It may happen that the same subset of S appears twice in our list M of matches of P. If we wish to eliminate such duplication, we may do so as follows.

   - Sort all the $k-$tuples in M lexicographically. This takes $T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)$ time.
   - Now sort M lexicographically. This takes $T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)$ time.
   - Perform a parallel prefix operation to remove every entry of the ordered list M that equals its predecessor. This takes $\Theta(\frac{kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}}{p}) + T_{sort}(p^2, p)$ time.

Thus, our algorithm takes $O[T_{sort}(n^2, p) \ + \ T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)]$ time. ■

Our algorithm for the general Point Set Pattern Matching Problem in R^3 is given below.

Theorem 6.11 *Let P and S be finite subsets of R^3. Let $|P| = k \leq n = |S|$. Then every subset P' of S such that P' is congruent to P can be identified on a $CGM(kn^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)$, in*

- *$O[T_{sort}(n^2, p) \ + \ T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4})]$ time, if P is a collinear set;*
- *$T_{sort}(kn^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)$ time in the general case.*

Proof: Without loss of generality, $i \neq j$ implies $a_i \neq a_j$. We give the following algorithm.

1. Determine whether or not P is a collinear set. This is done as follows. Broadcast a_0 and a_1 to all processors in $O(p)$ time. Then, for each $k \in \{2, \ldots, k-1\}$, determine if a_k is collinear with a_0 and a_1, in $\Theta(\frac{k}{p})$ time. P is a collinear set if and only if a_k is collinear with a_0 and a_1 for all $k \in \{2, \ldots, k-1\}$. If P is not a collinear set, note an index r such that a_0, a_1, and a_r are not collinear. This may be done, *e.g.*, by a minimum (with respect to indices) operation on $P \setminus \{a_0, a_1\}$ in $\Theta(\frac{k}{p}) \ + \ T_{sort}(p^2, p)$ time, followed by $O(p)$ time broadcasts of a_0, a_1, and a_r to all processors.

2. If P is a collinear set, execute the algorithm of Proposition 6.10. This finishes the current algorithm in an additional $O[T_{sort}(n^2, p) + T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p)]$ time. Otherwise, continue with the following steps.

3. Sort S lexicographically. This takes $T_{sort}(n, p)$ time.

4. For every pair s_i, s_j, $i < j$, of distinct members of S, form the line segment (s_i, s_j). Let $L(S)$ be the set of such line segments. By Proposition 2.8, this step takes $O(T_{sort}(n^2, p))$ time.

5. Form the set $L(P) \ = \ \{\pi_i\}_{i=1}^{k-1}$, where $\pi_i \ = \ \overline{a_0 a_i}$ is the line segment from a_0 to a_i. Since every processor has the value of a_0, this takes $\Theta(\frac{k}{p})$ time.

6. Sort the set $L(S)$, using the lengths of the members as the primary key and lexicographic order on the coordinates of the endpoints as the secondary key. This takes $T_{sort}(n^2, p)$ time.

7. Let M be the number of members of $L(S)$ whose length is equal to the length of π_1. Mark the sublist of $L(S)$ whose members' length equals the length of π_1 and determine the value of M by performing a parallel prefix operation on $L(S)$. The time required is $\Theta(\frac{n^2}{p}) + T_{sort}(p^2, p)$. If $M = 0$, the length of π_1 is not matched by that of a member of $L(S)$, so report failure and halt. Otherwise, let L^1 be the sublist of $L(S)$ whose members have length equal to the length of π_1. Note by Proposition 6.7 that
$$M \leq n^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4}.$$

8. As above, mark L^r, the sublist of $L(S)$ whose entries have length equal to the length of π_r. This is done via a parallel prefix operation on $L(S)$ in $\Theta(\frac{n^2}{p}) + T_{sort}(p^2, p)$ time.

9. For each $S_{ij} = (s_i, s_j) \in L^1$, find all $S_{jm} = (s_j, s_m) \in L^r$ such that $S_{ij} \cup S_{jm}$ matches $\pi_1 \cup \pi_r$. This may be done by a search on L^r to find the subrange of its members that have s_j as initial endpoint, then testing each member S_{jm} of the subrange for the match. Since there are M members of L^1, each of which requires a search to determine a subrange of L^r containing suitable candidates S_{jm}, the searches may be performed by a parallel search operation in $O(T_{sort}(M + n^2, p)) = O(T_{sort}(n^2, p))$ time. Since for each $S_{ij} \in L^1$ there are $O(n)$ suitable values of $S_{jm} \in L^r$ (since $O(n)$ members of L^r share an endpoint with S_{ij}), such pairs (S_{ij}, S_{jm}) may be formed by circular rotations of L^r accompanied by the formation of pairs in $O(pT_{sort}(M, p) + \frac{Mn}{p})$ time. By Lemma 2.1, this is
$$O(T_{sort}(Mp, p) + T_{sort}(Mn, p)) = O(T_{sort}(Mn, p))$$
time to form the $O(Mn) = O(n^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4})$ such pairs (S_{ij}, S_{jk}). Note each pair (S_{ij}, S_{jk}) corresponds to a triple (s_i, s_j, s_m) of vertices in S that match (a_0, a_1, a_r).

10. Since a_0, a_1, and a_r are not collinear, for each triple (s_i, s_j, s_m) of vertices in S that matches (a_0, a_1, a_r) we can describe in $\Theta(1)$ time the unique rigid transformation f_{ijm} of R^3 such that $f_{ijm}(a_0) = s_i$, $f_{ijm}(a_1) = s_j$, and $f_{ijm}(a_r) = s_m$. Since there are $O(Mn)$ such triples, creating all such descriptions takes $O(\frac{Mn}{p}) = O(n^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4}/p) = O(T_{sort}(n^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4}, p))$ time.

11. If $k > 3$, proceed as follows. For each of the $O(Mn)$ rigid transformations f_{ijm} of R^3 determined above, compute the set

$$V_{ijm} \;=\; \{f_{ijm}(a_q) \mid 2 \leq q \leq k-1,\ q \neq r\}$$

and, for each of its members, determine via a search of S which, if any, member of S it equals. These operations can be done by circular rotations of P, computation of all the sets V_{ijm}, and a parallel search operation. Altogether, these operations require, respectively, $(p-1)T_{sort}(k,p)$ $= O(T_{sort}(kp,p))$, $\Theta(\frac{kMn}{p})$, and $T_{sort}(kMn,p)$ time. Thus, the operations required for this step take $T_{sort}(kn^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4},p)$ time. If $V_{ijm} \subset S$, then $f_{ijm}(P) \subset S$.

12. Among the sets $f_{ijm}(P)$ that match P, there may be duplicate sets determined by distinct f_{ijm}. If desired, we may eliminate such duplication as follows.

    - Sort each of the $f_{ijm}(P)$ by lexicographic order. This takes $T_{sort}(kn^{5/2}[\lambda(n,6)/n]^{1/4},p)$ time.
    - Now sort the collection of matches $f_{ijm}(P)$ of P in S by lexicographic order. This takes $T_{sort}(kn^{5/2}[\lambda(n,6)/n]^{1/4},p)$ time.
    - Perform a parallel prefix operation to eliminate each member of the list that is equal to a predecessor on the list. This takes $\Theta(\frac{kn^{5/2}[\lambda(n,6)/n]^{1/4}}{p}) \;+\; T_{sort}(p^2,p)$ time.

The algorithm requires

- $T_{sort}(kn^{3/2}[\frac{\lambda(n,6)}{n}]^{1/4},p)$ time if P is a collinear set;
- $T_{sort}(kn^{5/2}[\frac{\lambda(n,6)}{n}]^{1/4},p)$ time in the general case. ■

6.4 PSPM in R^2 under rotations or translations

In this section, we give scaleable parallel algorithms for the PSPM Problem in R^2 under the restrictions that the pattern matching be realized via a rotation or a translation of P. As above, we assume the pattern set P has cardinality k, the sampling set S has cardinality n, and that $0 < k \leq n$.

We have the following.

Theorem 6.12 [G&K92]

- *Every rotation r of P about the origin such that $r(P) \subset S$ may be found in $O(kn + n\log n)$ serial time.*
- *Every translation T of P in R^2 such that $T(P) \subset S$ may be found in $O(kn + n\log n)$ serial time.* ■

We give a scaleable parallel version of Theorem 6.12.

Theorem 6.13
- *Every rotation r of P about the origin such that $r(P) \subset S$ may be found in $O(T_{sort}(kn,p))$ time on a $CGM(kn,p)$.*
- *Every translation T of P in R^2 such that $T(P) \subset S$ may be found in $O(T_{sort}(kn,p))$ time on a $CGM(kn,p)$.*

Proof: Let $\mathcal{R}$ be the set of angles θ, $0 \leq \theta < 2\pi$, such that a rotation r_θ of P by θ about the origin satisfies $r_\theta(P) \subset S$. For each $a \in P$, let $\mathcal{R}(a)$ be the set of angles θ, $0 \leq \theta < 2\pi$, such that a rotation r_θ of p by θ about the origin satisfies $r_\theta(a) \in S$. Note that

$$\mathcal{R} = \cap_{a\in P} \mathcal{R}(a)$$

and that, in the worst case, $|\mathcal{R}(a)| = n$ for all $a \in P$ (this happens if $P \cup S$ is contained in a circle centered at the origin). We give the following algorithm.

1. Sort S by distance from the origin as the primary key and angular coordinate as the secondary key. This takes $T_{sort}(n,p)$ time.
2. For all $a \in P$, compute $\mathcal{R}(a)$ by forming $O(kn)$ pairs (a,θ), $a \in P$, θ an angle by which a may be rotated into $s \in S$ such that a and s have the same distance from the origin. This may be done in $O(T_{sort}(kn,p))$ time, as follows.
   - Form $P \times S$ by the algorithm of Proposition 2.9 in $O(T_{sort}(kn,p))$ time.
   - In $\Theta(\frac{kn}{p})$ time, each processor examines each of its pairs $(a,s) \in P \times S$ and, if a and s have the same distance from the origin, forms the corresponding pair (a,θ).

3. Sort $\cup_{i=1}^{k} \mathcal{R}(a_i)$ with respect to the angular coordinate. This takes $O(T_{sort}(kn, p))$ time.

4. Note that $\theta \in \mathcal{R}$ if and only if θ appears as the angular component of k consecutive entries of the ordered list $\cup_{i=1}^{k} \mathcal{R}(a_i)$. Thus, the set $\mathcal{R}$ may be computed from a parallel prefix operation on $\cup_{i=1}^{k} \mathcal{R}(a_i)$ in $O(\frac{kn}{p}) \; + \; T_{sort}(p^2, p)$ time.

The algorithm to compute $\mathcal{R}$ thus takes $O(T_{sort}(kn, p))$ time.

A similar algorithm is used to find the set of all translations T of P in R^2 such that $T(P) \subset S$ in $O(T_{sort}(kn, p))$ time. The most important modifications to the algorithm above are the following.

- In the first step, S is sorted by lexicographical order.
- Replace the second step as follows. Define $\mathcal{R}(a)$ by

  $$\mathcal{R}(a) \;=\; \{s - a \mid s \in S\}.$$

  The sets $\mathcal{R}(a)$ can all be computed after forming all pairs (a, s), where $a \in P$, $s \in S$, in $O(T_{sort}(kn, p))$ time.
- $\cup_{a \in P} \mathcal{R}(a)$ is sorted as follows. Each $\mathcal{R}(a)$ is sorted lexicographically, then the union of the lists $\mathcal{R}(a)$ (for all $a \in P$) is sorted lexicographically.
- In the last step, a translation vector T takes P into a subset of S if and only if T appears as the translation component of k consecutive entries of the ordered list $\cup_{a \in P} \mathcal{R}(a)$.

■

7 Further remarks

7.1 Summary

In this paper, we have given examples of optimal and efficient scaleable parallel algorithms for the following.

- Finding all rectangles determined by a set of planar points. (We have also indicated solutions to some related problems.)

- Describing the lower envelope function for a set of polynomials of bounded degree.
- A variety of geometric problems whose solutions are dominated by description of lower (or upper) envelopes.
- Finding all maximal equally-spaced collinear subsets of a finite set in a Euclidean space.
- Solving various versions of the Point Set Pattern Matching Problem in Euclidean spaces.

As far as we know, our algorithms are in all cases the first scaleable parallel algorithms given in solution to their respective problems. In many cases, they are the first parallel algorithms given in solution to their respective problems for machines of any granularity.

7.2 Acknowledgment

We acknowledge suggestions of the anonymous referees that helped improve the presentation of our results.

References

[Agar90] P.K. Agarwal, Partitioning arrangements of lines: II, applications, *Discrete and Computational Geometry* 5 (1990), 533-573.

[Agar91] P.K. Agarwal, *Intersection and Decomposition Algorithms for Planar Arrangements*, Cambridge University Press, Cambridge, 1991.

[AShSh89] P.K. Agarwal, M. Sharir, and P. Shor, Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences, *Journal of Combinatorial Theory Series A* 52 (1989), 228-274.

[A&L93] S.G. Akl and K.A. Lyons, *Parallel Computational Geometry*, Prentice-Hall, New York, 1993.

[Atal85a] M.J. Atallah, Some dynamic computational geometry problems. *Computers and Mathematics with Applications* 11 (1985), 1171-1181.

[Boxe92] L. Boxer, Finding congruent regions in parallel, *Parallel Computing* 18 (1992), 807-810.

[Boxe96] L. Boxer, Point set pattern matching in 3-D, *Pattern Recognition Letters* 17 (1996), 1293-1297.

[Boxe97] L. Boxer, A scaleable parallel algorithm for the Hausdorff metric in digitized pictures, submitted.

[B&M89a] L. Boxer and R. Miller, Parallel dynamic computational geometry, *Journal of New Generation Computer Systems* 2 (1989), 227-246.

[B&M89b] L. Boxer and R. Miller, Dynamic computational geometry on meshes and hypercubes, *Journal of Supercomputing* 3 (1989), 161-191.

[B&M90] L. Boxer and R. Miller, Common intersections of polygons, *Information Processing Letters* 33 (1990), 249-254; *Corrigenda* in *Information Processing Letters* 35 (1990), 53.

[B&M93] L. Boxer and R. Miller, Parallel algorithms for all maximal equally-spaced collinear sets and all maximal regular coplanar lattices, *Pattern Recognition Letters* 14 (1993), 17-22.

[BMR96a] L. Boxer, R. Miller, and A. Rau-Chaplin, Some scalable parallel algorithms for geometric problems, SUNY at Buffalo Department of Computer Science Technical Report 96-12 (1996).

[BMR96b] L. Boxer, R. Miller, and A. Rau-Chaplin, Some scaleable parallel algorithms for geometric problems, *Proceedings IASTED Conference on Parallel and Distributed Computing and Systems* (1996), 426-430.

[Chaz91] B. Chazelle, An optimal convex hull algorithm and new results on cuttings, *Proc. 32nd IEEE Symposium on Foundations of Computer Science* (1991), 29-38.

[CEGSW90] Clarkson, K.L., H. Edelsbrunner, L.J. Guibas, M. Sharir, and E. Welzl, Combinatorial complexity bounds for arrangements of curves and surfaces, *Discrete and Computational Geometry* 5 (1990), 99-160.

[CKPSSSSE] 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.

[D&S65] H. Davenport and A. Schinzel, A combinatorial problem connected with differential equations. *Amer. J. Math.* 87 (1965), 684-694.

[DFR93] F. Dehne, A. Fabri, and A. Rau-Chaplin, Scalable parallel geometric algorithms for multicomputers, *Proc. 9th ACM Symp. on Computational Geometry*, (1993), 298-307.

[DDDFK95] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. Khokhar, A randomized parallel 3D convex hull algorithm for coarse grained multicomputers, *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, (1995), 27-33.

[De&Dy95] X. Deng and P. Dymond, Efficient routing and message bounds for optimal parallel algorithms, *Proceedings International Parallel Processing Symposium*, (1995), 556-562.

[dR&L95] P.J. de Rezende and D.T. Lee, Point set pattern matching in $d-$dimensions, *Algorithmica* 13 (1995), 387-404.

[Erd46] P. Erdös, On a set of distances of n points, *American Mathematical Monthly* 53 (1946), 248-250.

[FRU95] 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.

[G&K92] Goodrich, M.T., and D. Kravetz, Point set pattern matching, Johns Hopkins University Department of Computer Science Technical Report JHU-92/11 (1992).

[H&K93] 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).

[H&Sh86] S. Hart and M. Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes, *Combinatorica* 6 (1986), 151-177.

[Hersh89] J. Hershberger, Finding the upper envelope of n line segments in $O(n \log n)$ time, *Information Processing Letters* 33 (1989), 169-174.

[K&R91] A.B. Kahng and G. Robins, Optimal algorithms for extracting spatial regularity in images, *Pattern Recognition Letters* 12 (1991), 757-764.

[L&L92] C-L Lei and H-T Liaw, A parallel algorithm for finding congruent regions, *Computers and Graphics* 16 (1992), 289-294.

[M&S96] R. Miller and Q.F. Stout, *Parallel Algorithms for Regular Architectures: Meshes and Pyramids*, The MIT Press, Cambridge, Mass., 1996.

[Nadl78] S.B. Nadler, Jr., *Hyperspaces of Sets*, Marcel Dekker, Inc., New York, 1978.

[P&Sh92] J. Pach and M. Sharir, Repeated angles in the plane and related problems, *Journal of Combinatorial Theory* 59 (1) (1992), 12-22.

[Reic88] M. Reichling, On the detection of a common intersection of k convex objects in the plane, *Information Processing Letters* 29 (1988), 25-29.

[Röte91] G. Röte, Computing the minimum Hausdorff distance between two point sets on a line under translation, *Information Processing Letters* 38 (1991), 123-127.

[SL&Y90] Z.C. Shih, R.C.T. Lee, and S.N. Yang, A parallel algorithm for finding congruent regions, *Parallel Computing* 13 (1990), 135-142.

[SST84] J. Spencer, E. Szemeredi, and W.T. Trotter, Jr., Unit distances in the Euclidean plane, in *Graph Theory and Combinatorics*, Academic Press, London, 1984, 293-303.

[Vali90] L.G. Valiant, A Bridging Model for Parallel Computation, *Communications of the ACM* 33 (1990), 103-111.

[VK&D91] M.J. Van Kreveld and M.T. De Berg, Finding squares and rectangles in sets of points, *BIT* 31 (1991), 202-219.

[Wi&Sh88] A. Wiernik and M. Sharir, Planar realization of nonlinear Davenport-Schinzel sequences by segments, *Discrete and Computational Geometry* 3 (1988), 15-47.