Journal of Parallel and Distributed Computing **58**, 466–486 (1999)
Article ID jpdc.1999.1565, available online at http://www.idealibrary.com on IDEAL®

Scalable Parallel Algorithms for Geometric Pattern Recognition

Laurence Boxer*

Department of Computer and Information Siences, Niagara University, New York 14109 and Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, New York 14260

Russ Miller†

Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, New York 14260

and

Andrew Rau-Chaplin‡

Faculty of Computer Science, Dalhousie University, P.O. Box 1000, Halifax, Nova Scotia, Canada B3J 2X4

Received October 1, 1998; revised May 3, 1999; accepted May 14, 1999

This paper considers a variety of geometric pattern recognition problems on input sets of size n using a *coarse grained multicomputer* model consisting of p processors with $\Omega(n/p)$ local memory each (i.e., $\Omega(n/p)$ memory cells of $\Theta(\log n)$ bits apiece), where the processors are connected to an arbitrary interconnection network. It introduces efficient scalable parallel algorithms for a number of geometric problems including the *rectangle finding problem*, the *maximal equally spaced collinear points problem*, and the *point set pattern matching problem*. All of the algorithms presented are *scalable* 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 scalability, 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: parallel algorithms; computational geometry; scalable algorithms; coarse grained multicomputer.

* E-mail: boxer@niagara.edu. Research partially supported by a grant from the Niagara University Research Council.

† E-mail: miller@cse.buffalo.edu. Research partially supported by NSF Grant IRI9412415.

‡ E-mail: arc@tuns.ca. Research partially supported by Natural Sciences and Engineering Research Council of Canada.

0743-7315/99 $30.00

1. INTRODUCTION

Geometric pattern recognition is an important area of research with applications in computer image processing, manufacturing, robotics, VLSI design, military intelligence, etc. A typical problem in parallel computational geometry calls for an efficient solution to a query involving n geometric objects (e.g., points, lines, polygons) on a parallel computer with p processors. Much previous theoretical work in parallel computational geometry has assumed fine grained parallelism, i.e., $n/p = \Theta(1)$ 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 *scalable*, i.e., implementable and efficient over a wide range of ratios of n/p. There has been much recent interest in coarse-grained computational models [Vali90, CKPSSSSE, H&K93] and the design of coarse grained geometric algorithms [BMR98, DFR93, De&Dy95, DFRU99, DDDFK95], motivated in part by the observation that "fast algorithms" for fine-grained models rarely translate into fast code running on coarse-grained machines. This paper continues this effort by describing new scaleable algorithms for a variety of problems in pattern recognition.

The paper is organized as

- 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 scaleable parallel algorithm to find all maximal equally spaced collinear subsets of a finite point set in a Euclidean space.
- Section 5. 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 6. We give some concluding remarks.

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 $\Omega(n/p)$ local memory each (i.e., $\Omega(n/p)$ memory cells of $\Theta(\log n)$ bits apiece in every processor). The processors may be connected to some arbitrary interconnection network or may share global memory. Commonly used interconnection networks for a CGM include the

2D-mesh, 3D-mesh, hypercube, and the fat tree. A processor may exchange messages of $O(\log n)$ bits with any one of its immediate neighbors in constant time. In determining time complexities, we consider both local computation time and interprocessor communication time, in the standard way. The term "coarse-grained" refers to the fact that the size $\Omega(n/p)$ of each local memory is assumed to be "considerably larger" than $\Theta(1)$. Our definition of "considerably larger" will be that $n/p \geqslant p$. Thus, each processor has least enough local memory to store the ID number of every other processor. For a more detailed description of the model and its associated operations, see [DFR93].

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 the context.

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

- $x_1 < x'_1$; or
- for some integer j, $1 \leqslant j < k$, $x_1 = x'_1$ and $x_2 = x'_2$ and $\cdots$ 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 $T_{\text{par}} = \Theta(T_{\text{seq}}/p)$, 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 cause T_{par} to be asymptotically greater than $\Theta(T_{\text{seq}}/p)$.

We denote by $T_{\text{sort}}(n, p)$ 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.

- *Permutation exchange.* Let $\sigma: \{1, 2, ..., p\} \to \{1, 2, ..., p\}$ be a permutation (a function that is one-to-one and onto). Every processor P_i sends a list of 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 $X = \{x_1, ..., x_n\}$ 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 $x_1 \circ x_2 \circ \cdots \circ x_n$. Examples of such operations include *total*, *product*, *minimum*, *maximum*, *and*, and *or*.

• *Parallel prefix.* Let $X=\{x_1, ..., x_n\}$ 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 $\{x_1, x_1 \circ x_2, ..., x_1 \circ x_2 \circ \cdots \circ x_n\}$.

• *Parallel search.* Let $X=\{x_1, ..., x_k\}$ and $Y=\{y_1, ..., y_n\}$ 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 $X=\{x_1, ..., x_n\}$ 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, $\{\{x_{i_1}, ..., x_{i_k}\} \mid 1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n\}$.

• *Formation of pairs from lists.* Let $X=\{x_1, ..., x_k\}$ and let $Y=\{y_1, ..., y_n\}$. 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 [BMR98]. *For positive integers k, n, p, we have*

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

The next several results discuss algorithms for fundamental data operations that are implemented using sorting.

PROPOSITION 2.2. *A permutation exchange operation may be implemented in time $T_{\text{sort}}(n, p)$ 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 n/p data items. This takes $\Theta(n/p)$ time.

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

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

PROPOSITION 2.3 [BMR98]. *A semigroup operation on evenly distributed data $x_1, ..., x_n$ may be implemented in time $\Theta(n/p) + T_{\text{sort}}(p^2, p)$ on a CGM(n, p). At the end of this operation, all processors have the value of $X = x_1 \circ \cdots \circ x_n$.*

PROPOSITION 2.4 [BMR98]. *A parellel prefix operation may be implemented in $\Theta(n/p) + T_{\text{sort}}(p^2, p)$ time on a CGM(n, p). At the end of the operation, the prefix $x_1 \circ x_2 \circ \cdots \circ x_i$ is in the same processor as x_i, $i \in \{1, 2, ..., n\}$.*

PROPOSITION 2.5. *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 $T_{\text{sort}}(k+n, p)$ 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 $T_{\text{sort}}(k, p)$ time.

2. Let x_i' be the value sought by x_i. Let $X' = \{x_1', ..., x_k'\}$. For each x_i, create a record r_i with components x_i, x_i', and *report*. Let $R = \{r_1, ..., r_k\}$. This takes $\Theta(k/p)$ time.

3. Sort $R \cup Y$, using the x_i' component of members of R as the key field. This takes $T_{\text{sort}}(k+n, p)$ 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 $\Theta((k+n)/p) + T_{\text{sort}}(p^2, p)$ time.

5. Sort the members of R (found in $R \cup Y$) by the x_i component. This takes $O(T_{\text{sort}}(k+n, p))$ 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 $\Theta(k/p)$ time.

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

PROPOSITION 2.6. *Let $X = \{x_1, ..., x_n\}$. Let $k > 1$ be a fixed integer. Then the set of all combinations of members of X with k members apiece, $\{\{x_{i_1}, ..., x_{i_k}\} \mid 1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n\}$ can be formed in*

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

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

Proof. The algorithm follows.

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

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

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

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

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

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

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

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

PROPOSITION 2.7. *Let* $X = \{x_1, ..., x_k\}$ *and* $Y = \{y_1, ..., y_n\}$ *be two lists evenly distributed among the processors of a* CGM(kn, p), *with* $k \leqslant n$. *Then the set*

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

may be computed in

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

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

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

1. Allocate space for the array

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

its entries uninitialized, in $O(kn/p)$ time.

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

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 $\Theta(kn/p)$ time.

Thus, the algorithm requires

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

time.

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

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

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

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

and the running time is therefore $\Theta(kn/p)$, which is optimal, since there is $\Theta(kn)$ output. ▮

In several of our algorithms, it is desirable to remove, efficiently, duplicate list entries. We have the following.

LEMMA 2.8. *Let M be a list of n sets, each of cardinality k. Then duplicate members may be removed from M in $\Theta(T_{\text{sort}}(kn, p))$ time on a* CGM(kn, p).

Proof. We give the following algorithm:

1. Sort each of the k-tuples in M lexicographically. This takes $T_{\text{sort}}(kn, p)$ time.

2. Now, sort M lexicographically. This takes $T_{\text{sort}}(kn, p)$ time.

3. Perform a prefix operation to remove every entry of the ordered list M that equals its predecessor. Since the running time of the algorithm of Proposition 2.4 is based on the assumption of prefix values with complexity $\Theta(1)$, while the prefix values in the current operation have complexity $\Theta(k)$, this takes $\Theta(kn/p + T_{\text{sort}}(kp^2, p))$ time, which, since $p^2 \leqslant n$, is $O(T_{\text{sort}}(kn, p))$.

Thus, the algorithm requires $\Theta(T_{\text{sort}}(kn, p))$ time. ▮

3. RECTANGLE PROBLEMS

In this section, we give a scalable 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.*

Our CGM solution to the AR problem is obtained by forming all the line segments with endpoints in S, then sorting these segments so that sweeps (parallel prefix operations) of the ordered segments will yield the rectangles. The algorithm follows.

THEOREM 3.2. *Let $S = \{v_0, v_1, ..., v_{n-1}\}$ be given as input. Then the AR problem can be solved in $T_{\text{sort}}(n^2 \log n, p)$ 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 $O(T_{\text{sort}}(n^2, p))$ time by a trivial modification to the algorithm associated with Proposition 2.6.

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 nonzero 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 $T_{\text{sort}}(n^2, p)$ 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

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

Also determine for each $\ell \in L$

$$r_\ell = \text{ord}(\ell_1) - \text{ord}(\ell),$$

the number of rectangles for which ℓ is the first side, and

$$P_\ell = \sum_{\ell' < \ell} r_{\ell'},$$

the number of rectangles whose first sides precede ℓ. By Proposition 2.4, these operations require $\Theta(n^2/p) + T_{\text{sort}}(p^2, p)$ time.

4. Assign the first side of each of the $O(n^2 \log n)$ rectangles as follows. The ith rectangle, $P_\ell < i \leqslant P_\ell + r_\ell$, 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 (by Proposition 2.5) $T_{\text{sort}}(n^2 \log n, p)$ 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 ith rectangle, $P_\ell < i \leqslant P_\ell + r_\ell$, has for its second side the member of L whose index in L is $\text{ord}(\ell) + (i - P_\ell)$. Thus, the second side of all rectangles may be determined via parallel search operations in $T_{\text{sort}}(n^2 \log n, p)$ time.

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

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

Problem	Worst case output	T_{par}
All isonormal rectangles	$\Theta(n^2)$	$T_{\text{sort}}(n^2, p)$ on CGM(n^2, p)
All squares	$\Theta(n^2)$	$T_{\text{sort}}(n^2, p)$ on CGM(n^2, p)

4. MAXIMAL COLLINEAR SETS

In this section, we give a scalable 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 $\{s_1, ..., s_k\}$ 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.

Our algorithm is based on sorting steps, searches, and sweeps reminiscent of those in standard propagation algorithms. We give the algorithm below.

THEOREM 4.1. *Let d be a fixed positive integer. Let $S \subset R^d$, $|S| = n$. Then the AMESCS problem can be solved for S in $\Theta(T_{\text{sort}}(n^2, p))$ 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 $T_{\text{sort}}(n, p)$ 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 2.6 in $O(T_{\text{sort}}(n^2, p))$ 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 $T_{\text{sort}}(n^2, p)$ 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 $\Theta(n^2/p)$ 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 $\Theta(T_{\text{sort}}(n^2, p))$ 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 $\Theta(n^2/p)$ 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 $\{L_{i,j} \mid 1 \leqslant i < j \leqslant n\}$ of the same length are consecutive, and
- if $x_k = z_{(i,j)}$, then $L_{i,j} < L_{j,k}$.

Let $M = \{m_s \mid s = 1, ..., N\}$ be an enumeration of the members of $\{L_{i,j} \mid 1 \leqslant i < j \leqslant n\}$, $m_i < m_{i+1}$, 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 $\Theta(n^2/p + T_{\text{sort}}(p^2, p))$ time, to compute all of the members of

$$\{m_1, m_1 \circ m_2, ..., m_1 \circ m_2 \circ \cdots \circ m_N\},$$

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 $T_{\text{sort}}(n^2, p)$ time.

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

5. POINT SET PATTERN MATCHING

In this section, we give scalable 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, Boxe98, dR&L95, G&K92, L&L92, SL&Y90].

We assume that $|S| = n$, $|P| = k \leqslant n$, and that the coordinates of all members of $P = \{a_0, a_1, ..., a_{k-1}\}$ and $S = \{s_0, s_1, ..., s_{n-1}\}$ 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.

5.1. 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 $\Theta(k(n-k))$. Our CGM algorithm is based on determining which translations (or, reflection followed by translations) T of $a_0 \in P$, such that $T(a_0) \in S$, satisfy $T(P) \subset S$.

THEOREM 5.1. *The point set pattern matching problem in R^1 can be solved on a* CGM$(k(n-k), p)$ *in* $\Theta(T_{\text{sort}}(k(n-k), p))$ *time.*

Proof. We give the following algorithm.

1. Sort the members of S by their coordinates in $T_{\text{sort}}(n, p)$ time.
2. Sort the members of P by their coordinates in $T_{\text{sort}}(k, p)$ time.
3. Broadcast a_0 to all processors. This takes $O(p)$ time.
4. For $j \in \{0, 1, ..., n-k-1\}$, compute $d_j = s_j - a_0$. These values represent translations T of a_0 into a member of S such that at least $k-1$ members of S are greater than $T(a_0)$. This takes $\Theta((n-k)/p)$ time.
5. For $i \in \{0, 1, ..., k-1\}$, $j \in \{0, 1, ..., n-k-1\}$, 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 $\Theta(T_{\text{sort}}(k(n-k), p))$ time.
6. In $T_{\text{sort}}(k(n-k), p)$ 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 $\Theta(k(n-k)/p + T_{\text{sort}}(p^2, p))$ time, perform a parallel prefix operation on the $A_{i,j}$ to determine which indices j yield such translations. Let L_q be the qth 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 $\Theta(k(n-k)/p) + T_{\text{sort}}(p^2, p)$ 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 $1 \leqslant r \leqslant k-1$. 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 $\Theta(T_{\text{sort}}(k(n-k), p))$ 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 via the algorithm of Lemma 2.8 in $O(T_{\text{sort}}(k(n-k), p))$ time.

Thus, the algorithm takes $\Theta(T_{\text{sort}}(k(n-k), p))$ time. This is optimal if we wish our output to be ordered, as, in the worst case, there is $\Theta(k(n-k))$ output. ▮

5.2. *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 5.2 [SST84]. $D_2(n) = O(n^{4/3})$.

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

PROPOSITION 5.3. *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 $\{s_{i_2}, ..., s_{i_{k-1}}\} \subset S$ such that $\{s_{i_0}, s_{i_1}, s_{i_2}, ..., s_{i_{k-1}}\}$ 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 two such values of i_2, each of which determines at most one 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 5.4 [Agar90, Chaz91]. *For any fixed $\delta > 0$, $A_2(n) = O(n^{4/3+\delta})$.*

THEOREM 5.5 [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.* ▮

Our CGM algorithm for solving the point set pattern matching problem in R^2 is based on finding which rigid transformations T of the Euclidean plane, of those that take a fixed line segment with endpoints in P to some line segment with endpoints in S, satisfy $T(P) \subset S$. The algorithm is given below.

THEOREM 5.6. *The point set pattern matching problem in R^2 can be solved in*

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

time on a CGM($kD_2(n), p$). *For $p = O(kD_2(n)/n)$, the running time is*

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

Proof. Note it follows from Theorem 5.5 that $A_2(n)/p = O(T_{\text{seq}}/p)$. 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

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

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 $O(D_2(n)/p)$ 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 $O(T_{\text{sort}}(D_2(n), p))$ 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 $s_{i_3}, ..., s_{i_{k-1}}$ in S such that $(s_{i_0}, s_{i_1}, s_{i_2}, s_{i_3}, ..., s_{i_{k-1}})$ matches P.

This is done as follows.

- For each such triple $(s_{i_0}, s_{i_1}, s_{i_2})$ and each $j \in \{3, 4, ..., k-1\}$, 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 $O(kD_2(n)/p)$ time.

- For each such z_j, determine whether $z_j \in S$. If so, let the jth 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 jth component of this k-tuple be *fail*. This may be done a via parallel search operation in $T_{\text{sort}}(kD_2(n), p)$ 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 $\Theta(kD_2(n)/p) + T_{\text{sort}}(p^2, p)$ 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 by the algorithm of Lemma 2.8 in $O(T_{\text{sort}}(kD_2(n), p))$ time.

Thus, the algorithm requires

$$O\left(\frac{A_2(n)}{p} + pT_{\text{sort}}(n, p) + T_{\text{sort}}(kD_2(n), p)\right)$$

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

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

$$O\left(\frac{A_2(n)}{p} + T_{\text{sort}}(kD_2(n), p)\right) = O\left(\frac{T_{\text{seq}}}{p} + T_{\text{sort}}(kD_2(n), p)\right). \quad \blacksquare$$

5.3. *PSPM in R^3*

In this section, we present a scalable parallel algorithm for solving the point set pattern matching problem in R^3. The following considerations are used to construct an upper bound on the complexity of the output.

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 polynomial pieces of the minimum or *lower envelope* function of S is denoted by $\lambda(n, k)$. It was shown in [Atal85] that $\lambda(n, k)$ is the maximal length of a *Davenport–Schinzel sequence* [D&S65] defined by parameters n and k.

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]). In the current discussion, we only use $k = 6$. We have the following, as an example of a more general result.

THEOREM 5.7 [AShSh89].

$$\lambda(n, 6) = O(n \cdot 2^{O([\alpha(n)]^2)}).$$

PROPOSITION 5.8 [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 $O(n^{3/2}[\lambda(n, 6)/n]^{1/4})$.*

It follows from Theorem 5.1 that the expression $\lambda(n, 6)/n$, which appears in the analysis of our algorithm, is nearly constant. We have the following.

PROPOSITION 5.9 [Boxe98]. *The output of the point set pattern matching problem in R^2 has complexity $O(kn^2[\lambda(n, 6)/n]^{1/2})$.*

THEOREM 5.10 [Boxe98]. *The point set pattern matching problem in R^3 can be solved on a serial computer in*

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

Next, we give an algorithm for a special case.

PROPOSITION 5.11. *Let P and S be finite subsets of R^3. Let $|P| = k \leqslant n = |S|$. Suppose 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 on a* $\mathrm{CGM}(n^2 + kn^{3/2}[\lambda(n, 6)/n]^{1/4}, p)$ *in*

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

Proof. Let $S = \{s_0, s_1, ..., s_{n-1}\}$. Let $P = \{p_0, p_1, ..., p_{k-1}\}$. We give the following algorithm.

1. Sort P by lexicographic order. This takes $T_{\mathrm{sort}}(k, p)$ time.

2. Sort S by lexicographic order. This takes $T_{\mathrm{sort}}(n, p)$ time.

3. Form the list C of all the ordered pairs (s_i, s_j), $i \neq j$, of distinct members of S. By Proposition 2.6, this takes $\Theta(n^2/p + p \cdot T_{\mathrm{sort}}(n, p))$ time. Note $|C| = \Theta(n^2)$.

4. In $O(p)$ time, broadcast p_0 and p_{k-1} to all processors.

5. Use a prefix operation to form the list C' of members of C representing line segments whose length equals the length of the line segment from p_0 to p_{k-1}. By Proposition 2.4, this takes $\Theta(n^2/p + T_{\mathrm{sort}}(p^2, p))$ time. By Proposition 5.1, $|C'| = O(n^{3/2}[\lambda(n, 6)/n]^{1/4})$.

6. For every $(s_i, s_j) \in C'$, to identify a subset of S that matches P including a submatch of (s_i, s_j) with (p_0, p_{k-1}), it is necessary and sufficient to determine if there exists a $(k-2)$-tuple $(s_{i_1}, ..., s_{i_{k-2}})$ such that for each i_m, $m \in \{1, ..., k-2\}$,

- $s_{i_m} \in S$,
- 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 $\overline{p_0 p_m}$.

For all $(s_i, s_j) \in C'$ do the following:

- Determine the desired points $\{s_{i_m}\}_{m=1}^{k-1}$ such that $\{s_i, s_j\} \cup \{s_{i_m}\}_{m=1}^{k-1}$ match P. This may be done in $\Theta(k\,|C'|/p) = O(kn^{3/2}/p[\lambda(n, 6)/n]^{1/4})$ time.

• Perform a parallel search to determine (for all i, j, i_m) if $s_{i_m} \in S$. By Proposition 2.5, this may be done in

$$O\left(T_{\text{sort}}\left(k + n^{3/2}\left[\frac{\lambda(n,6)}{n}\right]^{1/4}, p\right)\right) = O\left(T_{\text{sort}}\left(kn^{3/2}\left[\frac{\lambda(n,6)}{n}\right]^{1/4}, p\right)\right) \text{ time.}$$

• Use a parallel prefix operation to consolidate the matches of P in S found in the previous step into a contiguous list. This may be done in $O(T_{\text{sort}}(kn^{3/2}[\lambda(n,6)/n]^{1/4}, p))$ time.

The list of matches of P in S has complexity $O(kn^{3/2}[\lambda(n,6)/n]^{1/4})$.

7. It may happen that the same subset of S appears in our list M of matches of P twice. If we wish to eliminate the duplications of subsets of S represented in M, we may do so via the algorithm of Lemma 2.8 in $O(T_{\text{sort}}(kn^{3/2}[\lambda(n,6)/n]^{1/4}, p))$ time.

Thus, our algorithm takes

$$O\left(\frac{n^2}{p} + T_{\text{sort}}\left(kn^{3/2}\left[\frac{\lambda(n,6)}{n}\right]^{1/4}, p\right)\right) \text{ time.} \quad \blacksquare$$

Proposition 5.9 follows from the next lemma, which we use to prove Theorem 5.13.

LEMMA 5.12 [Boxe98]. *Let P and S be finite subsets of $\mathscr{R}^3$, with $|P| = 3 \leqslant n = |S|$. Then a listing of all three-member subsets P' of S such that two line segments determined by P' match two line segments determined by P, has $O(n^2[\lambda_6(n)/n]^{1/2})$ output.*

Our CGM algorithm for solving the general point set pattern matching problem in R^3 may be described as follows. First, determine if P is a collinear set. If P is collinear, apply the algorithm of Proposition 5.11. Otherwise, there is a (non-collinear) triangle Δ in P, so we determine which rigid transformations T of R^3, of those that take Δ to some triangle with vertices in S, satisfy $T(P) \subset S$. The algorithm is given below.

THEOREM 5.13. *Let P and S be finite subsets of R^3. Let $|P| = k \leqslant n = |S|$. Then every subset P' of S such that P' is congruent to P can be identified on a* CGM$(kn^2[\lambda(n,6)/n]^{1/2}, p)$, *in*

- $O[n^2/p + T_{\text{sort}}(kn^{3/2}[\lambda(n,6)/n]^{1/4}, p)]$ *time, if P is a collinear set;*
- $O(T_{\text{sort}}(kn^2[\lambda(n,6)/n]^{1/2}, p))$ *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. Use circular rotations of P among all processors so every processor has a copy of P. This takes $(p-1)\, T_{\text{sort}}(k, p) = O(T_{\text{sort}}(kp, p))$ time.

2. Determine whether or not P is a collinear set. This is done as follows. Note each processor has a_0 and a_1. For each $k \in \{2, ..., k-1\}$, determine if a_k is

collinear with a_0 and a_1, in $\Theta(k/p)$ time. P is a collinear set if and only if a_k is collinear with a_0 and a_1 for all $k \in \{2, ..., k-1\}$. 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 $P \backslash \{a_0, a_1\}$ in $\Theta(k/p + T_{\text{sort}}(p^2, p))$ time (Proposition 2.3), followed by an $O(p)$ time broadcast of a_r to all processors.

3. If P is a collinear set, execute the algorithm of Proposition 5.1. This finishes the current algorithm in an additional $O[n^2/p + T_{\text{sort}}(kn^{3/2}[\lambda(n, 6)/n]^{1/4}, p)]$ time. Otherwise, continue with the following steps.

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

5. 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 2.6, this step takes $O(T_{\text{sort}}(n^2, p))$ time.

6. 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 $\Theta(k/p)$ time.

7. 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 $T_{\text{sort}}(n^2, p)$ time.

8. Let M be the number of members of $L(S)$ whose length is equal to the length of π_1. Mark the sublist L^1 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 $\Theta(n^2/p + T_{\text{sort}}(p^2, p))$. 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, note by Proposition 5.8 that

$$M = O\left(n^{3/2}\left[\frac{\lambda(n, 6)}{n}\right]^{1/4}\right).$$

9. 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 $\Theta(n^2/p + T_{\text{sort}}(p^2, p))$ time. As above, if $|L^r| = 0$, report failure and halt.

10. 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 parallel search on L^r to find, for each $S_{ij} \in L^1$, the subrange of members of L^r 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 $O(T_{\text{sort}}(M + |L^r|, p)) = O(T_{\text{sort}}(n^{3/2}[\lambda(n, 6)/n]^{1/4}, p))$ time. It follows from Lemma 5.2, that there are $O(n^2[\lambda(n, 6)/n]^{1/2})$ pairs (S_{ij}, S_{jm}) $S_{ij} \in L^1$, $S_{jm} \in L^r$, such that $S_{ij} \cup S_{jm}$ matches $\pi_1 \cup \pi_r$. Such pairs (S_{ij}, S_{jm}) may be formed by circular rotations of L^1 accompanied by the formation of pairs in $O(pT_{\text{sort}}(M, p) + n^2/p[\lambda(n, 6)/n]^{1/2})$ time. By Lemma 2.1, this is

$$O\left(T_{\text{sort}}(Mp, p) + \frac{n^2}{p}\left[\frac{\lambda(n, 6)}{n}\right]^{1/2}\right) = O\left(T_{\text{sort}}\left(n^2\left[\frac{\lambda(n, 6)}{n}\right]^{1/2}, p\right)\right)$$

(since $p \leqslant n^{1/2}$) time to form the $O(n^2[\lambda(n, 6)/n]^{1/2})$ such pairs (S_{ij}, S_{jm}). Note each pair (S_{ij}, S_{jm}) corresponds to a triple (s_i, s_j, s_m) of vertices in S that match (a_0, a_1, a_r).

11. 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 $O(n^2[\lambda(n, 6)/n]^{1/2})$ such triples, creating all such descriptions takes $O((n^2/p)[\lambda(n, 6)/n]^{1/2})$ time.

12. If $k > 3$, proceed as follows. For each of the $O(n^2[\lambda(n, 6)/n]^{1/2})$ rigid transformations f_{ijm} of R^3 determined above, compute the set

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

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

13. Among the sets $f_{ijm}(P) \subset S$ that match P, there may be duplicate sets determined by distinct f_{ijm}. If desired, we may eliminate such duplication by the algorithm of Lemma 2.2 in $O(T_{\text{sort}}(kn^2[\lambda(n, 6)/n]^{1/2}, p))$ time.

The algorithm requires

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

5.4. PSPM in R^2 under Rotations or Translations

In this section, we give scalable 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 $0 < k \leqslant n$.

We have the following.

THEOREM 5.14 [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 scalable parallel version of Theorem 5.14. Our algorithm for rotations is based on the observation that the set of rotations r of P about the origin such that $r(P) \subset S$ must be the intersection over all $a \in P$ of the set of rotations r of a about the origin such that $r(a) \in S$. A similar observation for translations is the key to our algorithm for translations.

THEOREM 5.15. • *Every rotation r of P about the origin such that $r(P) \subset S$ may be found in $O(T_{\text{sort}}(kn, p))$ time on a* CGM(kn, p).

• *Every translation T of P in R^2 such that $T(P) \subset S$ may be found in $O(T_{\text{sort}}(kn, p))$ time on a* CGM(kn, p).

Proof. Let $\mathscr{R}$ be the set of angles θ, $0 \leqslant \theta < 2\pi$, 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 $\mathscr{R}(a)$ be the set of angles θ, $0 \leqslant \theta < 2\pi$, such that a rotation r_θ of p by θ about the origin satisfies $r_\theta(a) \in S$. Then

$$\mathscr{R} = \bigcap_{a \in P} \mathscr{R}(a),$$

and, in the worst case, $|\mathscr{R}(a)| = n$ 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 $T_{\text{sort}}(n, p)$ time.

2. For all $a \in P$, compute $\mathscr{R}(a)$ 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 $O(T_{\text{sort}}(kn, p))$ time, as follows.

- Form $P \times S$ by the algorithm of Proposition 2.7 in $O(T_{\text{sort}}(kn, p))$ time.
- In $\Theta(kn/p)$ 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 $\bigcup_{i=1}^{k} \mathscr{R}(a_i)$ with respect to the angular coordinate. This takes $O(T_{\text{sort}}(kn, p))$ time.

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

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

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

The sets $\mathscr{R}(a)$ can all be computed after forming all pairs (a, s), where $a \in P$, $s \in S$, in $O(T_{\text{sort}}(kn, p))$ time.

• $\bigcup_{a \in P} \mathscr{R}(a)$ is sorted as follows. Each $\mathscr{R}(a)$ is sorted lexicographically, then the union of the lists $\mathscr{R}(a)$ (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 $\bigcup_{a \in P} \mathscr{R}(a)$. ∎

6. FURTHER REMARKS

6.1. Summary

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

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

ACKNOWLEDGMENTS

We gratefully acknowledge the suggestions of an anonymous referee concerning presentation of our results.

REFERENCES

[Agar90] P. K. Agarwal, Partitioning arrangements of lines. II. Applications, *Discr. Comput. Geom.* **5** (1990), 533–573.

[Agar91] P. K. Agarwal, "Intersection and Decomposition Algorithms for Planar Arrangements," Cambridge Univ. 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, *J. Combin. Theory Ser. A* **52** (1989), 228–274.

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

[Atal85] M. J. Atallah, Some dynamic computational geometry problems, *Comp. Math. Appl.* **11** (1985), 1171–1181.

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

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

[Boxe98] L. Boxer, Faster point set pattern matching in 3-D, *Pattern Recognition Letters* **19** (1998), 1235–1240.

[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 scalable parallel algorithms for geometric problems, *in* "Proceedings IASTED Conference on Parallel and Distributed Computing and Systems, ," pp. 426–430.

[BMR98] L. Boxer, R. Miller, and A. Rau-Chaplin, Scaleable parallel algorithms for lower envelopes with applications, *J. Parallel Distrib. Comput.* **53** (1998), 91–118.

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

[CEGSW90] K. L. Clarkson, 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, *in* "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, *in* "Proc. 9th ACM Symp. on Computational Geometry, 1993," pp. 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, *in* "Proc. 7th ACM Symp. on Parallel Algorithms and Architectures, 1995," pp. 27–33.

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

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

[DFRU99] M. Diallo, A. Ferreira, A. Rau-Chaplin, and S. Ubeda, Scalable 2d convex hull and triangulation algorithms for coarse grained multicomputers, *J. Parallel Distrib. Comput.* **56** (1999), 47–70.

[Erd46] P. Erdős, On a set of distances of n points, *Amer. Math. Monthly* **53** (1946), 248–250.

[G&K92] M. T. Goodrich 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.

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

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

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

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

[SL&Y90] Z. C. Shih, R. C. T. Lee, and S. N. Yang, A parallel algorithm for finding congruent regions, *Parallel Comput.* **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," pp. 293–303, Academic Press, London, 1984.

[Vali90] L. G. Valiant, A bridging model for parallel computation, *Comm. 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.