Dynamic Computational Geometry on Meshes and Hypercubes

LAURENCE BOXER
Department of Computer and Information Sciences, Niagara University, NY 14109

RUSS MILLER
Department of Computer Science, State University of New York, Buffalo, NY 14260

(Received October 1988; final version accepted May 1989.)

Abstract. Parallel algorithms are given for determining geometric properties of systems of moving point-objects. The objects are assumed to be moving in a Euclidean space such that each coordinate of a point's motion is a polynomial of bounded degree in the time variable. The properties investigated include nearest (farthest) neighbor, closest (farthest) pair, collision, convex hull, diameter, and containment. Several of these properties are investigated from both the dynamic and steady-state points of view. Efficient, and often optimal, implementations of these algorithms are given for the mesh and hypercube.

Key words and phrases: mesh, hypercube, dynamic computational geometry, parallel algorithms, nearest neighbor, closest pair, convex hull, smallest enclosing rectangle.

1. Introduction

Determining geometric properties of systems of objects has applications in areas such as pattern recognition, robotics, and air traffic control. Much is known about developing serial algorithms to determine geometric properties of objects for static systems (c.f. [Preparata and Lee 1984; Preparata and Shamos 1985]), and efficient parallel solutions have appeared recently for static systems (see [Miller and Stout 1989b] and the references contained therein). The literature of algorithms for describing geometric properties of dynamic systems includes [Atallah 1985; Boxer and Miller 1989; Chandran and Mount 1989]. The serial model of computation is used in [Atallah 1985], while [Boxer and Miller 1989] and [Chandran and Mount 1989] use the model of a concurrent read, exclusive write parallel random access machine (CREW PRAM).

Most supercomputers of the late 1980s are coarse-grained multiprocessors that rely on vector processing to achieve gigaflop speed (peak performance). These machines have typically been used to address problems in scientific computing. Examples of such machines include the 10-Gflops ETA-10, the 0.8-Gflops CYBER 205, the 1.3-Gflops NEC SX-2, and the CRAY series that ranges from the 0.16-Gflops CRAY-I to the 2-Gflops CRAY-2 (the CRAY-3 is expected to be an 8-processor 16-Gflops machine), where the ratings are the manufacturers' peak performance ratings. Advances in VLSI technology have also led to the recent production of commercially available massively parallel supercomputers. These

---

1 Research partially supported by a grant from the Niagara University Research Council.
2 Research partially supported by National Science Foundation grants DCR-8608640 and IRI-8800514.
machines include the 65,536-processor 2.5-Gflops Connection Machine (CM-2) [Hillis 1985], which can be configured as a mesh or as a hypercube, and the 16,384-processor 0.4-Gflops (6.5-Gops) Massively Parallel Processor (MPP) [Potter 1985], a mesh. Additional fine-grained machines, some of which are commercially available, include the ILLIAC IV [Barnes et al. 1968], DAP [Reddaway 1978], CLIP [Duff and Watson 1977], and SOLOMON I [Slotnick et al. 1962], each of which is configured as a mesh of processors. Commercially available medium-grained machines (both meshes and hypercubes), such as those produced by Intel [Intel 1986], NCube [Hayes et al. 1986], FPS [Gustafson et al. 1986], and Ametek [Ametek 1986], have the potential for gigaflop computing in the near future. In fact, one medium-grained machine promising gigaflop speed in the near future is the 576-processor 11-Gflops GF11 [Beetem et al. 1985].

In this paper, we concentrate on developing efficient algorithms for massively parallel mesh-connected computers like the MPP, CLIP, or DAP and for massively parallel hypercube computers like the CM-2. Algorithms for these massively parallel supercomputers will, necessarily, be significantly different from algorithms that could be developed for coarse-grained supercomputers. Specifically, we give implementations of algorithms for the two-dimensional mesh and for the hypercube.

Much of the previous work in algorithms for computational geometry on fine-grained meshes or hypercubes has been done for graphs [Atallah and Kosaraju 1984; Atallah and Hambrusch 1985; Nassimi and Sahni 1980], digitized pictures [Cypher et al. 1987a, 1987b; Cypher et al. 1989; Dehne et al. 1987; Miller and Stout 1985a, 1985b; Miller and Stout 1987; Stout 1983], and for static systems of objects [Dehne 1986; Miller and Stout 1988a, 1988b; Miller and Stout 1989a; Sanz and Cypher 1987]. The work in this paper is of a very different nature, as we are concerned with describing geometric properties of systems of moving point-objects given as functions of time.

Suppose \( n \) point-objects are moving in a Euclidean space of fixed dimension such that for some integer \( k \geq 0 \), every coordinate of each object’s motion is a polynomial of time whose degree is at most \( k \). For such a system, we present parallel algorithms described in terms of abstract data movement operations to solve a variety of problems involving proximity, collision, containment, and convexity. We give solutions to these problems for the dynamic situation and for the steady-state situation (as time approaches infinity). We give implementations of these algorithms on the mesh that are asymptotically optimal, and we give implementations on the hypercube that are within a logarithmic factor of the best CREW PRAM algorithms known [Chandran and Mount 1989]. Our algorithms produce ordered descriptions of their solutions and run asymptotically as fast as the time needed to sort deterministically. It is possible that these algorithms can be implemented on other architectures, such as the cube-connected cycles or shuffle-exchange network, to give efficient algorithms for these architectures.

Several of our algorithms for dynamic problems rely on being able to create a description of the minimum function (also called the lower envelope)

\[
h(t) = \min\{f_0(t), \ldots, f_{n-1}(t)\}
\]

(1)

from descriptions of \( n \) polynomials \( f_0, \ldots, f_{n-1} \) representing the trajectories of point-objects \( P_0, P_1, \ldots, P_{n-1} \), respectively, over time. Notice that \( h(t) \) is described by an ordered list of pieces, where each piece consists of an interval of time and a description of the function that is the minimum on the interval, and the interior.

The algorithm of [Chandran and Mount 1989] runs in \( O(\log n) \) time. The algorithms that are based on abstract data movement operations in this paper require that could be obtained by direct simulation on the hypercube, respectively. Since an \( n \)-processor current write operations in \( \Theta(n^{1.5}) \) time, during \( h(t) \) on a mesh would require \( \Theta(n^{1.5}) \) implementation of our algorithm has a \( \Theta(n^{1.5}) \) time. Since an \( n \)-processor current write operations in \( \Theta(\log^2 n) \) time (Valiant 1987), direct simulation of the PRAM would require \( \Theta(\log^2 n) \) time and expect hypercube implementations of our algorithm.

The processor requirements of some obtain function \( \lambda(n, s) \) that represents an upper bound in the system. It is known that \( \lambda(n, s) \) is \( \Theta(\log n) \) [Davenport and Schinzel 1965; Hart and Schinzel 1969]. We showed that the deviation of \( \lambda(n, s) \) from \( \Theta(\log n) \) (see Theorem 2.3).

The paper is organized as follows. Section 3 of computation, states assumptions, definitions, and a series of results concerning fundamental data movement operations. In Section 3, Equation (I) efficiently. The subject of Section 3 with time for a dynamic system. The properties from a given point, closest and farthest points, points of the convex hull, and several questions into which a dynamic system will fit as a function of the steady-state standpoint the properties of convex pairs, description of the convex hull, diameter, area rectangle that encloses the system.

2. Preliminaries

2.1. Order Notation

The notations \( O \), \( \Theta \), and \( \Omega \) will be used in this paper as "order exactly," and "order at least," respectively. In particular, \( \omega(1) \) is defined for all positive integers \( n \). The

DYNAMIC COMPUTATIONAL GEOMETRY

that is the minimum on the interval, and the interior.
that is the minimum on the interval, and where the intervals of the pieces have disjoint interiors.

The algorithm of [Chandran and Mount 1989] for describing $h(t)$ on an $n$-processor PRAM runs in $O(\log n)$ time. The algorithms that we present are given in a machine-independent fashion based on abstract data movement operations. The implementations of the parallel algorithms presented in this paper for the mesh and hypercube are faster than algorithms that could be obtained by direct simulation of the PRAM algorithm on the mesh and hypercube, respectively. Since an $n$-processor mesh can perform concurrent read and concurrent write operations in $\Theta(n^{1/2})$ time, direct simulation of the PRAM algorithm for describing $h(t)$ on a mesh would require $\Theta(n^{1/2}\log n)$ time in the worst case. By contrast, the mesh implementation of our algorithm has a worst-case running time of slightly more than $\Theta(n^{1/2})$ time. Since an $n$-processor hypercube can perform concurrent read and concurrent write operations in $\Theta(\log^2 n)$ time if sorting operations are based on bitonic sort [Batcher 1968], and in expected $\Theta(\log n)$ time if sorting operations are based on [Reif and Valiant 1987], direct simulation of the PRAM algorithm for describing $h(t)$ on a hypercube would require $O(\log^2 n)$ time and expected $\Theta(\log^2 n)$ time, respectively. However, hypercube implementations of our algorithm for describing $h(t)$ run in $O(\log^2 n)$ time.

The processor requirements of some of our algorithms will frequently depend on a certain function $\lambda(n,s)$ that represents an upper bound on the number of pieces of the minimum function, where $n$ is the number of moving point-objects in a dynamic system and $s$ is an upper bound on the degree of the polynomials used to describe trajectories of the points in the system. It is known that $\lambda(n,s)$ is not linear in $n$. However, the combined work of [Davenport and Schinzel 1965; Hart and Sharir 1986; Sharir 1987; Szemerédi 1974] has shown that the deviation of $\lambda(n,s)$ from linearity in $n$ is insignificant for reasonable values of $n$ (see Theorem 2.3).

The paper is organized as follows. Section 2 discusses the mesh and hypercube models of computation, states assumptions, defines terms that are used throughout the paper, presents a series of results concerning fundamental properties of the function $\lambda(n,s)$, and discusses data movement operations. In Section 3, we show how to construct the function $h(t)$ of Equation (1) efficiently. The subject of Section 4 is how various geometric properties change with time for a dynamic system. The properties studied include closest and farthest points from a given point, closest and farthest pairs, collision, membership in the set of extreme points of the convex hull, and several questions concerning shapes and sizes of containers into which a dynamic system will fit as a function of time. In Section 5, we examine from the steady-state standpoint the properties of closest and farthest points, closest and farthest pairs, description of the convex hull, diameter of the system, and description of a minimum-area rectangle that encloses the system. A summary is given in Section 6.

2. Preliminaries

2.1. Order Notation

The notations $O$, $\Theta$, and $\Omega$ will be used in this paper to mean, intuitively, "order at most," "order exactly," and "order at least," respectively. Let $T(n)$ and $f(n)$ be nonnegative functions defined for all positive integers $n$. Then $T(n)$ is
$O(f(n))$ if there are positive constants $N,C$ such that $n > N$ implies $T(n) < C f(n);$  
\[\Theta(f(n))\] if there are positive constants $N,C_0,C_1$ such that $n > N$ implies $C_0 f(n) < T(n) < C_1 f(n);$  
\[\Omega(f(n))\] if there are positive constants $N,C$ such that $n > N$ implies $T(n) > C f(n).$

These notions are fundamental to the analysis of algorithms, with $T(n)$ typically representing the running time of an algorithm that operates on input of size $n$. In particular, any operation that may be performed in a fixed number of steps is said to require $\Theta(1)$ (or constant) time.

2.2. Mesh-connected Computer

A two-dimensional mesh-connected computer (mesh) is a computer with multiple processors organized as a square lattice of processors. Each generic processor shares a bidirectional communication link with each adjacent processor in its row and in its column. A mesh of size $n$ has $n$ processors arranged as an $n^{1/2} \times n^{1/2}$ lattice (see Figure 1). We will use the terms processor and processing element (PE) interchangeably throughout the paper.

A mesh of size $n$ has communication diameter $\Theta(n^{1/2})$, meaning that the maximum number of communication links separating any pair of PEs in the mesh is $\Theta(n^{1/2})$. Therefore, if a problem on a mesh of size $n$ requires the possibility of two processors that are $\Theta(n^{1/2})$ communication links apart to communicate, and the running time of an algorithm to solve the problem is $O(n^{1/2})$, then the algorithm is optimal.

The PEs of a mesh of size $n$ are frequently numbered from 0 to $n-1$ so as to impose an order upon them. The common orderings of the PEs of a mesh include row-major, shuffled row-major, snake-like, and more recently, proximity [Miller and Stout 1989b]. The latter is based on the Peano-Hilbert scan curve [Koenderink and Van Doorn 1979; Lempel and Ziv 1986]. In this paper, we assume that the PEs are indexed via proximity order (see Figure 2). The properties of proximity order that are useful to us are the following.

1. In a mesh of size $n$, if $0 \leq i < n - 1$ then $\text{PE}_i$ and $\text{PE}_{i+1}$ are neighboring PEs.
2. A mesh may be recursively subdivided into submeshes such that each submesh contains consecutively indexed PEs.

Let $\Sigma$ be a nonempty subset of the processors of a mesh. We say $\Sigma$ is a string (of processors) of the mesh if and only if there are integers $i_0$ and $i_1$, $0 \leq i_0 \leq i_1 < n$, such that $\Sigma = \{\text{PE}_{i_0} \leq i \leq i_1\}$.

2.3. Hypercube Computer

A hypercube of size $n$, where $n$ is a nonnegative integral power of 2, has $n$ PEs indexed by the integers $\{0, \ldots, n - 1\}$. If we view each integer in the index range as a $(\log_2 n)$-bit string, two PEs are connected by a bidirectional communication link if and only if their indices differ in exactly one bit. See Figure 3.
Figure 1. A mesh computer of size $n$.

Figure 2a. Row-major.

Figure 2b. Shuffled row-major.

Figure 2c. Snake-like.

Figure 2d. Proximity.

Figure 2. Indexing schemes for a mesh of size 16.
Consecutively indexed PEs in a hypercube are not, in general, adjacent (see Figure 3). The usefulness of placing consecutive data in adjacent PEs during sorting makes it useful to relabel the PEs of a hypercube so that consecutive PEs in this order are adjacent in the hypercube. At the same time, we wish to be able to split the hypercube into subcubes so that the subcubes consist of consecutively labeled PEs. A commonly used method of ordering the PEs of a hypercube with these properties is the binary reflected Gray code [Reingold et al. 1977]. Such a code is a permutation $G_k$ of the integers $\{0,1,2,\ldots,2^k-1\}$, where $n = 2^k$ for some nonnegative integer $k$. For example, the following yields a binary reflected Gray code for all $k \geq 0$.

$$G_6(0) = 0.$$  

$$G_k(j) = \begin{cases} G_{k-1}(j) & \text{if } k > 0 \text{ and } 0 \leq j < 2^{k-1}; \\ 2^{k-1} + G_{k-1}(2^k - 1 - j) & \text{if } k > 0 \text{ and } 2^{k-1} \leq j < 2^k. \end{cases}$$

Throughout this paper, processors in a hypercube will be labeled not by node number, but according to a binary reflected Gray code ordering. A string of processors in a hypercube will be a nonempty set of consecutive processors according to Gray code ordering, that is, a set $\Sigma$ of PEs for which there are nonnegative integers $m$ and $n$ such that $m \leq n$ and $\Sigma = \{PE| m \leq j \leq n\}$ according to a fixed binary reflected Gray code ordering of the processors.

The communication diameter of a hypercube of size $n$ is $\log_2 n$. Therefore, if a problem on a hypercube of size $n$ requires the possibility of two nodes that are $\Theta(\log n)$ communication links apart to communicate, and the running time of an algorithm to solve the problem is $O(\log n)$, then the algorithm is optimal.

2.4. Motion Functions

Input to problems in this paper consists of descriptions of real-valued, or more generally, Euclidean vector-valued functions $f_0(t), f_1(t), \ldots, f_{n-1}(t)$ defined on the interval $[0, \infty)$. For convenience, we assume that no pair of these $f_i(0) \neq f_j(0)$ for $i \neq j$, $0 \leq i, j < n$. We refer to $f_i$ as a motion, and we may assume that each $f_i$ is a complex polynomial with real coefficients, and that $f_i(0) = 0$.

For many problems, these functions define respectively, in Euclidean $d$-dimensional space, a set of points that is a polynomial of degree no greater than $d$, and this is referred to as $k$-motion.

2.5. Pieces and the Function $\lambda$

Given a set of real-valued functions $F = \{f_0(t), \ldots, f_{n-1}(t)\}$ useful to construct the minimum function $h(t)$ generated by $F$ to consist over any interval $J \subseteq [0, \infty)$ such that $h(t) = \lambda(t)$, piece of the maximum function generated $h(t)$.

If $h_1(t)$ and $h_2(t)$ are real-valued functions by a family of functions $F$, then a piece of $F$ consists of a description of a function.

1. there exist $g_1, g_2 \in F$ such that $g = g_1 + g_2$.
2. $h_1 - h_2 = g$ identically on $I$.
3. $h_1 - h_2$ is not identically equal to $g$ on any subset of $J$.
convenience, we assume that no pair of the points have the same initial position. That is, \( f_i(0) \neq f_j(0) \) for \( i \neq j, 0 \leq i,j < n \). We assume that at the start of a problem, no processor contains a description of more than one of the functions \( f_0, \ldots, f_{n-1} \).

For many problems, these functions describe the motion of point-objects \( P_0, \ldots, P_{n-1} \), respectively, in Euclidean \( d \)-dimensional space. If every component of every function \( f_i \) is a polynomial of degree no greater than \( k \), then the collective movement of the points is referred to as \( k \)-motion.

2.5. Pieces and the Function \( \lambda \)

Given a set of real-valued functions \( F = \{f_0, \ldots, f_{n-1}\} \) defined on \([0, \infty)\), it will often be useful to construct the minimum function \( h(t) \) defined in Equation (1). Define a piece of the minimum function generated by \( F \) to consist of a description of some \( f_i \) and an interval \( I \subset [0, \infty) \) such that \( h = f_i \) identically on \( I \) and such that \( h \) is not identically equal to any \( f_j \) over any interval \( J \subset [0, \infty) \) such that \( I \) is properly contained in \( J \). (See Figure 4.) A piece of the maximum function generated by \( F \) is defined similarly.

If \( h_1(t) \) and \( h_2(t) \) are real-valued functions defined on \([0, \infty)\) whose pieces are generated by a family of functions \( F \), then a piece of \( h_1 - h_2 \) generated by differences of members of \( F \) consists of a description of a function \( g \) and an interval \( I \subset [0, \infty) \) such that

1. there exist \( g_1, g_2 \in F \) such that \( g = g_1 - g_2 \) identically on \([0, \infty)\),
2. \( h_1 - h_2 = g \) identically on \( I \), and
3. \( h_1 - h_2 \) is not identically equal to \( g \) on any interval \( J \subset [0, \infty) \) such that \( I \) is a proper subset of \( J \).

\[ y = g(t) \]
\[ y = h(t) \]
\[ y = f(t) \]

Figure 4. The pieces of \( \min\{f(t), g(t), h(t)\} \) are \((g(t), [0,a])\); \((h(t), [a,b])\); and \((f(t), [b,\infty])\).
Definition 2.1. [Atallah 1985] Let \(n\) and \(s\) be positive integers. Let \(S_n = \{a_1, \ldots, a_n\}\) be an alphabet of \(n\) distinct symbols. Let \(L_{n,s}\) be the set of strings over \(S_n\) that do not contain any \(a_i a_j\) as a substring and that do not contain as a subsequence of their characters any of the following forbidden sequences \(E_{ij}\), \(i \neq j\), defined by:

\[
E_{ij} = \begin{cases} 
  a_j a_i & \text{if } s = 1 \\
  E_{ij-1} a_i & \text{if } s = 2p \text{ for some positive integer } p \\
  E_{ij-1} a_i & \text{if } s = 2p + 1 \text{ for some positive integer } p.
\end{cases}
\]

Define \(\lambda(n, s)\) to be the maximum length of a string in \(L_{n,s}\). That is,

\[
\lambda(n, s) = \max \{|z| : z \in L_{n,s}\}.
\]

Notice that the presence of some \(E_{ij}\) 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, \(z = a_1 a_2 a_3 a_4 a_5 \notin L_{3,2}\) since \(z\) contains as a subsequence of its characters the sequence \(E_{12} = a_1 a_2 a_3 a_4\), which is forbidden to members of \(L_{3,2}\).

Observe that if \(f_1(t)\) and \(f_2(t)\) are distinct continuous real-valued functions of \([0, \infty)\) whose graphs have at most \(s\) points in common (for example, if \(f_1\) and \(f_2\) are distinct polynomials of degree at most \(s\)), then \(h(t) = \min\{f_1(t), f_2(t)\}\) has at most \(s + 1 = |E_{12}| - 1 = \lambda(2, s)\) pieces. A forbidden sequence represents a pair of functions whose graphs cross at least \(s + 1\) times, hence whose minimum function has at least \(s + 2 = |E_{12}|\) pieces. More generally, we have the following.

Lemma 2.2. [Atallah 1985] If \(F = \{f_0, \ldots, f_{n-1}\}\) is a set of continuous real-valued functions defined on \([0, \infty)\), no pair of which intersects more than \(s\) times, then \(h(t) = \min\{f_0(t), \ldots, f_{n-1}(t)\}\) has at most \(\lambda(n, s)\) pieces generated by \(F\), and this bound is best possible.

Lemma 2.2 shows that the maximum number of pieces, \(\lambda(n, s)\), of the minimum function generated by the set of functions \(\{f_0, \ldots, f_{n-1}\}\), is related to what [Davenport and Schinzel 1965] described as

the greatest length of a sequence with no immediate repetition, each term of which is one of 1, 2, \ldots, \(n\), and which contains no subsequence of the type

\[ababa\ldots \text{ with } s+1 \text{ terms and } a \neq b \text{ are chosen from } 1, 2, \ldots, n.\]

While [Davenport and Schinzel 1965] forbids subsequences with length \(s + 1\) of the type \(ababa\ldots\) with \(a \neq b\) and \(a, b \in \{1, 2, \ldots, n\}\), the subsequences we are forbidding have the same form but have length \(s + 2\).

Properties of \(\lambda(n, s)\) have been studied in several papers [Davenport and Schinzel 1965; Hart and Sharir 1986; Sharir 1987; Szemerédi 1974]. To describe the behavior of \(\lambda(n, s)\) we use the inverse Ackermann function \(\alpha(n)\), a description of which is given in [Hart and Sharir 1986]. It should be noted that \(\alpha(n)\) is a monotone nondecreasing function that grows to \(\infty\) extremely slowly. For example, [Hart and Sharir 1986] shows that

\[\alpha(n) \leq 4 \text{ for } n \leq 2^{2^2},\]

where the number of 2's in the tower is 65536,
and that if we denote $\log^{(1)} n = \log n$, and more generally, $\log^{(k+1)} n = \log(\log^{(k)} n)$ for integer $k > 0$, then

$$\alpha(n) = O(\log^{(j)} n) \text{ for all integer } j > 0.$$  

**Theorem 2.3.** The following results concerning the function $\lambda(n,s)$ are known.

- $\lambda(n,1) = n$ and $\lambda(n,2) = 2n - 1$ [Davenport and Schinzel 1965].
- $\lambda(n,3) = \Theta(n\alpha(n))$ [Hart and Sharir 1986].
- For $s \geq 3$, $\lambda(n,s) = \Omega(n\alpha(n))$ (this follows from the previous result and the fact that $\lambda$ is an increasing function of $s$), and
- For $s \geq 3$, $\lambda(n,s) = O(n^{\alpha(n)}\log r)$ [Sharir 1987].

For all problems considered in this paper that use the function $\lambda(n,s)$, the parameter $s$ will be a bounded integer. Under such circumstances, the above implies that for reasonable values of $n$, $\lambda(n,s)$ is essentially $\Theta(n)$.  

The next result gives a property that will be useful for bounding the number of processors in the algorithm associated with Theorem 3.2 for constructing the min function.

**Lemma 2.4** [Boxer and Miller 1989] For all positive integers $n$ and $s$, $2\lambda(n,s) \leq \lambda(2n,s)$.  

An interval is said to be nondegenerate if and only if it contains more than one point.  

Two intervals have a nondegenerate intersection if and only if their intersection contains a nondegenerate interval. If $p$ is a piece of a function $f$ and $q$ is a piece of a function $g$, we say $p$ and $q$ have nondegenerate intersection if and only if the interval of $p$ and the interval of $q$ have nondegenerate intersection. The next two results give useful bounds on the number of pieces in a combined function, and will be referred to in Section 4.

**Lemma 2.5.** [Boxer and Miller 1989] Let $f(t)$ and $g(t)$ be real-valued functions defined for all $t \geq 0$. Let $m$ and $n$ be positive integers. Suppose $f(t)$ has $m$ pieces and $g(t)$ has $n$ pieces. Then the pieces of $f(t)$ have, altogether, at most $m + n$ nondegenerate intersections with the pieces of $g(t)$.

**Lemma 2.6.** [Boxer and Miller 1989] Let $p$ and $s$ be positive integers. Let $f(t)$ and $g(t)$ be real-valued functions defined for all $t \geq 0$. 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 $s$. Assume that the pieces of $f(t)$ have $p$ nondegenerate intersections with the pieces of $g(t)$. Then the function $\min\{f(t),g(t)\}$ has at most $p(s + 1)$ pieces.

### 2.6 Data Movement Operations

Our algorithms make use of a variety of fundamental data movement operations. Brief descriptions of the most important of these are given below. See Table 1 for a summary.
of the running times of the operations, and [Miller and Stout 1989b] for descriptions of algorithms that implement these operations, analysis of running times, and further references. We assume data values are distributed among the PEs of a parallel machine so that no PE has more than $O(1)$ elements. We also assume that there are multiple strings in which the operations are to be performed in parallel, with the longest string consisting of $n$ PEs containing $m = O(n)$ data items $D = \{x_1, x_2, \ldots, x_m\}$. Note that the entire computer may be considered as a single string.

Semigroup computation Determine the result of applying a unit-time associative binary operation (i.e., a semigroup operation) to all elements of $D$. Examples of such operations include minimum, maximum, and sum.

Broadcast Send a copy of a single marked data item within each string to all PEs of the string.

Parallel prefix Assume $D$ is distributed $\Theta(1)$ items per processor in a string of size $n$ so that for $i < j$, if $x_i$ is stored in $PE_i$ and $x_j$ is stored in $PE_j$, then $i' \leq j'$. Also assume that for $i \in \{1, \ldots, m - 1\}$, if $x_i$ is stored in $PE_i$, then $x_{i+1}$ is stored in either $PE_i$ or $PE_{i+1}$. Let $*$ be an associative binary operation for which data items $x_1, \ldots, x_m$ are operands. For all $i \in \{1, 2, \ldots, m\}$, the $i$-th prefix is defined to be $p_i = x_1 * x_2 * \ldots * x_i$. The parallel prefix operation computes $p_i$, for all $i \in \{1, \ldots, m\}$, and stores $p_i$ in the PE that contains $x_i$. Parallel prefix is a powerful operation that can be used to find the minimum or maximum of a set of elements, broadcast values, compare data, and so forth.

Merging Given two ordered sequences of data stored in disjoint substrings $\Theta(1)$ items per PE, merge the data into a single ordered sequence stored $\Theta(1)$ items per PE.

Sorting Given a set of data with a linear order stored $\Theta(1)$ per PE in a string, the result of a sort operation is to redistribute the data within the string $\Theta(1)$ items per PE so that if $x_i \leq x_j$, then $x_i$ will be stored in some $PE_i$ and $x_j$ will be stored in some $PE_j$ such that $i' \leq j'$. Notice that the expected time to sort on the hypercube is faster [Reif and Valiant 1987] than the fastest known worst-case time to sort on the hypercube, which is based on the bitonic sort algorithm of [Batcher 1968]. (See [Thompson and Kung 1977] for a nice description of an optimal sorting algorithm for the mesh.)
Dynamic Computational Geometry on Meshes and Hypercubes

**Grouping** This operation is often performed when one set of ordered data needs to perform multiple simultaneous searches on another set of ordered data. It is typically accomplished within each string by sorting both sets of ordered data together, performing sort-based concurrent reads within strings to determine substrings representing intervals, and then performing a semigroup or parallel prefix operation within the substrings.

**3. Constructing the MIN Function**

The minimum function plays an important role in some of our algorithms. In this section, we show how a description of the minimum function may be constructed efficiently, under relatively mild restrictions, from descriptions of a set of real-valued functions.

The first result of this section shows how to obtain a description of the function \( \min \{ f(t), g(t) \} \). The reader should note that the algorithm we give to construct the min function can also be used to construct a description of the function that results from applying any of a variety of operations (e.g., max, sum, product) to a pair of real-valued functions.

If \( I \) is a subset of the domain of the function \( f(t) \), we denote by \( f|_I \) the restriction of \( f \) to \( I \). That is, \( f|_I \) is the function whose domain is \( I \) such that \( f|_I(t) = f(t) \) for all \( t \in I \).

**Lemma 3.1.** Let \( s \) be a nonnegative integer. Let \( \Phi \) be a family of real-valued functions defined on \( [0, \infty) \) with \( s \)-motion. Let \( f(t) \) and \( g(t) \) be real-valued functions defined on \( [0, \infty) \) by pieces generated by \( \Phi \). Suppose \( m \) is a positive integer such that the total number of pieces of \( f \) and \( g \) is \( m \). Suppose the pieces of \( f \) and the pieces of \( g \) are stored in an ordered fashion in disjoint strings of a mesh of size \( 4^{\lfloor \log m \rceil} \) or a hypercube of size \( 2^{\lceil \log m \rceil} \), at most one piece per PE. Then a description of the function \( h(t) = \min \{ f(t), g(t) \} \) can be constructed by the mesh in \( \Theta(m^{1/2}) \) time and by the hypercube in \( \Theta(\log m) \) time.

**Proof:** The algorithm is given in six steps.

1. In parallel, each PE containing a piece of \( f \) or a piece of \( g \) creates two records, Left and Right, each containing the following information:
   - A tag indicating whether the record represents a piece of \( f \) or a piece of \( g \).
   - A tag whose value is Left in the Left records, Right in the Right records.
   - A description of the piece.
   - An endpoint field, whose value is the left endpoint of the interval of the piece for the Left records, the right endpoint for the Right records.
   - An other-piece field, initially undefined.

2. Merge all of the records generated in Step 1 together with respect to the endpoint field. Ties should be broken in favor of a Right record.

3. A string of \( f \) is a string whose first PE contains a Left record of \( f \) and whose last PE contains a Right record of \( f \) such that no intermediate PE contains a record of \( f \). A string of \( g \) is defined analogously. Use a parallel prefix so that each Right record of \( f \) (respectively, \( g \)) finds the PE-index of its corresponding Left record of \( f \) (respectively, \( g \)). In parallel, the last PE of each string of \( f \) (respectively, \( g \)) broadcasts throughout its string a description of its piece of \( f \) (respectively, \( g \)), which is taken by each record of \( g \) (respectively, \( f \)) in the string as its other-piece field.
4. We now construct the subpieces determined by nondegenerate intersections of a piece of $f$ and a piece of $g$. All PEs act in parallel as follows.

If $PE_i$ contains a piece of $p$ of $f$, then $PE_i$ handles the leftmost and rightmost nondegenerate intersections of $p$ with pieces of $g$ by performing the following three steps, once for the record with tag Left and once for the record with tag Right.

(a) Compute the intersection, $I$, of the intervals of $p$ and the tag-record's other-piece.

(b) Determine the (at most $s$) solutions of the equation $f|_I(t) = g|_I(t)$.

(c) The roots found in (b) determine at most $s + 1$ closed nondegenerate subintervals of $I$ with disjoint interiors. For each such subinterval $J$, determine which of $f|_J$ and $g|_J$ is minimal by comparing $f(t)$ and $g(t)$, where $t$ is any interior point of $J$.

Let $p$ be a piece of $f$ and let $q$ be a piece of $g$ such that $p$ and $q$ have nondegenerate intersection and $q$ is neither the leftmost nor the rightmost piece of $g$ whose intersection with $p$ is nondegenerate. Then the interval of $q$ is contained in the interval of $p$. Hence, in the PE in which $q$ is stored, the Left record and Right record have identical other-piece fields. Such processors $PE_i$ perform the following two steps:

(a) Determine the (at most $s$) solutions of the equation $f|_J(t) = g|_J(t)$, where $J$ is the interval of the piece of $g$.

(b) The roots found in (a) determine at most $s + 1$ closed nondegenerate subintervals of $J$ with disjoint interiors. For each such subinterval $K$, determine which of $f|_K$ and $g|_K$ is minimal by comparing $f(t)$ and $g(t)$, where $t$ is any interior point of $K$.

Suppose $f$ has $u$ pieces generated by $F$ and $g$ has $v$ pieces generated by $F$. By Lemma 2.5, the intervals of the pieces of $f$ and the intervals of the pieces of $g$ have at most $u + v$ nondegenerate intersections. Since the pieces of $f$ and the pieces of $g$ were stored one per PE, there are at most $s + 1 = \Theta(1)$ subpieces per PE.

5. Each PE has at most $\Theta(1)$ subpieces obtained in the previous step by solving an equation of the form $f|_J(t) = g|_J(t)$. Since the solutions to the latter equation may not have been found in order, each PE now orders its $\Theta(1)$ subpieces from left to right. Since the pieces whose intersections generated the subpieces were already sorted, the subpieces are now sorted.

6. At this point, there may be adjacent subpieces with the same function $F(t)$. Such pairs should be combined into a single piece. That is, if there are pieces of the form $(F(t), [a, b])$ and $(F(t), [b, c])$, they must be combined as $(F(t), [a, c])$ (respectively, $(F(t), [a, \infty])$). Adjacent subpieces that have the same function may be combined by using a parallel prefix operation.

For the mesh: Steps 1, 4, and 5 require $\Theta(1)$ time. Merging (Step 2), broadcasting (Step 3), and parallel prefix (Steps 3 and 6) require $\Theta(m^{1/2})$ time. Therefore, the running time of the algorithm is $\Theta(m^{1/2})$.

For the hypercube: Steps 1, 4, and 5 require $\Theta(1)$ time. Merging (Step 2), broadcasting (Step 3), and parallel prefix (Steps 3 and 6) take $\Theta(\log m)$ time. Therefore, the running time of the algorithm is $\Theta(\log m)$.

Constructing the minimum function for a pair of functions, as described above, is part of a recursive algorithm for describing the function $h(t)$ of Equation (1). An efficient description of $h(t)$ is obtained by means of the algorithm that follows in Theorem 3.2. It should be noted that this algorithm is not limited to computing the min function of a set of functions.
With trivial modifications, it can be used to compute the result of applying to a set of \( n \) functions any commutative, associative operation that can be performed in \( O(1) \) time for a pair of single-piece functions.

Since the number of PEs in a mesh must be a power of 4, define
\[
\lambda_M(n, s) = 4 \lceil \log_4 \lambda(n, s) \rceil.
\]
Since the number of nodes in a hypercube must be a power of 2, define
\[
\lambda_H(n, s) = 2 \lceil \log_2 \lambda(n, s) \rceil.
\]
Observe that \( \lambda_M(n, s) \geq \lambda(n, s) \), \( \lambda_H(n, s) \geq \lambda(n, s) \), \( \lambda_M(n, s) = \Theta(\lambda(n, s)) \), and \( \lambda_H(n, s) = \Theta(\lambda(n, s)) \).

**Theorem 3.2.** Let \( s \) be a nonnegative integer. Let \( f_0, \ldots, f_{n-1} \) be real-valued functions defined on \([0, \infty)\) with \( s \)-motion. Suppose descriptions of \( f_0, \ldots, f_{n-1} \) are stored one per PE in a mesh with \( \lambda_M(n, s) \) PEs or in a hypercube with \( \lambda_H(n, s) \) PEs. Then the minimum function \( h(t) \) can be constructed by the mesh in \( \Theta(\lambda_M(n, s)) \) time and by the hypercube in \( \Theta(\log^2 n) \) time. At the end of the algorithm, the description of \( h(t) \) is given with the pieces ordered by their intervals, one piece per PE.

**Proof.** A general algorithm is given in three steps.

1. Split the descriptions of \( \{f_0, f_1, \ldots, f_{n-1}\} \) evenly among the processors, that is, redistribute the functions so that \( f_0, \ldots, f_{\lceil n/2 \rceil} \) are stored \( \Theta(1) \) per PE in \( PE_0, \ldots, PE_{\lambda_M(n, s)/2} \) (respectively, in \( PE_0, \ldots, PE_{\lambda_M(n, s)/2} \)) and \( f_{\lceil n/2 \rceil}, \ldots, f_{n-1} \) are stored \( \Theta(1) \) per PE in \( PE_{\lambda_M(n, s)/2}, \ldots, PE_{\lambda_M(n, s)-1} \) (respectively, \( PE_{\lambda_M(n, s)/2}, \ldots, PE_{\lambda_M(n, s)-1} \)).

2. Recursively, and in parallel, have the string with \( f_0, \ldots, f_{\lceil n/2 \rceil} \) construct the ordered pieces \( p_1, \ldots, p_n \) for
\[
h_1(t) = \min \{ f_0(t), \ldots, f_{\lceil n/2 \rceil}(t) \}
\]
generated by \( \{f_0, \ldots, f_{\lceil n/2 \rceil}\} \), while the string with \( f_{\lceil n/2 \rceil}, \ldots, f_{n-1} \) constructs the ordered pieces \( q_1, \ldots, q_{n-1} \) representing
\[
h_2(t) = \min \{ f_{\lceil n/2 \rceil}, \ldots, f_{n-1}(t) \}
\]
generated by \( \{f_{\lceil n/2 \rceil}, \ldots, f_{n-1}\} \). Since \( u, v \leq \lambda(\lceil n/2 \rceil, s) \), then from Lemma 2.4, each of the PEs is responsible for at most \( \Theta(1) \) pieces of a minimum function. At the end of this step, descriptions of the pieces \( \{p_1, \ldots, p_n\} \) and \( \{q_1, \ldots, q_{n-1}\} \) are ordered by their intervals in disjoint strings.

3. Construct \( h(t) = \min \{h_1(t), h_2(t)\} \) via the algorithm of Lemma 3.1.
For the mesh we have the following analysis. Step 1 requires $O(\lambda^2(n,s))$ time. Step 2 is a recursive call. Since we have $u + v \leq 2\lambda([n/2],s) \leq \lambda(n + 1,s)$, the latter inequality by Lemma 2.4, it follows from Lemma 3.1 and Theorem 2.3 that Step 3 requires $O(\lambda^2(n,s))$ time. Therefore, the running time of the algorithm satisfies the recurrence $T(n) = T(n/2) + O(\lambda^2(n,s))$. It follows from Lemma 2.4 that $T(n) = O(\lambda^2(n,s))$.

For the hypercube we have the following analysis. Step 1 requires $o(\log n)$ time. Step 2 is a recursive call. Step 3 requires $o(\log n)$ time, by Lemma 3.1. Therefore, the running time of the algorithm satisfies the recurrence $T(n) = T(n/2) + O(\log n)$, which is $O(\log^2 n)$.

The function $f(t)$ has a jump discontinuity at $t$ if and only if

- $t_0 > 0$, and
- there exists $\delta > 0$ such that either
  1. for all $t$ such that $0 < t < \delta$, $f(t_0 - t)$ is defined and $f(t_0 + t)$ is undefined, or
  2. for all $t$ such that $0 < t < \delta$, $f(t_0 - t)$ is undefined and $f(t_0 + t)$ is defined.

Intuitively, $f$ has a transition at the positive number $t_0$ if and only if $f(t)$ switches between being defined and undefined on either side of $t_0$. See Figure 5.

The next result is a generalization of Lemma 2.2.

**Lemma 3.3.** [Boxer and Miller 1989] Let $k$ be a positive integer. Let $f_0, \ldots, f_{n-1}$ be real-valued functions of time such that (a) every $f_j$ is continuous except for at most $p_j$ jump discontinuities, (b) every $f_j$ has at most $q_j$ transitions, where (c) $p_j + q_j \leq k$, and (d) no pair of distinct functions $f_i$ and $f_j$ intersect more than $s$ times. Then $h(t) = \min\{f_0(t), \ldots, f_{n-1}(t)\}$ has no more than $\lambda(n,s + 2k)$ pieces generated by $\{f_0, \ldots, f_{n-1}\}$.
The following theorem is used in Section 4.2 to develop an algorithm that determines when a query point \( P_0 \) in a dynamic system \( S = \{P_0,P_1,\ldots,P_{n-1}\} \) is an extreme point of the convex hull of \( S \). Its proof is virtually identical to that given for Theorem 3.2.

**Theorem 3.4.** Let \( k \) be a positive integer and let \( f_0, \ldots, f_{n-1} \) be as in Lemma 3.3. Assume also that (a) each \( f_i \) has a \( O(1) \) storage description, (b) each \( f_i(t) \) may be calculated in \( O(1) \) serial time if it is defined, and (c) \( i \neq j \) implies there are at most \( k \) solutions to \( f_i(t) = f_j(t) \), and they may be computed in \( O(1) \) serial time. Then a description of the function \( h(t) = \min\{f_0(t), \ldots, f_{n-1}(t)\} \) can be constructed in \( O(\lambda^{1/2}(n,s + 2k)) \) time by a mesh of size \( \lambda_{d}(n,s + 2k) \) so that the pieces are ordered, one per PE. A description of the function \( h(t) \) can be constructed in \( O(\log^2 n) \) time by a hypercube of size \( \lambda_{d}(n,s + 2k) \) so that the pieces are ordered, one per PE.

Analyses of our algorithms are given for worst-case running times on the mesh. Observe that \( \min\{f_0, \ldots, f_{n-1}\} \) may have less than \( \lambda(n,k) \) pieces, in which case it may be possible to use a submesh and obtain asymptotically faster running times (\( O(n^{1/2}) \) in the best case). The same is not true of the hypercube. Roughly, this is because \( \lambda^{1/2}(n,k) \neq O(n^{1/2}) \), while \( \log \lambda(n,s) = O(\log n) \).

4. Transient Behavior Computations

We apply the results of the previous section to a dynamic system \( S \) of points, showing how to determine geometric properties of \( S \). We assume in this section that point-objects have \( k \)-motion, for some fixed integer \( k \geq 0 \). A summary of the results presented in this section is given in Table 2.

### 4.1. Closest Points, Farthest Points, and Collision

Let \( R \) be a sequence of points closest, say, to \( P_0 \), listed in chronological order. For example, the first member of \( R \) is a closest point to \( P_0 \) at time \( t = 0 \), and the last member of \( R \) is a closest point to \( P_0 \) as \( t \) approaches infinity. Let \( R' \) be a sequence of farthest points from \( P_0 \), listed in chronological order.

**Table 2.** Transient behavior problems.

<table>
<thead>
<tr>
<th>Problem Description</th>
<th>Number of PEs</th>
<th>Mesh Running Time</th>
<th>Hypercube Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence of closest points to ( P_0 )</td>
<td>( \Theta(\lambda(n - 1,2k)) )</td>
<td>( \Theta(\lambda^{1/2}(n - 1,2k)) )</td>
<td>( \Theta(\log^2 n) )</td>
</tr>
<tr>
<td>Sorted times when ( P_0 ) collides</td>
<td>( \Theta(n) )</td>
<td>( \Theta(n^{1/2}) )</td>
<td>( \Theta(\log^2 n) ), expected ( \Theta(\log n) )</td>
</tr>
<tr>
<td>Ordered intervals when ( P_0 ) is hull vertex</td>
<td>( \Theta(\lambda(n,4k)) )</td>
<td>( \Theta(\lambda^{1/2}(n,4k)) )</td>
<td>( \Theta(\log^2 n) )</td>
</tr>
<tr>
<td>Sorted intervals when ( S \subset ) fixed-size hyper-rectangle</td>
<td>( \Theta(\lambda(n,k)) )</td>
<td>( \Theta(\lambda^{1/2}(n,k)) )</td>
<td>( \Theta(\log^2 n) )</td>
</tr>
<tr>
<td>Edgelength function for min iso-oriented hypercube ( \subset S )</td>
<td>( \Theta(\lambda(n,k)) )</td>
<td>( \Theta(\lambda^{1/2}(n,k)) )</td>
<td>( \Theta(\log^2 n) )</td>
</tr>
<tr>
<td>Smallest-ever iso-oriented hypercube ( \subset S )</td>
<td>( \Theta(\lambda(n,k)) )</td>
<td>( \Theta(\lambda^{1/2}(n,k)) )</td>
<td>( \Theta(\log^2 n) )</td>
</tr>
</tbody>
</table>
An algorithm for constructing \( R \) follows. Broadcast a description of function \( f_0 \) so that, without loss of generality, \( PE_j \) has descriptions of the distinct pairs \( (f_0, f_j) \), \( 0 < j < n \). Let \( d_0(t) \) be the Euclidean distance between points \( P_0 \) and \( P_j \) at time \( t \). In \( O(1) \) parallel time, each processor \( PE_j \) constructs the function \( d_0^j(t) \), a polynomial of degree \( \leq 2k \). Now construct the min function \( h(t) \) of the family of functions \( d_0^j(t) \) by the algorithm of Theorem 3.2. For each piece of \( h(t) \), a pair of points that yielded the piece corresponds to an element of \( R \). A similar algorithm may be used to construct \( R' \). This gives the following result:

**Theorem 4.1.** For a system of \( n \) points in \( d \)-dimensional space with \( k \)-motion, each of \( R \) and \( R' \) can be constructed on a mesh of size \( \lambda_d(n - 1, 2k) \) in \( \Theta(n^{1/2}(n - 1, 2k)) \) time and on a hypercube of size \( \lambda_d(n - 1, 2k) \) in \( \Theta(n \log^2 n) \) time.

Sometimes it is more important to determine whether or not two points collide rather than which neighbor is closest. Define points \( P_i \) and \( P_j \) to collide at time \( t \) if and only if \( f_j(t) = f_j(t) \). We observe that \( P_0 \) and \( P_j \) (\( j > 0 \)) collide if and only if \( d_0^j(t) = 0 \) has a solution for \( t > 0 \). Thus, an ordered list of collision times for \( P_0 \) can be produced by constructing the functions \( d_0^j \), solving the equations \( d_0^j(t) = 0 \) (positive roots only), and sorting the union of the solutions. This gives the following:

**Theorem 4.2.** Assume that a system of \( n \) points in \( d \)-dimensional space with \( k \)-motion is given. Then a chronological list of times at which \( P_0 \) collides with any other point of the system can be created in \( \Theta(n^{1/2}) \) time on a mesh of size \( 4^{1/\log n} \); in \( \Theta(n \log^2 n) \) time on a hypercube of size \( 2^{1/\log n} \); and in expected \( \Theta(n \log n) \) time on such a hypercube.

### 4.2. Convex Hull

The **convex hull** of a set of points \( S = \{P_0, \ldots, P_{n-1}\} \), denoted \( \text{hull}(S) \), is the smallest convex set containing \( S \). A point \( P_i \in S \) is an **extreme point** or vertex of \( \text{hull}(S) \) if \( P_i \notin \text{hull}(S - \{P_i\}) \). In this section, we develop a general parallel algorithm to generate a description of the intervals of time over which a given point \( P_0 \in S \) is an extreme point of \( \text{hull}(S) \). We also give an efficient implementation of the algorithm for the mesh and hypercube.

Assume \( k \)-motion in the plane. Let \( T_i(t) \) 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 \( P_0 \) to \( P_j \) at time \( t \). By convention, \( -\pi < T_i(t) \leq \pi \). Formally, if \( x_i(t), y_i(t) \), and \( y_j(t) \) are the \( x \) and \( y \) coordinates of the points \( P_i \) and \( P_j \), respectively, at time \( t \), then

\[
T_i(t) = \begin{cases} 
\pi/2 & \text{if } x_i(t) = x_j(t) \text{ and } y_i(t) < y_j(t) \\
-\pi/2 & \text{if } x_i(t) = x_j(t) \text{ and } y_i(t) > y_j(t) \\
\arctan\left(\frac{y_j(t)-y_i(t)}{x_j(t)-x_i(t)}\right) & \text{if } x_i(t) < x_j(t) \\
\arctan\left(\frac{y_j(t)-y_i(t)}{x_j(t)-x_i(t)}\right) + \pi & \text{if } x_i(t) > x_j(t) \text{ and } y_i(t) < y_j(t) \\
\arctan\left(\frac{y_j(t)-y_i(t)}{x_j(t)-x_i(t)}\right) - \pi & \text{if } x_i(t) > x_j(t) \text{ and } y_i(t) > y_j(t) \\
\text{undefined} & \text{if } x_i(t) = x_j(t) \text{ and } y_i(t) = y_j(t)
\end{cases}
\]

Define \( G_0(t) = \{ T_0(t) \} \) if undefined, and \( G_j(t) = \{ T_j(t) \} \) if undefined.

Define \( B_1(t) = \{ T_1(t) \} \) if undefined, and \( B_j(t) = \{ T_j(t) \} \) if undefined.

If at time \( t \), \( G_j(t) \) is undefined (respectively, \( B_j(t) \) is undefined), then \( c(t) \) or \( d(t) \) is undefined.

Define \( T = \{ T_i \} \) if undefined.

**Lemma 4.3.** [Atallah 1985; Boxer and Miller 1985] Given a set \( S \) of points, \( \text{hull}(S) \) is an extreme point of \( \text{hull}(S) \) at time \( t \) if and only if:

1. \( a(t) - d(t) \geq 2 \pi \), or
2. \( b(t) - c(t) \leq 2 \pi \), or
3. \( a(t) \) and \( b(t) \) are undefined, or
4. \( c(t) \) and \( d(t) \) are undefined.

The theorem that follows will determine \( P_0 \in S = \{P_0, \ldots, P_{n-1}\} \) is an extreme point of \( \text{hull}(S) \).

**Theorem 4.5.** Let \( S = \{P_0, \ldots, P_{n-1}\} \) be a set of \( n \) points in \( d \)-dimensional space with \( k \)-motion, and let \( \text{hull}(S) \) be the convex hull of \( S \). Let \( P_0 \) be a point such that the directed line segment from \( P_0 \) to \( P_j \) is parallel and similarly oriented to the directed line segment from \( P_0 \) to \( P_k \), and let \( x_j(t) \) be the \( x \)-coordinate of \( P_j \) at time \( t \). Observe that, if \( x_j(t) \) is undefined, then \( x_j(t) = x_k(t) \) at time \( t \).

Define \( \phi_j(t) = \frac{x_k(t) - x_j(t)}{y_k(t) - y_j(t)} \).

If at time \( t \), \( \phi_j(t) \) is undefined (respectively, \( \phi_j(t) \) is undefined), then \( z_j(t) \) is undefined (respectively, \( y_j(t) \) is undefined).

\[
|z_j(t) - z_k(t)||y_j(t) - y_k(t)| = |z_j(t)|.
\]
Define \( G_j(t) = \begin{cases} T_j(t) & \text{if } T_j(t) \geq 0 \\ \text{undefined} & \text{otherwise.} \end{cases} \)

Define \( B_j(t) = \begin{cases} T_j(t) & \text{if } T_j(t) < 0 \\ \text{undefined} & \text{otherwise.} \end{cases} \)

Define the functions \( a_i, b_i, c_i, \) and \( d_i \) as follows:

\[
a_i(t) = \min \{ G_j(t) | 0 < j < n, i \neq j, G_j(t) \text{ is defined} \},
\]

\[
b_i(t) = \max \{ G_j(t) | 0 < j < n, i \neq j, G_j(t) \text{ is defined} \},
\]

\[
c_i(t) = \min \{ B_j(t) | 0 < j < n, i \neq j, B_j(t) \text{ is defined} \},
\]

\[
d_i(t) = \max \{ B_j(t) | 0 < j < n, i \neq j, B_j(t) \text{ is defined} \}.
\]

If at time \( t, G_j(t) \) is undefined (respectively, \( B_j(t) \) is undefined) for all \( j, \) then \( a_i(t) \) and \( b_i(t) \) (respectively, \( c_i(t) \) and \( d_i(t) \)) are undefined.

Define \( T = \{ T_j | 0 < j < n \} \).

**Lemma 4.3.** [Atallah 1985; Boxer and Miller 1989] For a system of \( n \) points with \( k \)-motion, each of the functions \( a_0, b_0, c_0, \) and \( d_0 \) has at most \( \lambda(n, 4k) \) pieces generated by \( T. \)

**Lemma 4.4.** [Atallah 1985] Given a set \( S \) of \( n \) points moving in the plane, a point \( P_0 \) is an extreme point of \( \text{hull}(S) \) at time \( t \) if and only if:

1. \( a_i(t) - d_i(t) \geq \pi, \) or
2. \( b_i(t) - c_i(t) \leq \pi, \) or
3. \( a_i(t) \) and \( b_i(t) \) are undefined, or
4. \( c_i(t) \) and \( d_i(t) \) are undefined.

The theorem that follows will determine the intervals of time over which a given point \( P_0 = \{ P_0, \ldots, P_{n-1} \} \) is an extreme point of \( \text{hull}(S). \)

**Theorem 4.5.** 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 \( P_0 \) is an extreme point of \( \text{hull}(S) \) can be determined in \( \Theta(\lambda^{1/2}(n, 4k)) \) time on a mesh of size \( \lambda(n, 4k) \) and in \( \Theta(\log^2 n) \) time on a hypercube of size \( \lambda(n, 4k). \)

**Proof.** For each \( j, 0 \leq j < n, \) let \( x_j(t) \) be the \( x \)-coordinate of \( P_j \) at time \( t \) and let \( y_j(t) \) be the \( y \)-coordinate of \( P_j \) at time \( t. \) Observe that solving \( T_0(t) = T_{\text{on}}(t) \) means finding instants at which the directed line segment from \( P_0 \) to \( P_j \) and the directed line segment from \( P_0 \) to \( P_n \) are parallel and similarly oriented. Finding instants when the line segments are parallel requires solving

\[
[x_0(t) - x_j(t)](y_j(t) - y_0(t)) = [x_0(t) - x_n(t)](y_n(t) - y_0(t)),
\]

where \( x_0(t), y_0(t), x_n(t), y_n(t) \) are the \( x \)-coordinates and \( y \)-coordinates of \( P_0 \) and \( P_n \) at time \( t, \) respectively.
a polynomial equation of degree at most 2k. 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 \( T_{th}(t) = T_{ho}(t) \) can be solved by a single processor in \( \Theta(1) \) time.

Define

\[
A_0(t) = \begin{cases} 
1 & \text{if } a_0(t) - d_0(t) \geq \pi \\
0 & \text{otherwise}, 
\end{cases}
\]

\[
B_0(t) = \begin{cases} 
1 & \text{if } b_0(t) - c_0(t) \leq \pi \\
0 & \text{otherwise}, 
\end{cases}
\]

\[
C_0(t) = \begin{cases} 
1 & \text{if both } a_0(t) \text{ and } b_0(t) \text{ are undefined} \\
0 & \text{otherwise}, 
\end{cases}
\]

and

\[
D_0(t) = \begin{cases} 
1 & \text{if both } c_0(t) \text{ and } d_0(t) \text{ are undefined} \\
0 & \text{otherwise}. 
\end{cases}
\]

Our general algorithm is given below:

1. It is shown in [Boxer and Miller 1989] that each \( G_{ij} \) (similarly, each \( B_{ij} \)) has at most \( k \) values of \( t \) that yield jump discontinuities or transitions. Construct the functions \( a_{ij}(t), b_{ij}(t), c_{ij}(t), \) and \( d_{ij}(t) \).
2. It follows from Lemma 4.3 and Lemma 2.5 that each of \( a_{ij}(t) - d_{ij}(t) \) and \( b_{ij}(t) - c_{ij}(t) \) has no more than \( 2\lambda(n, 4k) = \Theta(\lambda(n, 4k)) \) pieces generated by differences of members of \( T \). Construct the ordered pieces of the functions \( a_{ij}(t) - d_{ij}(t) \) and \( b_{ij}(t) - c_{ij}(t) \). Similarly, construct the at most \( \Theta(\lambda(n, 4k)) \) ordered maximal intervals on which \( a_{ij}(t) \) and \( b_{ij}(t) \) are both undefined (respectively, on which \( c_{ij}(t) \) and \( d_{ij}(t) \) are both undefined).
3. Observe that if \( I_1 \) and \( I_2 \) are intervals of pieces of \( a_0 \) and \( d_0 \), respectively, where \( I = I_1 \cap I_2 \) is nondegenerate, then \( a_0|_I - d_0|_I = \pi \) only if the integers \( j \) and \( m \) associated with \( I_1 \) and \( I_2 \), respectively, where \( a_0|_I = T_{0j}, d_0|_I = T_{0m} \), are such that \( T_{0j}(t) - T_{0m}(t) = \pi \). Solving the latter means finding instants at which the directed line segment from \( P_0 \) to \( P_j \) and the directed line segment from \( P_0 \) to \( P_m \) are parallel and oppositely oriented. There are at most \( 2k \) instants when the line segments are parallel, and these may be found by a single PE in \( \Theta(1) \) time. Determining whether or not two parallel directed line segments are oppositely oriented may also be done in \( \Theta(1) \) serial time.

It follows from Lemma 2.6 that every piece of \( a_0(t) - d_0(t) \) generated by differences of members of \( T \) yields at most \( 2k + 1 \) pieces of \( A_0(t) \) generated by the set of constant functions \{0, 1\}. Therefore, \( A_0(t) \) has at most \( 2(2k + 1)2\lambda(n, 4k) = \Theta(\lambda(n, 4k)) \) pieces generated by \{0, 1\}. Similarly, \( B_0(t) \) has at most \( \Theta(\lambda(n, 4k)) \) pieces generated by \{0, 1\}.

Construct descriptions of the functions \( A_0(t) \) and \( B_0(t) \) by using the algorithm of Lemma 3.1. Similarly, construct the at most \( \Theta(\lambda(n, 4k)) \) ordered pieces generated by \{0, 1\} of each of the functions \( C_0(t) \) and \( D_0(t) \).
4. It follows from Lemma 2.6 that the function

\[ H_d(t) = \max \{ A_d(t), B_d(t), C_d(t), D_d(t) \} \]

has at most \( \Theta(\lambda(n, 4k)) \) pieces generated by the set of constant functions \( \{0, 1\} \). Describe \( H_d(t) \) via a fixed number of applications of the algorithm of Lemma 3.1. At the end of this step, the pieces of \( H_d(t) \) are stored \( \Theta(1) \) per PE and are ordered by their intervals.

5. Now Lemma 4.4 implies \( P_0 \) is an extreme point at time \( t \) if and only if \( H_d(t) = 1 \). Since the ordered pieces of \( H_d(t) \) are generated by the set of constant functions \( \{0, 1\} \), the desired sequence of intervals is now found in alternate pieces of \( H_d \). A parallel prefix operation may be used to pack this sequence of intervals into a string.

For the mesh: Step 1 requires \( \Theta(n^{1/2}(n, 4k)) \) time, by Theorem 3.4. Steps 2, 3, and 4 each require \( \Theta(n^{1/2}(n, 4k)) \) time, by Lemma 3.1. Step 5 requires \( \Theta(n^{1/2}(n, 4k)) \) time. Hence the running time of the algorithm is \( \Theta(n^{1/2}(n, 4k)) \).

For the hypercube: Step 1 requires \( \Theta(\log^2 n) \) time, by Theorem 3.4. Steps 2, 3, and 4 each require \( \Theta(\log n) \) time, by Lemma 3.1. Step 5 requires \( \Theta(\log n) \) time. Hence the running time of the algorithm is \( \Theta(\log^2 n) \).

4.3. Containment Problems

In this section, we address a variety of problems concerning shapes and sizes of containers into which a dynamic system of points will fit. We assume \( k \)-motion in \( d \)-dimensional space, for fixed \( k \) and \( d \).

The first problem of this section is concerned with constructing the ordered list \( J \) whose elements are the intervals of time during which the points \( P_0, \ldots, P_{n-1} \) can be enclosed within a rectilinear, iso-oriented hyper-rectangle (a \( d \)-dimensional analog of a box with sides parallel or perpendicular to each of the coordinate axes) of given fixed dimensions.

In addition to \( k \) and \( d \), input to the problem will include descriptions of the motions of points \( P_0, \ldots, P_{n-1} \) and the dimensions \( X_1, \ldots, X_d \) of a \( d \)-dimensional hyper-rectangle.

Upon termination of the algorithm, the ordered list \( J \) will be given.

Theorem 4.6. For a set of \( n \) points with \( k \)-motion in \( d \)-dimensional space, the ordered sequence \( J \) can be constructed on a mesh of size \( \lambda_{\alpha}(n, k) \) in \( \Theta(n^{1/2}(n, k)) \) time and on a hypercube of size \( \lambda_{\beta}(n, k) \) in \( \Theta(\log^2 n) \) time.

Proof: For \( 1 \leq i \leq d \), let \( p_i : \mathbb{R}^d \to \mathbb{R} \) be the \( i \)-th coordinate function. That is, for a point \( \mathbf{X} = (x_1, \ldots, x_d) \), \( p_i(\mathbf{X}) = x_i \). Note that for each \( i \) and \( j \), a description of \( p_i(f_j(t)) \) is stored in the PE containing a description of \( f_j \). Define

\[ m_i(t) = \min \{ p_i(f_0(t)), \ldots, p_i(f_{n-1}(t)) \} , \]

and

\[ M_i(t) = \max \{ p_i(f_0(t)), \ldots, p_i(f_{n-1}(t)) \} . \]
Our algorithm is given below.

1. Construct descriptions of the functions \( m_1(t), \ldots, m_d(t), M_1(t), \ldots, M_d(t) \) using the algorithm of Theorem 3.2 such that each of \( m_i \) and \( M_j \), \( 1 \leq i \leq d \), has at most \( \lambda(n,k) \) pieces generated by \( F_i = \{ p(f_0(t)), \ldots, p(f_{n-1}(t)) \} \).

2. Construct descriptions of all the functions \( D_i(t) = M_i(t) - m_i(t), 1 \leq i \leq d \), \( D_i(t) \) is the maximum separation in the \( i \)-th coordinate among the points \( \{ P_0, \ldots, P_{n-1} \} \) at time \( t \), by the algorithm of Lemma 3.1. It follows from Lemma 2.5 that each \( D_i(t) \), \( 1 \leq i \leq d \), has at most \( 2\lambda(n,k) \) pieces generated by differences of pairs of members of \( F_i \).

3. Let \( X_i \) be the length of the hyper-rectangle in the \( i \)-th coordinate, \( 1 \leq i \leq d \). Then for each \( i, 1 \leq i \leq d \), the function

\[
W_i(t) = \begin{cases} 
1 & \text{if } D_i(t) \leq X_i; \\
0 & \text{otherwise}
\end{cases}
\]

has at most \( 2(k + 1)\lambda(n,k) \) pieces generated by the set of constant functions \( \{0, 1\} \), by Lemma 2.6. Construct descriptions of \( W_1(t), \ldots, W_d(t) \) by the algorithm of Lemma 3.1.

4. Let \( C(t) = \min \{ W_1(t), \ldots, W_d(t) \} \). Notice that \( C(t) = 1 \) if and only if \( \{ P_0, \ldots, P_{n-1} \} \) will fit inside a hyper-rectangle of the given fixed dimensions at time \( t \). Describe \( C(t) \) from descriptions of the set of functions \( \{ W_1(t), \ldots, W_d(t) \} \). This is accomplished by performing \( \Theta(d) = \Theta(1) \) stages of the algorithm of Lemma 3.1, where at each stage, pairs of functions are combined. At the end of this step, the pieces of \( C(t) \) are ordered by their intervals.

5. At the end of the last step, the desired sequence \( J \) consists of the intervals for which \( C(t) = 1 \). These intervals are in alternate ordered pieces of \( C(t) \). A parallel prefix operation is used to pack the intervals into a string.

For the mesh: Step 1 requires \( \Theta(n) \) time by Theorem 3.2. Steps 2, 3, and 4 require \( \Theta(n) \) time, by Lemma 3.1. Step 5 (parallel prefix) requires \( \Theta(n) \) time.

For the hypercube: Step 1 requires \( \Theta(n) \) time. Steps 2, 3, and 4 require \( \Theta(n) \) time, by Lemma 3.1. Step 5 (parallel prefix) requires \( \Theta(n) \) time. Thus the algorithm requires \( \Theta(n) \) time.

We now consider the problem of finding the edgelength of a smallest iso-oriented rectilinear hypercube (an iso-oriented rectilinear hyper-rectangle with the same measurement in all coordinates) that will contain the set of points \( \{ P_0, \ldots, P_{n-1} \} \) as a function of time. Define

\[
\Phi = \{ p(f_j(t)) | 1 \leq i \leq d, 0 \leq j < n \}.
\]

**Theorem 4.7.** Assume a system of points \( S = \{ P_0, \ldots, P_{n-1} \} \) has \( k \)-motion in \( d \)-dimensional space. The function \( D(t) \), whose value at time \( t \) is the edgelength of the smallest iso-oriented rectilinear hypercube that will contain \( S \), has \( \Theta(n) \) pieces generated by differences of members of \( \Phi \). A description of \( D(t) \) can be constructed in an ordered fashion on a mesh of size \( \lambda_f(n,k) \) in \( \Theta(n \log n) \) time and on a hypercube of size \( \lambda_f(n,k) \) in \( \Theta(n \log n) \) time.
Proof. We give a general algorithm of two steps.

1. Let $D_1(t), \ldots, D_d(t)$ be as in Theorem 4.6. Construct descriptions of $D_1(t), \ldots, D_d(t)$ as in the algorithm of Theorem 4.6. It was shown in the proof of Theorem 4.6 that each of these functions has at most $2\lambda(n, k)$ pieces generated by differences of members of $\Phi$.

2. Since $D(t) = \max\{D_1(t), \ldots, D_d(t)\}$, observe $D(t)$ can be described from $D_1(t), \ldots, D_d(t)$ by performing $\Theta(\log d) = \Theta(1)$ stages of the algorithm of Lemma 3.1, where at each stage ordered pieces of pairs of functions are combined. If each function being combined has no more than $c\lambda(n, k)$ pieces, $c$ a constant, then the maximum of the two functions has no more than $2c(k + 1)\lambda(n, k)$ pieces, by Lemma 2.5 and Lemma 2.6.

Since each of the $\Theta(1)$ combine steps increases the number of pieces by no more than a constant factor, the number of pieces of $D(t)$ is $\Theta(\lambda(n, k))$.

Our claims concerning the running times follow from Theorem 4.6 and Lemma 3.1.

We now show how to determine a smallest iso-oriented rectilinear hypercube that can contain the set of points $S = \{P_0, \ldots, P_{n-1}\}$. First, construct a description of $D(t)$ by Theorem 4.7, where the function $D(t)$ has at most $\Theta(\lambda(n, k))$ pieces that are generated by differences of members of $\Phi$ and are distributed $\Theta(1)$ pieces per PE. Next, in parallel each PE determines the edgelength of a smallest iso-oriented rectilinear hypercube that will contain $S$ at some instant during its $\Theta(1)$ pieces’ intervals. Finally, compute the minimum of all the PEs’ minimas. Thus we have the following:

**Corollary 4.8.** Assume a system of $n$ points with $k$-motion in $d$-dimensional space. Let $D_{\min} = \min\{D(t) | t \geq 0\}$, where $D(t)$ is as in Theorem 4.7. Then $D_{\min}$ and a time $t_{\text{min}}$ at which $D(t_{\text{min}}) = D_{\min}$ can be computed in $\Theta(\lambda^{1/2}(n, k))$ time on a mesh of size $\lambda_M(n, k)$ and in $\Theta(\log^2 n)$ time on a hypercube of size $\lambda_H(n, k)$.

5. Steady-State Computations

We use the term steady-state to refer to conditions as $t$ (time) approaches infinity. For example, a steady-state nearest neighbor to $P_0$ is the point corresponding to the last member of a sequence $R$ described in Section 4.1. In this section, we give parallel algorithms for determining steady-state properties of dynamic systems, mostly in the plane. We also give implementations of these algorithms on the mesh and hypercube. Point-objects are assumed to have $k$-motion, for some fixed integer $k \geq 0$.

It should be noted that several of the steady-state problems investigated in this section may be naively solved by their analogs in the previous section. However, such an approach may require more time or memory than is necessary for a solution solely to the steady-state problem.

For example, a steady-state nearest neighbor to a given point $P_0$ may be found by considering the last piece of the solution to the nearest-neighbor-to-$P_0$, from Theorem 4.1, in $\Theta(\lambda^{1/2}(n - 1, 2s))$ time by a mesh with $\lambda_M(n - 1, 2s)$ PEs. By restricting our attention specifically to the steady-state behavior, we can solve the problem in $\Theta(n^{1/2})$ time using a mesh with $4^{\log^2 |s|}$ PEs (see Theorem 5.2, below).
**Table 3. Time complexity of solutions to steady-state problems for n-processor machines.**

<table>
<thead>
<tr>
<th>Problem Description</th>
<th>Mesh Time</th>
<th>Hypercube Time</th>
<th>Hypercube Expected Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest neighbor to (P_0)</td>
<td>(\Theta(n^{1/2}))</td>
<td>(\Theta(\log n))</td>
<td>(\Theta(\log n))</td>
</tr>
<tr>
<td>Closest pair</td>
<td>(\Theta(n^{1/2}))</td>
<td>(\Theta(\log^2 n))</td>
<td>(\Theta(\log n))</td>
</tr>
<tr>
<td>Ordered vertices of (hull(S))</td>
<td>(\Theta(n^{1/2}))</td>
<td>(\Theta(\log^2 n))</td>
<td>(\Theta(\log n))</td>
</tr>
<tr>
<td>Diameter function of convex polygon</td>
<td>(\Theta(n^{1/2}))</td>
<td>(\Theta(\log^2 n))</td>
<td>(\Theta(\log n))</td>
</tr>
<tr>
<td>Farthest pair</td>
<td>(\Theta(n^{1/2}))</td>
<td>(\Theta(\log^2 n))</td>
<td>(\Theta(\log n))</td>
</tr>
<tr>
<td>Minimal-area enclosing rectangle</td>
<td>(\Theta(n^{1/2}))</td>
<td>(\Theta(\log^2 n))</td>
<td>(\Theta(\log n))</td>
</tr>
</tbody>
</table>

**Table 4. Static algorithms used for n-processor machines.**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Model</th>
<th>Time Complexity</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closest Pair</td>
<td>Mesh</td>
<td>(\Theta(n^{1/2}))</td>
<td>[Miller and Stout 1989a]</td>
</tr>
<tr>
<td></td>
<td>Hypercube</td>
<td>(\Theta(\log^2 n), \Theta(\log n))</td>
<td>[Sanz and Cypher 1987]</td>
</tr>
<tr>
<td>Convex Hull</td>
<td>Mesh</td>
<td>(\Theta(n^{1/2}))</td>
<td>[Miller and Stout 1989a]</td>
</tr>
<tr>
<td></td>
<td>Hypercube</td>
<td>(\Theta(\log^2 n), \Theta(\log n))</td>
<td>[Miller and Stout 1988b]</td>
</tr>
<tr>
<td>Antipodal Vertices</td>
<td>Serial</td>
<td>(\Theta(n \log n))</td>
<td>[Shamos 1985]</td>
</tr>
<tr>
<td>Minimal Enclosing Rectangle</td>
<td>Hypercube</td>
<td>(\Theta(\log^2 n), \Theta(\log n))</td>
<td>[Miller and Stout 1988a]</td>
</tr>
</tbody>
</table>

A summary of the results presented in this section is given in Table 3. All algorithms are implemented on meshes and hypercubes with \(\Theta(n)\) PEs. In each case we assume there are \(n\) objects of interest. In Table 4, we list static algorithms used to solve steady-state problems in the plane.

### 5.1. Reduction Lemma

The following lemma, from [Boxer and Miller 1989], is useful for several of our steady-state algorithms, in that it will allow us to adapt efficient static solutions to some problems in dynamic computational geometry.

**Lemma 5.1.** Let \(k\) be a fixed nonnegative integer. Let \(f(t)\) and \(g(t)\) be polynomials of degree at most \(k\). Then the steady-state minimum of \(f(t)\) and \(g(t)\) can be determined in serial \(\Theta(1)\) time.

### 5.2. Closest and Farthest Points

In this section, we give parallel algorithms for determining a steady-state nearest neighbor, farthest neighbor, and closest pair. Implementations are presented for the mesh and hypercube.

The nearest (farthest) neighbor problem can be solved as follows. Assume \(PE_0\) stores a description of \(f_0\). To find the steady-state nearest neighbor to \(P_0\), broadcast the description of \(f_0\) to all PEs, have each \(PE_i, 1 \leq i \leq n - 1\), determine the function \(d^0_i(t)\), a polynomial of degree at most \(2k\), and then determine \(\text{Steady-State-min}(d^0_i(t), \ldots)\).
Proposition 5.2. Let \( k \) and \( d \) be fixed integers such that \( k \geq 0 \) and \( d > 0 \). Given a set of points \( \{P_0, \ldots, P_{n-1}\} \) with \( k \)-motion in \( d \)-dimensional space, a steady-state nearest (farthest) neighbor to a given point \( P_0 \) can be determined on a mesh of size \( 4^{\lfloor \log_2 n \rfloor} \) in \( O(n^{1/2}) \) time and on a hypercube of size \( 2^{\lfloor \log_2 n \rfloor} \) in \( \Theta(\log n) \) time.

Since the functions \( d_i(t) \) are polynomials of degree no more than \( 2k \), then by Lemma 5.1, steady-state comparisons of the (squares of) distances separating two pairs of points can be done in serial \( \Theta(1) \) time. Thus the steady-state closest pair problem is transformed to the problem of finding a closest pair for a system of static points. It follows that we may apply the algorithm of [Miller and Stout 1989a] for a mesh of size \( 4^{\lfloor \log_2 n \rfloor} \) that identifies a closest pair for a set of \( n \) planar points in \( \Theta(n^{1/2}) \) time. Similarly, we may apply the algorithm of [Sanz and Cypher 1987] for a hypercube of size \( 2^{\lfloor \log_2 n \rfloor} \) that identifies a closest pair for a set of \( n \) planar points in \( \Theta(\log^2 n) \) time and in expected \( \Theta(\log n) \) time.

This gives the following:

Proposition 5.3. For a system of \( n \) points with \( k \)-motion in the plane, a steady-state closest pair can be identified by a mesh of size \( 4^{\lfloor \log_2 n \rfloor} \) in \( O(n^{1/2}) \) time; by a hypercube of size \( 2^{\lfloor \log_2 n \rfloor} \) in \( \Theta(\log^2 n) \) time; and by a hypercube of size \( 2^{\lfloor \log_2 n \rfloor} \) in expected \( \Theta(\log n) \) time.

5.3. Convex Hull

The convex hull of a set of point \( S = \{P_0, \ldots, P_{n-1}\} \) at time \( t \) is specified by giving the extreme points of \( \text{hull}(S) \) at time \( t \). In this section, we give an algorithm for describing the steady-state \( \text{hull}(S) \) of a planar system. Later, we will show that steady-state \( \text{hull}(S) \) can be used to identify a steady-state farthest pair of \( S \) and a steady-state minimal-area rectangle enclosing all the points of \( S \).

Proposition 5.4. For a system \( S \) of \( n \) points with \( k \)-motion in the plane, the steady-state \( \text{hull}(S) \) can be constructed by a mesh of size \( 4^{\lfloor \log_2 n \rfloor} \) in \( \Theta(n^{1/2}) \) time; by a hypercube of size \( 2^{\lfloor \log_2 n \rfloor} \) in \( \Theta(\log^2 n) \) time; and by a hypercube of size \( 2^{\lfloor \log_2 n \rfloor} \) in expected \( \Theta(\log n) \) time.

Proof. The algorithms of [Miller and Stout 1989b, 1989a] for constructing \( \text{hull}(S) \) are based on the relative positions of points, which, for any pair known to a single PE, may be determined in \( \Theta(1) \) serial time by Lemma 5.1. Thus the problem is transformed to that of constructing \( \text{hull}(S) \) for a system \( S \) of \( n \) static planar points on a mesh, which is accomplished by the algorithms cited in \( \Theta(n^{1/2}) \) time for the mesh, in \( \Theta(\log^2 n) \) time and in expected \( \Theta(\log n) \) time for the hypercube.

We note that the algorithms associated with Proposition 5.4 contain optimal solutions (for the mesh; expected optimal for the hypercube) to the question of whether or not a
query point $P_0$ is an extreme point of the steady-state hull($S$). If one is only interested in the latter question, another optimal solution may be obtained by modifying the algorithm used for Theorem 4.5.

5.4. Diameter, Farthest Pair, and Smallest Enclosing Rectangle

In this section, we give solutions to the steady-state diameter, farthest pair, and smallest enclosing rectangle problems. Our solutions are based on using Lemma 5.1 to show that the steady-state situation can be reduced to the static situation for these problems, and then adapting efficient static algorithms from [Miller and Stout 1988a, 1988b, 1989a, 1989b] to solve the problems. Details of the algorithms are given in this section for clarity.

Let $C$ be a convex set in the plane. A line of support of $C$ is a line $L$ that meets $C$ at least at one point such that $C$ does not intersect both of the complementary domains of $L$ in the plane. A pair of vertices $v$ and $w$ of $C$ are antipodal if there are distinct parallel lines of support of $C$, one of which passes through $v$ and the other of which passes through $w$. (See Figure 6a). The diameter of $C$ is the greatest distance separating members of pairs of antipodal extreme points of $C$ [Yaglom and Botyanskii 1961].

A cyclic ordering of the extreme points of $C$ induces a direction on each edge of $C$. Therefore, the edges may be thought of as rays whose endpoints are at the origin. This induces a natural mapping of extreme points of $C$ to sectors of the plane determined by the edge rays, where an extreme point is mapped to the sector interior to the angle formed by the rays representing the edges of the polygon incident to the extreme point. (See Figure 6b). To find a pair of antipodal extreme points corresponding to some line of support $L$ of $C$, consider a parallel translation $L'$ of $L$ so that $L'$ passes through the origin (of the ray diagram). The sectors through which $L'$ passes correspond to points of an antipodal pair [Shamos 1975].

Lemma 5.5. Let $S = \{P_0, \ldots, P_{n-1}\}$ be the set of distinct extreme points of a convex polygon $C$. For all edges $e$ of $C$, a vertex of $C$ on a line of support of $C$ that is parallel to and disjoint from $e$ may be determined in $\Theta(n^{1/2})$ time by a mesh of size $4^{\lceil \log n \rceil}$; in $\Theta(\log^2 n)$ time by a hypercube of size $2^{\lceil \log n \rceil}$; and in expected $\Theta(\log n)$ time by a hypercube of size $2^{\lceil \log n \rceil}$. Further, at the end of the algorithm, every pair of antipodal vertices of $C$ is represented in at least one PE, and no PE has more than four antipodal pairs of vertices.

Proof. Our algorithm is based on [Shamos 1975].

1. Broadcast $P_0$ to all PEs.
2. In parallel, each PE computes its point's angle with respect to $P_0$ as the origin.
3. Sort the extreme points into counterclockwise order, with respect to the angles determined above. Relabel the extreme points according to the counterclockwise order so that extreme point $P_{i-1}$ precedes the extreme point $P_i$ in the ordering of the extreme points. Thus $P_i$ is stored in $PE_i$.
4. In parallel, all processors $PE_i$ inform $PE_{i+1}$ ($PE_{n-1}$ informs $PE_0$) of the coordinates of $P_i$. Similarly, $PE_i$ informs $PE_{i-1}$ ($PE_0$ informs $PE_{n-1}$) of the coordinates of $P_i$. Thus, each PE has the coordinates of its point and of its two (cyclic) neighbors' points.
5. In parallel, each $PE_i$ computes the angle between the edge from $P_{i-1}$ to $P_i$ and a positively directed horizontal ray with vertex $P_{i-1}$, and the angle between the edge from $P_i$ to $P_{i+1}$ and a positively directed horizontal ray with vertex $P_i$. These angles determine the sector of $P_i$ in the plane. (See Figure 6).

6. In parallel, all processors $PE_j$ do the following. $PE_i$ is responsible for the edge $e_i$ from $P_{i-1}$ to $P_i$. Consider a line $L$ through the origin containing the ray $R$ representing $e_i$.
Using a grouping operation based on the angles computed in the previous step, determine the sector(s) containing \(-R\), the ray collinear with \(R\) with opposite direction. This sector (pair of sectors if \(-R\) coincides with an edge-ray) corresponds to a vertex of \(C\) (pair of vertices of \(C\)) antipodal to both \(P_{i-1}\) and \(P_i\), hence on a line of support parallel to and disjoint from \(e_i\).

Since each PE discovered at most two sectors for its edge, each PE has at most four antipodal pairs of vertices. To show our algorithm finds each antipodal pair, we argue as follows. Suppose \(P_i\) and \(P_j\) form an antipodal pair. Then there is a line \(L\) through the origin that falls in the sectors of \(P_i\) and \(P_j\). Rotate \(L\) clockwise until \(L\) contains a boundary ray of one of the sectors of \(P_i\) or \(P_j\). Without loss of generality, \(L\) contains a boundary ray of the sector of \(P_i\). Since \(L\) was rotated clockwise, this boundary ray represents \(e_i\). Its opposite ray is contained in the sector of \(P_j\). Therefore \(PE_i\) discovers the antipodal pair \(\{P_i, P_j\}\) in Step 6.

For the mesh: Step 1 requires \(\Theta(n^{1/2})\) time. Step 2 requires \(\Theta(1)\) time. Step 3 requires \(\Theta(n^{1/2})\) time. The communications in Step 4 require \(\Theta(n^{1/2})\) time. Step 5 requires \(\Theta(1)\) time. Step 6 requires \(\Theta(n^{1/2})\) time. Therefore, the running time of the algorithm is \(\Theta(n^{1/2})\).

For the hypercube: Step 1 requires \(\Theta(\log n)\) time. Step 2 requires \(\Theta(1)\) time. Step 3 requires \(\Theta(\log^2 n)\) time, expected \(\Theta(\log n)\) time. The communications in Step 4 require \(\Theta(\log n)\) time. Step 5 requires \(\Theta(1)\) time. Step 6 requires \(\Theta(\log^2 n)\) time and expected \(\Theta(\log n)\) time. Therefore, the running time of the algorithm is \(\Theta(\log^2 n)\), expected \(\Theta(\log n)\).

The farthest pair and diameter problems are related to the convex hull problem. [Shamos 1975] observes that a farthest pair of points in a (static or dynamic) system must be a pair of extreme points of the system. Our solutions to the farthest pair and diameter problems are patterned after a serial algorithm given in [Shamos 1975] and techniques for developing parallel algorithms for static systems that are discussed in [Miller and Stout 1989b]. An algorithm for a static system \((k = 0)\) can be constructed as follows. First, find all the antipodal pairs of vertices, so that each processor contains at most four antipodal pairs, by the algorithm associated with Lemma 5.5. Next, every processor \(PE_i\) determines the maximum of the squares of the distances between antipodal pairs in \(PE_i\). Finally, the global maximum is determined, where an antipodal pair associated with it is recorded. Notice that this maximum is the square of the diameter of \(C\), and the noted antipodal pair is a farthest pair. For the dynamic case \((k > 0)\) we observe that Lemma 5.1 reduces the algorithm above to the static case. This gives the following:

**Proposition 5.6.** Let \(k\) be a fixed nonnegative integer. Let \(S = \{P_0, \ldots, P_{n-1}\}\) be a set of point-objects moving in the plane with \(k\)-motion such that when steady-state is reached, \(S\) is the set of distinct extreme points of a convex polygon \(C\). The diameter function of \(C\) may be determined in \(\Theta(n^{1/2})\) time by a mesh of size \(4^\lceil \log_2 n \rceil\); in \(\Theta(\log^2 n)\) time by a hypercube of size \(2^{\lceil \log_2 n \rceil}\); and in expected \(\Theta(\log n)\) time by a hypercube of size \(2^{\lceil \log_2 n \rceil}\).

The general problem of finding a steady-state farthest pair in a system \(S\) of point-objects with \(k\)-motion may be solved by determining the extreme points of the steady-state hull(\(S\)), then determining the steady-state diameter function of hull(\(S\)), noting a pair of elements of \(S\) that yields this diameter function. Hence the following is an immediate consequence of Proposition 5.4 and Proposition 5.6.
Corollary 5.7. For a set $S$ of $n$ points with $k$-motion in the plane, a steady-state farthest pair can be determined by a mesh of size $4\lfloor \log_2 n \rfloor$ in $O(n^{1/2})$ time, by a hypercube of size $2^\lceil \log_2 n \rceil$ in $O(\log^2 n)$ time, and by a hypercube of size $2^\lceil \log_2 n \rceil$ in expected $O(\log n)$ time.

The next problem we consider is that of constructing a description of a steady-state minimal-area rectangle enclosing all the point-objects of a planar system in $k$-motion. An implementation of the algorithm is given in Corollary 5.9 that is optimal for the mesh, and expected time optimal for the hypercube. This algorithm exploits the result given below in Theorem 5.8 to determine a smallest enclosing rectangle of a convex planar set of point-objects with $k$-motion. It should be noted that an algorithm from [Miller and Stout 1989b] for determining the smallest enclosing rectangle of a static convex planar set, with imple-mentation from [Miller and Stout 1989a] for the mesh and from [Miller and Stout 1988a] for the hypercube, are used in Theorem 5.8.

Theorem 5.8. Let $k$ be a fixed nonnegative integer. Let $S = \{P_0, P_1, \ldots, P_{n-1}\}$ be a system of point-objects in $k$-motion such that, in steady-state, $S$ is the set of distinct extreme points of a convex polygon $C$. A rectangle of minimum area enclosing all the points of $C$ may be determined by a mesh of size $4\lfloor \log_2 n \rfloor$ in $O(n^{1/2})$ time; by a hypercube of size $2^\lceil \log_2 n \rceil$ in $O(\log^2 n)$ time; and by a hypercube of size $2^\lceil \log_2 n \rceil$ in expected $O(\log n)$ time.

Proof. A minimal-area rectangle enclosing all the points of $C$ must have one side collinear with an edge of $C$, and each of the other sides of the rectangle must pass through an extreme point of $C$. Hence, for each edge $e$ we determine a minimal-area rectangle $R_e$ enclosing all the points of $C$ such that an edge of $R_e$ contains $e$, then compute $\min\{\text{area}(R_e) : e \text{ is an edge of } C\}$ and note which $R_e$ yields the minimum. The algorithm follows:

1. Following the algorithm of Lemma 5.5, create descriptions of all the edges of $C$, which are used to divide the plane into sectors, and, for each edge $e$, describe a line of support $S_e$ of $C$ such that $S_e$ is parallel to and disjoint from $e$ and $S_e$ contains a vertex of $C$ antipodal to each of the vertices of $e$. Since $S$ is a system in $k$-motion, $S_e$ is described as a function of time by a polynomial equation of degree at most $2k$.

2. In parallel, for each edge, $e$, determine the line $L_e$ through the origin such that $L_e \perp S_e$. $L_e$ is described as a function of time by a polynomial equation of degree at most $k + 1$.

3. Each $L_e$ meets the sectors corresponding to an antipodal pair of vertices $v_e$ and $v_{e'}$. In parallel, determine such a pair via a grouping operation based on the angles of the edges determined in Step 1.

4. In parallel, for each edge $e$ describe lines $M_{e_1}$ and $M_{e_2}$ containing $v_e$ and $v_{e'}$, respectively, such that $M_{e_1}$ and $M_{e_2}$ are parallel to $L_e$. The lines $M_{e_1}$ and $M_{e_2}$ are described by polynomial equations of time of degree at most $2k$.

5. For each edge $e$, let $E_e$ be the line containing $e$. It is clear that the rectangle $R_e$ determined by the lines $E_e$, $S_e$, $M_{e_1}$, and $M_{e_2}$ is the minimal-area rectangle with an edge containing $e$ such that the rectangle contains all the points of $S$. In parallel, compute $A_e(t)$, the square of the area function of $R_e$. $A_e(t)$ is a polynomial function of degree at most $8k$.

6. Determine the steady-state-min $\{A_e(t) : e \text{ is an edge of } C\}$, and note a rectangle corresponding to the minimum. This is the smallest rectangle enclosing the elements of $S$. 

DYNAMIC COMPUTATIONAL GEOMETRY ON MESHES AND HYPER Cubes
For the mesh: By Lemma 5.5, Step 1 takes \( \Theta(n^{1/2}) \) time. Step 2 requires \( \Theta(1) \). Step 3 requires \( \Theta(n^{1/2}) \) time. Steps 4 and 5 require \( \Theta(1) \) time. Step 6 requires \( \Theta(n^{1/2}) \) time, by Lemma 5.1. Hence the algorithm uses \( \Theta(n^{1/2}) \) time.

For the hypercube: By Lemma 5.5, Step 1 takes \( \Theta(\log^2 n) \) time, expected \( \Theta(\log n) \) time. Step 2 requires \( \Theta(1) \) time. Step 3 requires \( \Theta(\log^2 n) \) time, expected \( \Theta(\log n) \) time. Steps 4 and 5 require \( \Theta(1) \) time. Step 6 requires \( \Theta(\log n) \) time, by Lemma 5.1. Hence the algorithm uses \( \Theta(\log^2 n) \) time, expected \( \Theta(\log n) \) time.

Given a set \( S = \{P_0, P_1, \ldots, P_{n-1}\} \) of point-objects in \( k \)-motion, the general problem of describing a steady-state minimal-area rectangle enclosing \( S \) may be solved by describing the steady-state hull(\( S \)) and then describing a minimal-area rectangle enclosing steady-state hull(\( S \)). Hence the following is an immediate consequence of Proposition 5.4 and Theorem 5.8.

**Corollary 5.9.** Let \( k \) be a fixed nonnegative integer. Let \( S = \{P_0, \ldots, P_{n-1}\} \) be a system of point-objects in \( k \)-motion. Then a description of a steady-state minimal-area rectangle enclosing all the points of \( S \) can be given by a mesh of size \( 4^{\log^2 n} \) in \( \Theta(n^{1/2}) \) time; by a hypercube of size \( 2^{\log n} \) in \( \Theta(\log^2 n) \) time; and by a hypercube of size \( 2^{\log n} \) in expected \( \Theta(\log n) \) time.

6. Further Remarks

We have considered problems of determining geometric properties of points moving in Euclidean space. The motion considered includes trajectories whose coordinate functions are polynomials of bounded degree. In fact, many of the results presented in this paper are valid for a somewhat more general setting. Given functions \( f \) and \( g \), the properties necessary for most of our algorithms are the following:

1. \( f \) is continuous on the interval \([0, \infty)\);
2. \( f \) has a \( \Theta(1) \) storage description;
3. for all \( t \geq 0 \), \( f(t) \) can be evaluated in \( \Theta(1) \) serial time; and
4. there is an integer \( k \geq 0 \) such that if \( f \neq g \), then there are at most \( k \) solutions to the equation \( f(t) = g(t) \), all of which may be found in \( \Theta(1) \) serial time.

Notice that for some of the problems we consider, these properties are required for the coordinates of the trajectories, while for other problems, the functions that must have these properties, such as (squares of) distances, are derived from the trajectory functions.

This paper presents parallel algorithms described in terms of global operations and fundamental parallel paradigms for solving problems in dynamic computational geometry. We also show that many steady-state problems can be reduced to static problems, from which efficient solutions can be determined. Implementations and analyses of running times are given for massively parallel mesh and hypercube connected computers and can be applied to commercially available fine-grained supercomputers with these topologies. Problems considered include both transient-behavior and steady-state versions of closest pair, farthest pair, and convex hull. Also considered were questions of collision, containment, and diameter.
Running times of all our algorithms are asymptotically optimal for the mesh in the worst case. We have indicated how our transient behavior algorithms may, in the best case, run faster than indicated by our analysis on the mesh. All of our algorithms are within a logarithmic factor of being optimal for the hypercube. Indeed, several of our algorithms have asymptotically optimal or expected optimal running times for the hypercube. Our hypercube implementations for dynamic problems produce sorted output in no more time than the fastest worst-case deterministic sorting algorithm currently known. However, as these algorithms do not contain explicit global sort steps, the question of whether these algorithms are optimal remains open. For several of the algorithms, the memory bound is almost-linear, while for others it is linear.

The techniques presented in this paper can be adapted to related problems. For example, by using a mesh of size $\lambda_d(n(n - 1)/2, 2k)$ (respectively, a hypercube of size $\lambda_d(n(n - 1)/2, 2k)$), trivial modifications to the algorithm of Theorem 4.1 give a sequence of closest or farthest pairs for a system of $n$ points with $k$-motion in $d$-dimensional space in $O(\lambda^{1/2}(n(n - 1)/2, 2k))$ time for the mesh and in $O(\log^2 n)$ time for the hypercube. However, we leave as an open problem whether the same or better running times can be obtained using $O(\lambda(n, 2k))$ PEs.

Simulation of the PRAM algorithms of [Chandran and Mount 1989] or of the serial algorithms of [Atallah 1985] by a mesh or by a hypercube would yield algorithms that are not as efficient as those presented in this paper. For example, the PRAM algorithm of [Chandran and Mount 1989] for describing the function

$$h(t) = \min\{f_0(t), f_1(t), \ldots, f_{p-1}(t)\}$$

has a running time of $O(\log n)$. Since a mesh of size $n$ can perform concurrent read and concurrent write operations in $O(n^{1/2})$ time, it follows that direct simulation of the PRAM algorithm would yield a mesh algorithm for describing $h(t)$ whose running time is $O(n^{1/2}\log n)$, as opposed to the running time of $O(\lambda^{1/2}(n, k))$ of the mesh algorithm presented in this paper. Similarly, since a hypercube of size $n$ can perform concurrent read and concurrent write operations in $O(\log^2 n)$ time based on bitonic sorting and in expected $O(\log n)$ time based on [Reif and Valiant 1987], it follows that direct simulation of the PRAM algorithm would yield hypercube algorithms for describing $h(t)$ whose running times are $O(\log^3 n)$ based on bitonic sort and expected $O(\log^2 n)$ based on [Reif and Valiant 1987], as opposed to $O(\log^2 n)$ running times for the algorithms presented in this paper.

7. Acknowledgments

We are grateful to Quentin Stout for a suggestion that enabled us to improve substantially the hypercube implementations of some of our algorithms and to the anonymous referees for suggestions that greatly improved the presentation of this paper.

References


Intel Corp. 1986. IPSC System Overview (Jan.).


