Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers *

Frank Dehne§ Andreas Fabri† Andrew Rau-Chaplin§

†INRIA
BP 93
06902 Sophia Antipolis Cedex,
France

§School of Computer Science
Carleton University,
Ottawa, Canada K1S 5B6

1 Introduction

Parallel Computational Geometry is concerned with solving some given geometric problem of size n on a parallel computer with p processors (e.g., a PRAM, mesh, or hypercube multiprocessor) in time $T_{parallel}$. We call the parallel solution *optimal*, if $T_{parallel} = O(\frac{T_{sequential}}{p})$, where $T_{sequential}$ is the sequential time complexity of the problem. Theoretical work for Parallel Computational Geometry has so far focussed on the case $\frac{n}{p} = O(1)$, also referred to as the *fine grained* case. However, for parallel geometric algorithms to be relevant in practice, such algorithms must be *scalable*, that is, they must be applicable and efficient for a wide range of ratios $\frac{n}{p}$. The design of such scalable algorithms is also listed as a major goal in the recent "Grand Challenges" report [6].

Yet, only little theoretical work has been done for designing scalable parallel algorithms for Computational Geometry. The first and, to our knowledge, only previous theoretical paper to address this problem was [1]. The model considered there was a host machine with $O(n)$ memory attached to a systolic array of size p with $O(1)$ memory per processors. This model suffers however from fact that data has to be frequently swapped between the host and the systolic array, and this "I/O bottleneck" is the main factor determining the computation time. The architectures of most existing multicomputers (e.g. the Intel Paragon, Intel iPSC/860, and CM-5) are quite different. They consist of a set of p *state-of-the-art* processors (e.g. SPARC processors), each with considerable local memory, connected by some interconnection network (e.g. mesh, hypercube, fat tree). These machines are usually *coarse grained*, i.e. the size of each local memory is considerably larger than $O(1)$. In order to minimize the I/O bottleneck, the entire data set for a given problem is immediately loaded into the local memories and remains there until the problem is solved.

The *Coarse Grained Multicomputer* model, or $CGM(n,p)$ for short, is a set of p processors with $O(\frac{n}{p})$ local memory each, connected by some arbitrary interconnection network. Our model is *coarse grained*, as the size $O(\frac{n}{p})$ of each local memory is defined to be considerably larger than $O(1)$, e.g., $\frac{n}{p} \geq p$ or $\frac{n}{p} \geq p^2$. Note that, for determining time complexities, we will consider both, local computation time and inter processor communication time, in the standard way.

The problem studied in this paper is the design of *scalable parallel geometric algorithms* for such architectures, which are optimal or at least efficient for a wide range of ratios $\frac{n}{p}$. We present new techniques for designing efficient scalable parallel geometric algorithms, which are independent of the communication network. A particular strength of our approach, which is very different from the one presented in [1], is that all inter processor communication is restricted to a constant number of two types of global routing operations: *global sort* and *segmented broadcast* (to be explained in Section 2).

* *This work was partially supported by the Natural Sciences and Engineering Research Council of Canada and the ESPRIT Basic Research Actions Nr. 3075 (ALCOM) and Nr. 7141 (ALCOM II).*

9th Annual Comptutational Geometry,5/93/CA,USA

In a nutshell, the basic idea for our methods is as follows: We try to combine optimal sequential algorithms for a given problem with an efficient global routing and partitioning mechanism. We devise a constant number of partitioning schemes of the global problem (on the entire data set of n data items) into p subproblems of size $O(\frac{n}{p})$. Each processor will solve (sequentially) a constant number of such subproblems, and we use a constant number of global routing operations to permute the subproblems between the processors. Eventually, by combining the $O(1)$ solutions of it's $O(\frac{n}{p})$ size subproblems, each processor determines it's $O(\frac{n}{p})$ size portion of the *global* solution.

The above is necessarily an oversimplification. The actual algorithms will do more than just those permutations. The main challenge lies in devising the above mentioned partitioning schemes. Note that, each processor will solve only a constant number of $O(\frac{n}{p})$ size subproblems, but eventually will have to determine it's part of the entire $O(n)$ size problem.

In particular, we present algorithms for the following geometric problems:

1. lower envelope of non-intersecting line segments in the plane (and, with a slight modification of the model, for possibly intersecting line segments),
2. $2D$-nearest neighbors of a point set,
3. $3D$-maxima,
4. $2D$-weighted dominance counting,
5. area of the union of rectangles,
6. $2D$-convex hull of a point set.

Our scalable parallel algorithms for Problems 1-6 have a running time of

$$O(\frac{T_{sequential}}{p} + T_s(n,p))$$

on a p-processor Coarse Grained Multicomputer $CGM(n,p)$ with $\frac{n}{p} \geq p^2$ for Problem 5, $\frac{n}{p} \geq p$ for Problems 1-4, and $\frac{n}{p} \geq p \log p$ for Problem 6 and where $T_s(n,p)$ refers to the time of a global sort operation on a $CGM(n,p)$. As $T_{sequential} = \Theta(n \log n)$ for Problems 1-6, our algorithms either run in optimal time $\Theta(\frac{n \log n}{p})$ or in sort time $T_s(n,p)$ for the respective architecture. We will show that the first term dominates the sort time for $n \geq 2^{T_s(p,p)}$. For example, for hypercube networks, we obtain optimal algorithms for $n \geq p^{\log p}$.

Experiments have shown that, in addition to being scalable, our methods do quickly reach the point of optimal speed-up for reasonable data sizes. The fact that our algorithms use only very few well known and extensively studied global routing operations is also very positive in practice. These communication operations are usually available as system calls or as highly optimized public domain software. All other programming is within the sequential domain. Thus, even with modest programming efforts, the actual timings obtained are quite impressive.

The remainder of this paper is organized as follows: In the next section, we describe the above mentioned global routing operations. In the following sections we present our algorithms, one per section, in the above order. We discuss experimental results and give a conclusion in the last two sections.

2 Communication Model

The processors communicate via an interconnection network in which each processor may exchange messages of size $O(\log n)$ with any one of its immediate neighbors in constant time. Commonly used interconnection networks for CGM include 2D-mesh (e.g. Intel Paragon), hypercube (e.g. Intel iPSC/860) and the fat-tree (e.g. Thinking Machines CM-5). We refer the reader to [2, 4, 7, 12] for a more detailed discussion of the different architectures and algorithms.

We will now outline the four operations involving interprocessor communication which we will use in this paper and give the time complexity of the operations for the above three architectures. The first two operations concern all n data. Assume that the p processors of the $CGM(n,p)$ are numbered from 0 to $p-1$.

1) global sort: $T_s(n,p)$ refers to the time to sort $O(n)$ data items stored on a $CGM(n,p)$, $O(\frac{n}{p})$ data items per processor, with respect to the above mentioned processor numbering. The time complexity $T_s(n,p)$ of the global sort is $\Theta(\frac{n}{p}(\log n + \sqrt{p}))$ for a 2D-mesh, it is $O(\frac{n}{p}(\log n + \log^2 p))$ for a hypercube[1] and $\Theta(\frac{n}{p} \log n)$ for a fat-tree.

[1] The time complexities for the hypercube and the 2D-mesh are based on Batcher's bitonic sort[2]. Note that better deterministic [5] and randomized [15] sorting algorithms exist, which, however, are not of practical use.

It is interesting to study, for which ratio of n and p the global sort becomes optimal, that is $T_s(n,p) = O(\frac{n \log n}{p})$. The time complexity of the coarse grained version of Batcher's bitonic sort is $T_s(n,p) = O(\frac{n}{p}(\log n + T_s(p,p)))$. Hence, $\log n > T_s(p,p) \Leftrightarrow n \geq 2^{T_s(p,p)}$. We thus obtain optimal global sort algorithms for $n \geq p^{\log p}$ on a hypercube and for $n \geq 2^{\sqrt{p}}$ on a 2D-mesh. The fat-tree sorting algorithm is optimal for $n \geq p$.

2) segmented broadcast: In a segmented broadcast operation, $q \leq p$ processors with numbers $j_1 < j_2 < \ldots < j_q$ are selected. Each such processor p_{j_i} broadcasts $O(\frac{n}{p})$ data from its local memory to the processors p_{j_i+1} to $p_{j_{i+1}-1}$. The time complexity $T_{sb}(n,p)$ of the segmented broadcast is $\Theta(\frac{n}{p}\sqrt{p})$ for a 2d-mesh and $\Theta(\frac{n}{p}\log p)$ for a hypercube and a fat-tree.

The next two operations concern p or p^2 data and their time complexity is thus independent of the problem size n.

3) multinode broadcast: In a multinode broadcast operation, every processor (in parallel) sends one message to all other processors. The time complexity $T_b(p)$ for any interconnection network is $T_b(p) = \Theta(p)$.

4) total exchange: In a total exchange operation, every processor (in parallel) sends a different message to each other processors. The time complexity $T_x(p)$ of the total exchange is $T_x(p) = \Theta(p^{\frac{3}{2}})$ for a 2D-mesh, and $T_x(p) = \Theta(p \log p)$ for a hypercube and a fat-tree.

3 Lower Envelope of Line Segments in the Plane

Given a set S of n opaque line segments in the Euclidean plane, the *Lower Envelope* Problem, $LE(S)$, consists of computing the segment portions visible from the point $(0, -\infty)$. We use the following fact.

Lemma 1 *The lower envelope of n line segments is x-monotonic. If the line segments do not intersect it has a size of $O(n)$. If the line segments may intersect the size of the lower envelope is $O(n\alpha(n))$, where $\alpha()$ is the extremely slow growing inverse Ackermann function.*

Proof: The x-monotonicity is a trivial fact. The same holds for the size of the lower envelope in the case of non-intersecting line segments. For the general case see [10]. ∎

We will restrict ourselves first on the case of non-intersecting line segments. Running the lower envelope algorithm sequentially on the $\frac{n}{p}$ segments in the local memory of each processor reduces the problem to solve in parallel to computing the lower envelope of p x-monotonic chains, each of size $O(\frac{n}{p})$. We then subdivide the Euclidean plane in p vertical slabs. The details of the algorithm are as follows.

1. Let $S_i \subset S$ denote the set of $\frac{n}{p}$ segments in the local memory of processor p_i. Locally compute $LE(S_i)$ in processor p_i, which results in x-monotonic chains C_i.
2. Globally sort the segments in $\bigcup_{i=1}^{p} C_i$ by the x-coordinate of their right endpoints, which yields in sets V_i on processor p_i.
3. Perform a multinode broadcast with processor p_i sending l_i, the vertical line passing through the endpoint of a segment in V_i with largest x-coordinate as message. Now, each processor stores the set of lines defining the p vertical slabs.
4. Perform a total exchange, with processor p_i sending segment $s \in C_i$ as message to processor p_j, iff s intersects the vertical line l_j.
5. Each processor p_i receives the set R_i of segments intersecting l_i. The cardinality of R_i is p. Locally compute $LE(V_i \cup R_i)$.

The correctness of the algorithm follows from the monotonicity of the chains $C_i \in C$. The local lower envelope computations take time $O(\frac{n}{p}\log n)$. The communication time is $T_b(p) + T_x(p) + T_s(n,p) = O(T_s(n,p))$, for $\frac{n}{p} \geq p$. We thus obtain the following result.

Theorem 2 *Given a set S of n non-intersecting line segments in the Euclidean plane, then the Lower Envelope Problem can be solved on a p-processor Coarse Grained Multicomputer $CGM(n,p)$, $\frac{n}{p} \geq p$, in time $O(\frac{n \log n}{p} + T_s(n,p))$.*

With the same subdivision and communication scheme we can solve the problem for possibly intersecting line segments. We only have to replace the above used sequential algorithm, by the algorithm for computing the lower envelope due to [11]. As the size of the output is not linear in n, we need some extra memory. More

exactly, after the first local computation of the lower envelope each processor stores a chain of size $O(\frac{n}{p}\alpha(n))$. After the second computation of the lower envelope each processor stores a chain of size $O(\frac{n}{p}\alpha^2(n))$. Note that the total size of the lower envelope is only $O(n\alpha(n))$, but it can be unevenly distributed over the memory, such that we need $O(n\alpha^2(n))$ memory. We thus can state the following corollary.

Corollary 1 *Given a set S of n possibly intersecting line segments in the Euclidean plane, then the Lower Envelope Problem can be solved on a p-processor Coarse Grained Multicomputer $CGM(n\alpha^2(n), p)$, $\frac{n}{p} \geq p$, in time $O(\frac{n\alpha(n)\log n}{p} + T_s(n\alpha^2(n), p))$.*

4 2D-Nearest Neighbors of a Point Set

Given a set P of n points in the Euclidean plane, the Nearest Neighbor Problem, $NN(P)$, is to determine for each point $v \in P$ a point $w = NN_P(v)$, where $w \in P\backslash\{v\}$ and $dist(v,w) \leq dist(v,u)$ for all $u \in P\backslash\{v\}$. The following is an outline of our scalable algorithm for solving the Nearest Neighbor Problem on a p-processor Coarse Grained Multicomputer $CGM(n,p)$. We use the following lemma from [1].

Lemma 3 *(See Figure 1.) Let V and H be two point sets in a vertical and a horizontal slab. Let I be the set of four intersection points defined by the limiting lines of the two slabs and let C_{VH} be the set $\{w \in V\setminus H;\ \min_{p\in I}(dist(w,p)) < dist(w, NN_V(w))\}$. Then, $|\, C_{VH} \,| \leq 8$, and for all $w \in V\backslash H$ whose nearest neighbor v is in $H\backslash V$ holds: $w \in C_{VH}$.*

Proof: The ideas of the proofs are as follows. A point in a plane can be nearest neighbor to at most 6 points, as the angle between those must be at least $\frac{\pi}{3}$. Analoguously a point $p \in I$ can be nearest neighbor to at most 2 points in $V\setminus H$ and the size of set I is 4, thus $|\, C_{VH} \,| \leq 8$.

Let $v \in H\setminus V$ be the nearest neighbour to a point $w \in V\setminus H$. Assume w.l.o.g. that v is to the left of V and w below H, and let $p_{i-1,j-1}$ be the lower left intersection point of the limiting lines of V and H. Then $dist(w, NN_V(w)) > dist(w,v) > dist(w, p_{i-1,j-1}) \geq \min_{p\in I} dist(w,p)$, that is $w \in C_{VH}$. ∎

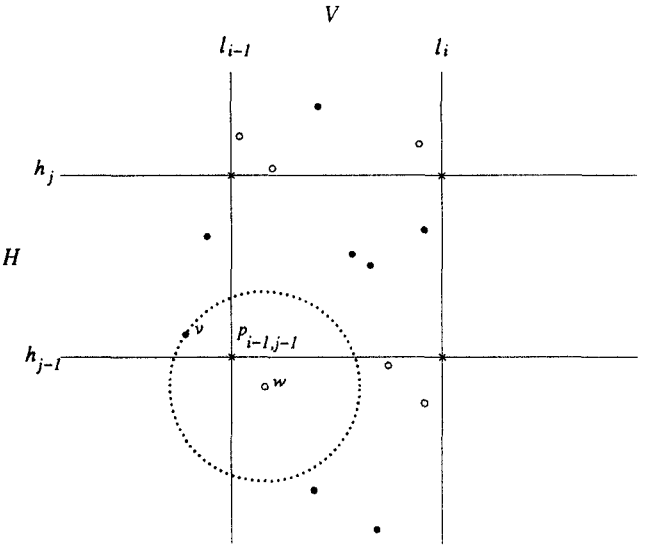

Figure 1: The points ∘ belong to the set C_{VH}, the points × denote the set I.

We subdivide the plane in p vertical and horizontal slabs, each containing point sets V_i and H_j, $1 \leq i,j \leq p$, of size $O(\frac{n}{p})$, respectively. According to the above lemma a point $v \in V_i \cap H_j$ is the nearest neighbor of either a point in V_i or in H_j or in $C_j := \bigcup_{k=1}^{p} C_{kj}$, where C_{kj} denotes $C_{V_k H_j}$. Note that the size of C_j is $\leq 8p$, i.e. independent of the problem size n.

Our algorithm maps the subproblem $NN(V_i)$, the computation of the sets C_{ik}, $1 \leq k \leq p$, and the subproblem $NN(H_i \cup C_i)$, onto processor p_i, and solves them sequentially. The details of the algorithm are as follows.

1. Globally sort the points by their x-coordinates, which yields sets V_i on processors p_i.
2. Each processor p_i solves $NN(V_i)$ independently.
3. Globally sort the points by their y-coordinates which yields sets H_i on processors p_i.
4. In order to compute the sets C_j, we perform a multinode broadcast with processor p_i sending h_i, the element of H_i with largest y-coordinate, as its message.
5. Each processor p_i computes all sets C_{ik}, $1 \leq k \leq p$, sorting V_i locally by the y-coordinates of the points and performing a scan over this sorted sequence.
6. Perform a total exchange, where each processor p_i sends the set C_{ik} as message to processors p_k, for $1 \leq k \leq p$.

7. Each processor p_i computes $NN(H_i \cup C_i))$ independently.

8. As a point v in a slab V_i can have up to p copies in the sets C_k, $1 \leq k \leq p$, we have to route them back to p_i, in order to determine the nearest neighbor of v among its copies. To do so perform a total exchange, where each processor p_i sends the set C_{ki} (with the nearest neighbor information) as message to processors p_k, $1 \leq k \leq p$.

9. Locally sort the points and scan over this sorted sequence of size p to determine the copy with the nearest neighbor.

10. Perform a total exchange operation to send these copies to the processor holding the appropriate horizontal slice.

The correctness of the algorithm follows immediately from Lemma 3 and the running time can be analyzed as follows. The local nearest neighbor computations take time $O(\frac{n}{p} \log n)$ [14]. The computation of the sets C_{ij}, $1 \leq i,j \leq p$, is dominated by the local sort and takes time $O(\frac{n}{p} \log n)$. The communication time is $T_b(p) + 3T_x(p) + 2T_s(n,p)$, which is dominated by the global sorting steps to compute the horizontal and vertical slabs.

Theorem 4 *Given a set P of n points in the Euclidean plane, the Nearest Neighbor Problem can be solved on a p-processor Coarse Grained Multicomputer $CGM(n,p)$, $\frac{n}{p} \geq p$, in time $O(\frac{n \log n}{p} + T_s(n,p))$.*

5 3D-Maxima

Given P a set of n points in the Euclidean space, the 3D-Maxima Problem, *3Dmax*(P), is to determine the set of points $v \in P$, such that there is no point $w \in P$ with $x(w) > x(v)$ and $y(w) > y(v)$ and $z(w) > z(v)$. We say point v is not *dominated* by any other point of P.

Lemma 5 *(See figure 2) Let V be a set of points in the vertical slab delimited by the two planes $\mathcal{V}$ and $\mathcal{V}'$, with $x(\mathcal{V}) < x(\mathcal{V}')$ and which are parallel to the yz-plane. Let further H be a set of points in the horizontal slab delimited by the planes $\mathcal{H}$ and $\mathcal{H}'$, with $z(\mathcal{H}) < z(\mathcal{H}')$ and which are parallel to the xy-plane. Then a point $v \in V \setminus H$ which is dominated by a point $w \in H \setminus V$ iff it is also dominated by the intersection point q of $\mathcal{V}'$ with 2Dmax of the projection of H on the plane $\mathcal{H}$.*

Proof: Let $v \in V \setminus H$, $w \in H \setminus V$. As v is dominated by w the following holds $z(v) < z(\mathcal{H}) = z(q) \leq z(w)$. Further $x(v) < x(\mathcal{V}') = x(q) \leq x(w)$. For all points r on the 2D maximum of the projection of H on the plane $\mathcal{H}$, with $x(r) \leq x(w)$ holds $y(w) \geq y(r)$, and thus $y(v) \leq y(w) \leq y(q)$. That is v is dominated by q. ∎

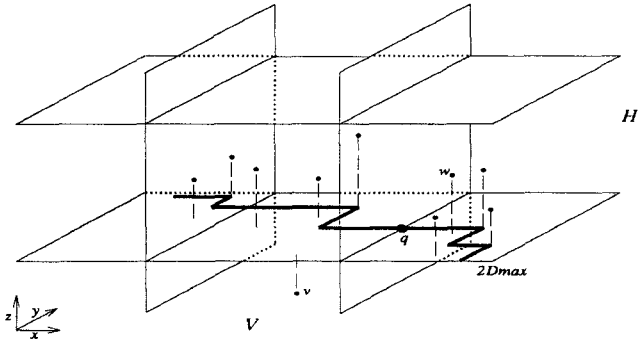

Figure 2: The point $v \in V \setminus H$ is dominated by $w \in H \setminus V$ and thus dominated by the intersection point q.

We subdivide the Euclidean space in p vertical and horizontal slabs containing $O(\frac{n}{p})$ points each. According to the above lemma a point $v \in V_i \cap H_j$ is either dominated by a point in V_i or H_j, or by one of the p intersection points in I_i which are induced by the other horizontal slabs.

Our algorithm maps the subproblems *3Dmax*(H_i), *2Dmax*$(proj(H_i))$ and *3Dmax*$(V_i \cup I_i)$ onto processor p_i and solves them sequentially. The details of the algorithm are as follows.

1. Globally sort P by z-coordinate, which yields the horizontal slabs H_i on processor p_i. Let $\mathcal{H}_i$ be the plane parallel to the xy-plane passing through the point in H_i with the smallest z-coordinate.

2. Locally compute *3Dmax*(H_i) and remove all dominated points from H_i. Compute locally the *2Dmax* of the projection of H_i to the plane $\mathcal{H}_i$. These form x-monotonic chains $C(P_i)$.

3. Globally sort the point set $\cup_{i=1}^{p} H_i$ by x-coordinate which yields a partition of the space in vertical slabs V_i.

4. Perform a multinode broadcast, where processor p_i sends v_i, the element of V_i with largest x-coordinate as message.

5. The broadcasted points define planes $\mathcal{V}_j$, $1 \leq j \leq p$, parallel to the yz-plane. Locally determine the in-

tersections of the segments of the chain $C(P_i)$ with the p planes. As $C(P_i)$ is x-monotonic each processor holds at most p intersection points.

6. Perform a total exchange operation, where processor p_i sends the intersection point of chain $C(P_i)$ with plane $\mathcal{V}_j$ as message to processor p_j. Let I_i denote the set of points processor p_i receives. As there are p monotonic chains, $\mid I_i \mid \leq p$.
7. Locally compute $3Dmax(V_i \cup I_i)$.

The correctness of the above algorithm follows immediately from Lemma 5 and we can analyze the running time as follows. The local 3D-Maxima computation in steps 2 and 7 as well as the local 2D-Maxima computation in step 2 can be performed in time $O(\frac{n}{p} \log n)$ using the algorithm from [14]. The intersection points in step 5 can be computed in time $O(\frac{n}{p})$, as we have to merge the chains with the planes $\mathcal{H}_j$, $1 \leq j \leq p$. The communication time is $2T_s(n,p) + T_b(p) + T_x(p)$ and dominated by the global sorting steps. We thus obtain the following result.

Theorem 6 *Given a set P of n points in the Euclidean space, the 3D-Maxima Problem, can be solved on a p-processor CGM(n,p), $\frac{n}{p} \geq p$, in time $O(\frac{n \log n}{p} + T_s(n,p))$.*

6 2D-Weighted Dominance Counting

Given a set P of n weighted points in the Euclidean plane, the *2D-Weighted Dominance* Problem, $wdom(P)$, is to determine for all points $v \in P$, the sum of all points dominated by v. Let $wsum(X)$ for some set X of points denote the sum of the weights of all points $x \in X$.

Lemma 7 *Let V and H be a vertical and a horizontal slab, delimited from left and from below by the lines l and h respectively. For a point $v \in V \cap H$, $wdom(v,P)$ = $wdom(v,V)$ + $wdom(v,H)$− $wdom(v,V \cap H)$ + $wsum(\{w \in P;\ x(w) < x(l)$ and $y(w) < y(h)\})$.*

Proof: Let $v \in H \cap V$. Points $w \in H \cap V$ which are dominated by v are counted in $wdom(v,V)$ and in $wdom(v,H)$. All points $w \in P$ to the left of l and below h are dominated by v as $x(w) < x(l) < x(v)$ and $y(w) < y(l) < y(v)$. ∎

We subdivide the Euclidean plane in p vertical and horizontal slabs containing $O(\frac{n}{p})$ points each. The last term of the formula of the above lemma for a point $v \in V_i \cap H_j$, with h_{j-1} as delimiting line of H_j, becomes then $s_{ij} := \sum_{k=1}^{i-1} wsum(\{w \in V_k;\ y(w) < y(h_{j-1})\})$

Our algorithm maps the subproblems $wdom(v,H_i)$, $wdom(v,V_i)$, $wdom(v,V_i \cap H_j)$, $1 \leq j \leq p$, and the computation of s_{ij}, $1 \leq j \leq p$, onto processor p_i and solves them sequentially. The details of the algorithm are as follows.

1. Globally sort the points by their y-coordinates which yields sets H_i on processors p_i.
2. Locally compute $wdom(H_i)$.
3. Perform a multinode broadcast, where processor p_i sends h_i, the horizontal line passing through the point with the largest y-coordinate in H_i.
4. Globally sort the points by their x-coordinates which yields sets V_i on processors p_i.
5. Locally sort the by their y-coordinates and merge this sorted sequence with the lines $h_1, \ldots, h_p$ which yields the sets $V_i \cap H_j$, $1 \leq j \leq p$, on each processor p_i.
6. Locally compute $wdom(V_i)$ and $wdom(V_i \cap H_j)$, $1 \leq j \leq p$.
7. Locally compute $s_{ij} = wsum(\{v \in V_i; y(v) < y(h_j)\})$, $1 \leq j \leq p$.
8. Perform a total exchange, with processor p_i sending s_{ij} as message to processor p_{j+1}, $1 \leq j < p$.
9. Locally compute $wdom(v,P)$, using the formula from the above lemma.

The correctness of the above algorithm follows immediately from Lemma 7 and we can analyze the running time as follows. The local weighted dominance computation in steps 2 and 6 can be performed in time $O(\frac{n}{p} \log n)$. The local sort and merge in step 5 takes time $O(\frac{n}{p} \log n + p)$. The communication time is $2T_s(n,p) + T_b(p) + T_x(p)$ and dominated by the global sorting steps. We thus obtain the following theorem.

Theorem 8 *The 2D-Weighted Dominance Counting Problem can be solved on a p-processor Coarse Grained Multicomputer CGM(n,p), $\frac{n}{p} \geq p$, in time $O(\frac{n \log n}{p} + T_s(n,p))$.*

7 Area of the Union of Isothetic Rectangles

Given a set R of n isothetic rectangles the "Measure Problem" is to compute the area M covered by the union of R.

Let V and H be a vertical and a horizontal slab, let box b be their intersection, and assume that b is subdivided into disjoint horizontal stripes not containing any corner of a rectangle of R. Let $xcover(s)$ be the horizontal length covered by rectangles intersecting the stripe s with at least one vertical edge, and let $ycover(s)$ be the vertical length covered by rectangles intersecting the box b with at least one horizontal edge and having no corner in b (see Figure 3). Note that $xcover(s)$ and $ycover(s)$ are not symmetric.

Lemma 9 *With V, H, b and s defined as above, the following holds: either box b is covered by a rectangle $r \in R$ with no corner in $V \cup H$ and each stripe $s \in b$ contributes its whole surface to M, or stripe s contributes $m(s)$ area to M, with*

$$m(s) := xcover(s) \times height(s) + ycover(s) \times length(s) - xcover(s) \times ycover(s).$$

Proof: As stripes do not contain corners of rectangles, their coverage can be expressed as the product of horizontal and vertical coverage and the area covered twice must be subtracted. ∎

We subdivide the plane into p vertical and horizontal slabs, each containing $O(\frac{n}{p})$ corners of rectangles of R. This subdivision yields p^2 boxes. Lemma 9 suggests the following procedure of the algorithm. To detect the coverage of boxes by rectangles, each processor checks $\frac{n}{p}$ rectangles against all p^2 boxes. The computation of *xcover* of the stripes in a vertical slab and the computation of *ycover* of the stripes in a horizontal slab are assigned to a single processor. The details of the algorithm are as follows.

1. Globally sort the vertical edges of the rectangles by their x-coordinate and compute the set $\mathcal{L} = \{l_0, l_1, \ldots, l_p\}$ of vertical lines passing through every $\frac{n}{p}$-th vertical edge. Analogously compute $\mathcal{H} = \{h_0, h_1, \ldots, h_p\}$ the set of horizontal lines passing through every $\frac{n}{p}$-th horizontal edge. Perform a multinode broadcast with the lines as message, such that each processor holds a copy of $\mathcal{L}$ and $\mathcal{H}$.

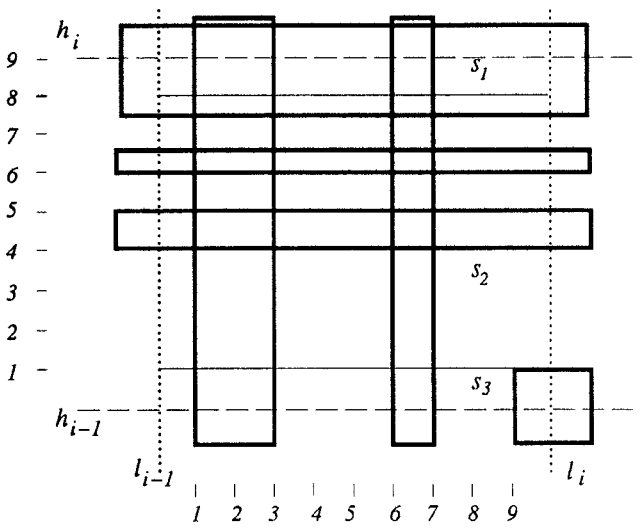

Figure 3: Box b_{ij} consists of stripes s_1, s_2 and s_3, with $xcover(s_1) = xcover(s_2) = 3$, $xcover(s_3) = 4$, $ycover(s_1) = 1$, $ycover(s_2) = 2$, and $ycover(s_3) = 0$.

2. Locally compute which of the p^2 boxes defined by $\mathcal{L}$ and $\mathcal{H}$ are completely covered by one of the $\frac{n}{p}$ rectangles stored in the local memory using Lemma 10 below. Perform a total exchange operation, where the message for processor p_j consists of a bitvector of length p with bit i set to 1, iff b_{ij} is covered. The logical OR of all messages received by processor p_j yields the set of all covered boxes in the horizontal slab (h_{j-1}, h_j). More details about the sequential algorithm are given below.

3. Locally sort $\mathcal{H}$ and the horizontal lines through the corners of the rectangles in a vertical slab by their y-coordinates. This defines a subdivision of each vertical slab into $O(\frac{n}{p})$ stripes.

4. Locally compute $xcover(s)$ for all stripes s in a vertical slab. Perform a plane sweep in upwards direction in time $O(\frac{n}{p} \log n)$, using the algorithm from [16] with the following simplification. Instead of computing the covered area between two successive sweep line positions, associate the covered length in x-direction with the stripe between these lines.

5. Globally sort the stripes by the y-coordinate of the lower left corner, such that all $O(\frac{n}{p})$ stripes between lines h_{i-1} and h_i are on processor p_i.

6. Locally compute $ycover(s)$ for all stripes s in a horizontal slab using Lemma 11 below. This gives us the contribution $m(s)$ of the stripe s to the measure

of the rectangles. More details about the sequential algorithm are given below.

7. Sum $m(s)$ for all stripes s.

We next show how to perform Step 2 of the above algorithm.

Lemma 10 *Given $\frac{n}{p}$ rectangles and the sets $\mathcal{L}$ and $\mathcal{H}$ it takes time $O(\frac{n}{p}\log p + p^2)$ to sequentially compute the boxes which are covered by a rectangle.*

Proof: We perform a left to right plane sweep. The Y-structure is a static segment tree with intervals $(h_0, h_1], \ldots, (h_{p-1}, h_p]$ as leaves. A variable $xmax$, initialized with 0, is associated with each node. The X-events are slabs (l_i, l_{i+1}) and the rectangles r, sorted by $x(l_i)$ and $x(\textit{left-edge}(r))$. Ties are broken in favor of the rectangles.

The X-events are handled as follows. For a rectangle r, we perform the following computation for all nodes w, whose associated interval is covered by the left edge of r, and whose parent's interval is not: $xmax(w) := \max(xmax(w), x(\textit{right-edge}(r)))$. For a slab (l_i, l_{i+1}) we first compute $xmax$ for the set of leaf nodes propagating $xmax$ down to the leaves. For each leaf node v representing an interval $(h_{j-1}, h_j]$, we report box b_{ij} if $x(l_{i+1}) < xmax(v)$.

The running time is $O(\frac{n}{p}\log p + p^2)$ as the processing of each of the $\frac{n}{p}$ rectangle event takes time $O(\log p)$ and the processing of each of the p slab event takes time $O(p)$. ∎

We finally show how to perform step 6 of the algorithm.

Lemma 11 *Given $\frac{n}{p}$ stripes in a horizontal slab H and $\frac{n}{p}$ rectangles with at least one horizontal edge in H, we can sequentially compute ycover(s) in time $O(\frac{n}{p}\log n)$.*

Proof: Only rectangles which intersect a box b_{ij} with a horizontal edge and have no corner in it can contribute to $ycover(s)$ of a stripe s in b_{ij}. Hence, we align each rectangle r with the leftmost and rightmost lines in $\mathcal{L}$ intersecting r.

We sweep from left to right using the plane sweep algorithm from [16] (see also [14]) and additionally perform range queries in the course of the algorithm. The Y-structure is a static segment tree built on the y-coordinates of the horizontal edges of the rectangles.

The X-events are the vertical edges of the aligned rectangles and of the stripes. When the sweep line reaches a vertical line l_i we insert all left and delete all right rectangle edges aligned on l_i. We then perform range queries in the Y-structure for the left edges of all stripes s in b_{ij}, which yields $ycover(s)$.

Each insertion, deletion and query takes time $O(\log n)$, which proves our lemma. ∎

The correctness of the overall algorithm follows immediately from Lemma 9, and the running time can be analyzed as follows. The sequential plane sweeps take time $O(\frac{n}{p}\log p + p^2 + \frac{n}{p}\log n) = O(\frac{n}{p}\log n)$, for $\frac{n}{p} \geq p^2$. The communication time is $2T_b(p) + \lceil \frac{p}{\log n} \rceil T_r(p) + 3T_s(n,p)$ and thus dominated by the global sorting steps to compute the horizontal and vertical slabs. We thus can state the following theorem.

Theorem 12 *Given a set R of n isothetic rectangles the Measure Problem can be solved on a p-processor Coarse Grained Multicomputer CGM(n,p), $\frac{n}{p} \geq p^2$, in time $O(\frac{n\log n}{p} + T_s(n,p))$.*

8 2D-Convex Hull of a Point Set

Given a set S of n points in the Euclidean plane the *convex hull* of S is the smallest convex set containing all points. The two points of S with minimum and maximum x-coordinate partition $CH(S)$ into two parts: the *upper* and the *lower chain*. By symmetry it is sufficient to show how to compute the upper chain. We use the following fact due to [13].

Lemma 13 *Given two upper chains of size $O(n)$ each whose x-coordinates do not overlap and which are stored in an array in sorted order, a single processor can compute in $O(\log n)$ time the unique tangent to both chains and the two points of tangency.*

Proof: We just recall the idea of the algorithm. Let C_1 and C_2 be the two arrays which store the upper chains. Starting with a line passing through the points $C_i[\frac{|C_i|}{2}]$, $i = 1, 2$, it adjusts the pair of points the line passes through. To do so the algorithm performs a binary search on the arrays, halving the number of points to look at in at least one of the arrays in constant time. The decision in which half to continue the search is based on whether the line touches, enters or leaves an

upper chain (c.f.Figure 4). Note that this is a local criterium, in the sense that we only have to look at the predecessor and successor of the point the line passes. ∎

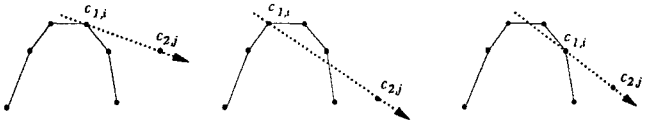

Figure 4: Line l_{ij} touches, enters and leaves the upper chain.

The idea of our convex hull algorithm is as follows. We subdivide the plane in p vertical slabs, each containing point sets V_i, $1 \leq i \leq p$, of size $O(\frac{n}{p})$. We map the subproblem $CH(V_i)$ on processor p_i and solve them sequentially. The remaining task is then to compute the p^2 pairwise tangents. The details of the algorithm are as follows.

1. Globally sort the points by their x-coordinates, which yields sets V_i on processor p_i.
2. Locally compute the upper chain of V_i, and store the edges in an array C_i in sorted order.
3. Perform a broadcast, where processor p_i sends a message consisting of the triple $c_i := C_i[\frac{|C_i|}{2}]$, its predecessor and its successor in C_i to the other processors.
4. Compute all p^2 pairwise tangents as follows. Each processor p_i maintains p lines l_{ij} which initially pass all through c_i and c_j. The following two steps are performed $2 \log n$ times. Each processor p_i performs the binary search step for the p lines l_{ij} on its upper chain C_i. Let c_{ij} denote the point the line l_{ij} passes through next. Perform a total exchange, where processor p_i sends the p different triples, consisting of c_{ij}, its predecessor and successor in C_i, as message to processors p_j, $1 \leq j \leq p$.
5. Locally compute for each set V_i the contribution to the upper chain of S. This is the (possibly empty) sequence of points on the upper chain between the rightmost point of tangency of a right tangent endpoint and the leftmost point of tangency of a left tangent endpoint. If these points of tangency fall together we simply have to check whether they form an angle larger than π.

The correctness of the algorithm is easy to see. The time complexity can be analyzed as follows. The local convex hull computation takes time $O(\frac{n}{p})$ using the Graham scan algorithm. The communication complexity is $T_s(n,p) + 2 \log n T_x(p)$. The sorting step dominates the communication time, if $\frac{n}{p} \geq pT_s(p,p)$. We thus obtain the following result.

Theorem 14 *The 2D-Convex Hull Problem for a set of n points can be solved on a p-processor Coarse Grained Multicomputer $CGM(n,p)$, $\frac{n}{p} \geq pT_s(p,p)$, in time $O(\frac{n \log n}{p} + T_s(n,p))$.*

9 Experimental Results

To demonstrate the practical relevance of our scalable CGM algorithms, we implemented the Lower Envelope algorithm for non intersecting line segments on an Intel iPSC/860 multicomputer. Our code is less than 400 lines long and is largely unoptimized, except for the public domain sorting code. We used Intel's standard FORTRAN compiler (we did not have available a high performance i860 compiler). Our input data consisted of randomly generated line-segment sets.

We ran two kinds of experiments. In the first we fixed the problem size and varied the number of processors (see Figure 5). The super linear speedup we obtained is due to the fact, that first running a sequential algorithm on the data drastically reduces the number of segments which must be further treated.

number of processors	communication time [msec]	total time [msec]	work
1	0	11533	11533
2	67	5212	10424
4	184	2422	9688
8	359	1396	11168

Figure 5: Running time for a problem of fixed size (65536 line segments) with a varying number of nodes of an Intel iPSC/860 multicomputer.

In the second experiment we ran the algorithm with 8 processors and varied the problem size between 2^8 and 2^{19}. Figure 6 shows that up to a problem size of 2^{13} the communication time dominates the local computation.

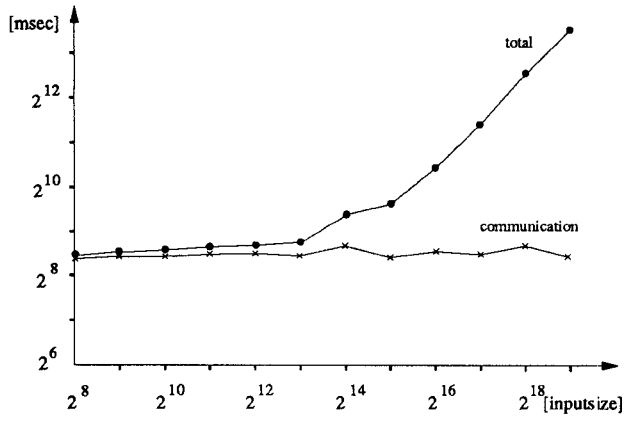

Figure 6: Running time of the lower envelope algorithm (for non-intersecting line segments) on an 8 node Intel iPSC/860 multicomputer, with varying number of line segments.

10 Conclusion

In this paper we have presented efficient or optimal scalable algorithms for a number of geometric problems, designed for a machine model which reflects real parallel machines. All algorithms have in common that they need a bare minimum of inter processor communication which is in general much more expensive than local computation. They do not depend on a specific architecture, are easy to implement and they are not only efficient in a theoretical sense, but fast in practice as experiments show.

It remains open if it is possible to make our algorithms fully scalable, that is to develop algorithms for any $n \geq p$ and not only for $n \geq p^2$ or $n \geq p^3$. Another open problem is to develop scalable algorithms for some other fundamental problems as for example the convex hull of a point set in the Euclidean space or the Voronoi diagram of a point set in the Euclidean plane.

Finally, we remark that we can use similar techniques to decompose data structures as for example the segment tree. Using this data structure we solve problems as trapezoidal decomposition, red/blue segment intersection counting and reporting, and finding, for a set of simple polygons, all directions for which a uni or multidirectional translation ordering exists [8]. These results will be presented in a forthcoming companion paper [9].

References

[1] M. J. Atallah and J.-J. Tsay. *On the parallel-decomposability of geometric problems. Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 104–113, 1989.

[2] K.E. Batcher. *Sorting networks and their applications. Proc. AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

[3] J. L. Bentley. *Algorithms for Klee's rectangle problems.* Carnegie-Mellon Univ., Penn., Dept. of Comp. Sci. Unpublished notes, 1977.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[5] R. Cypher and C. G. Plaxton. *Deterministic sorting in nearly logarithmic time on the hypercube and related computers. ACM Symposium on Theory of Computing*, 193–203. ACM, 1990.

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

[7] R. I. Greenberg and C. E. Leiserson. *Randomized Routing on Fat-trees. Advances in Computing Research*, 5:345–374, 1989.

[8] F. Dehne and J.-R. Sack. *Translation separability of sets of polygons. The Visual Computer 3:* 227–235, 1987.

[9] F. Dehne and A. Fabri and A. Rau-Chaplin. *Data structure decomposition for Coarse Grained Multicomputers. Manuscript.*

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

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

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

[13] M. H. Overmars and J. van Leeuwen. *Maintenance of configurations in the plane. J. Comput. Syst. Sci.*, 23:166–204, 1981.

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

[15] J. H. Reif and L. G. Valiant. *A logarithmic time sort for linear size networks. J. ACM*, Vol.34, 1:60–76, 1987.

[16] J. van Leeuwen and D. Wood. *The measure problem for rectangular ranges in d-space. J. Algorithms*, 2:282–300, 1981.