Scalable parallel geometric algorithms for coarse grained multicomputers

Franck Dehne, Andreas Fabri, Andrew Rau-Chaplin

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Rapports de Recherche

N°1819

Programme 4

Robotique, Image et Vision

SCALABLE PARALLEL GEOMETRIC ALGORITHMS FOR COARSE GRAINED MULTICOMPUTERS

Frank Dehne
Andreas Fabri
Andrew Rau-Chaplin

2 Decembre 1992

Institut National
de Recherche
en Informatique
et en Automatique

2004 route des Lucioles
B.P. 93
06902 Sophia-Antipolis
France

Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers

Algorithmes géométriques parallèles multi-échelles pour machines multi-processeurs à gros grain *

Frank Dehne † **Andreas Fabri** ‡ **Andrew Rau-Chaplin** †

Programme 4 : Robotique, Image et Vision.

Keywords: Computational Geometry, Decomposability, Parallel Algorithms, Coarse Grained Multicomputer

---

* *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).*

† *School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6.*

‡ *INRIA — B.P.93 — 06902 Sophia-Antipolis cedex, France.*

Abstract

Whereas most of the literature assumes that the number of processors p is a function of the problem size n, in *scalable algorithms* p becomes a parameter of the time complexity. This is a more realistic modelisation of real parallel machines and yields optimal algorithms, for the case that $n \gg p$, where "$\gg$" is a function depending on the architecture of the interconnection network. In this paper we present scalable algorithms for a number of geometric problems, namely lower envelope of line segments, $2D$-nearest neighbour, $3D$-maxima, $2D$-weighted dominance counting, area of the union of rectangles, $2D$-convex hull.

The main idea of these algorithms is to decompose the problem in p subproblems of size$O(\frac{n}{p} + f(p))$, with $f(p) \leq \frac{n}{p}$, which can be solved independently using optimal sequential algorithms. For each problem we present a spatial decomposition scheme based on some geometric observations. The decomposition schemes have in common that they can be computed by globally sorting the entire data set at most twice. The data redundancy of $f(p)$ duplicates of data elements per processor does not increase the asymptotic time complexity, and ranges, for the algorithms presented in this paper, from p to p^2. The algorithms do not depend on a specific architecture, they are easy to implement and in practice efficient as experiments show.

Résumé

Tandis que la plupart de la littérature suppose que le nombre de processeurs p est une fonction de la taille du problème n, dans les *algorithmes multi-échelles* p devient un paramètre de la complexité en temps. Ceci est une modélisation plus réaliste de machines parallèles réelles et donne des algorithmes optimaux, dans le cas $n \gg p$, où "$\gg$" est une fonction dépendant de l'architecture du réseau d'interconnection. Dans cet article nous présentons des algorithmes multi-échelles pour plusieurs problèmes géométriques, l'enveloppe inférieure de segments de droites, plus proche voisin dans le plan, maxima $3D$, comptage de la dominance à poids, aire de l'union de rectangles, enveloppe convexe dans le plan.

L'idée principale de ces algorithmes consiste à décomposer le problème en p sous-problèmes de taille $O(\frac{n}{p} + f(p))$, avec $f(p) \leq \frac{n}{p}$, qui peuvent être résolus indépendamment en utilisant des algorithmes séquentiels optimaux. Pour chaque problème nous présentons un schéma de décomposition spatiale basé sur des observations géométriques. Les schémas de décomposition ont en commun le fait qu'ils peuvent être calculés en triant l'ensemble entier de données au plus deux fois. La redondance de $f(p)$ copies de données par processeur n'augmente pas la complexité asymptotique en temps, et varie, pour les algorithmes présentés dans cet article, entre p et p^2. Les algorithmes ne dépendent pas d'une architecture spécifique, ils sont simples à implémenter et efficaces en pratique, comme des résultats expérimentaux le montrent.

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 $\frac{n}{p}$[1] 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).

In a nutshell, the basic idea for our methods is as follows: We try to combine

[1]Note that, if there exists an optimal fine grained algorithm, then, at least from a theoretical point of view, the problem is trivial. Simulation via Brent's Theorem [12] gives an optimal algorithm for any ratio of n and p. However, for most interconnection networks used in practice, many problems do not as yet have optimal fine grained algorithms, or such optimal algorithms are impossible due to bandwidth or diameter limitations.

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[2] and $\Theta(\frac{n}{p}\log n)$ for a fat-tree.

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.

[2]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.

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 $O(\frac{n}{p})$. The details of the algorithm are as follows. We subdivide the Euclidean plane in p vertical slabs

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 $w \in P \setminus \{v\}$ and $dist(v,w) \leq dist(v,u)$ for all $u \in P \setminus \{v\}$. 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 $w \in V \setminus H$ whose nearest neighbor v is in $H \setminus V$ 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 $w \in C_{VH}$. ∎

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.