Applied Geometric Algorithms on Boolean N-Cube Computers

Wen-Jing Hsu, Xiaola Lin, and Kuan-Tsae Huang

† Department of Computer Science Michigan State University East Lansing, MI 48824 hsu@cpswh.msu.edu
‡ IBM T. J. Watson Research Center Yorktown Heights, NY 10598 (914)789-7742 kth@ibm.com

(Extended Abstract)

Due to the polymorphism of the N-Cube interconnections, the N-Cube Computers have applications in numerous applications. The computer architecture is also one of the most promising for massively parallel processing in the future. Both algorithms and software tools for these parallel computers are in demand. To investigate techniques of designing parallel algorithms for this type of machines, we studied solutions for a class of geometric problems on the N-Cube parallel computers. The geometric problems are the Convex Hull Problem, the Line Intersection Problem, and the Nearest Neighbors Problem. These problems have found applications in VLSI design, Computer Graphics, Image Processing/Pattern Recognition, and Robotics. The selected problems are also known to be related to many other problems of theoretical as well as practical importance. Efficient solutions of these key problems have immediate applications and can lead to solutions of other problems.

Key words: Parallel Algorithms, Hypercube, Geometric Problems, Divide-and-Conquer.

1. Introduction

The Boolean N-Cubes, (or, N-Cubes, Hypercubes, for short) are parallel computers with the hypercube interconnection between the processors (Ref. 3). These parallel computers are becoming popular due to their relatively low costs (compared to other supercomputers like Crays) and vast potential. It is believed that this potential is derived from the following two structural advantages:

i) The computer is polymorphic:

It has been demonstrated that the Boolean N-Cubes can be programmed to simulate many other parallel architectures: linear arrays (for systolic, pipelined operations), meshes, trees, Pyramids, etc., where each of the architectures has identified application areas. Thus, the cube architecture is open to many applications, which is a great advantage over many other parallel architectures.

ii) The Boolean N-Cubes have a recursive structure:

An N-cube can be recursively divided into two isomorphic subcubes. This property matches very well with the recursive programming techniques (e.g. Divide-and-Conquer (Ref. 1)). Thus the N-Cube computers are a natural choice for designing recursive algorithms. Furthermore, the experience gained with the N-Cube computers should be of value to future system architects as well as programmers of current parallel computers.

2. The Geometric Problems

The problems studied fall in the domain of Computational Geometry (Refs. 4-5), a recent branch of Analysis of Algorithms (Ref. 3). Each of the problems (Convex Hull, Line Intersection, and Nearest Neighbor) actually has a cluster of related problems.

(1) the Intersection Problem is to determine whether two geometric objects (points, lines, polygons, etc.) share a common point. For example, the Line-Intersection Problem is to determine intersections of a set of lines segments. This problem has many variations, by specializing on the type of queries (e.g. reporting all instances or detecting one instance of intersections), or the type of objects involved (e.g. vertical/horizontal lines, half-planes, polygons, etc.).
The number of processors is bounded by N. In (Ref. 19), Miller and Stout showed that the convex hull of a digitized picture input can be found in O(NlogN) time on an N-node mesh. In (Ref. 20), Miller and Miller presented a Hypercube algorithm for computing Convex Hulls for an M x M digitized picture input. Using N processors, the algorithm has a worst-case complexity of O(M^2 / N^2 + P^2 + M^2/3 + N^2/3), where P denotes the number of "candidates" for the vertices on the convex hull; t_v and t_p, respectively, the time required for a computation step on the individual processor and the time required for sending (or receiving) a unit-length message. Note that, in both cases, the input points are represented by finite integer coordinates, which is a valid assumption in the applications they are concerned. However, it is a relatively strong assumption, and must be considered when doing comparisons.

(2) Intersection Algorithms

In (Ref. 18), Miller and Stout give asymptotically optimal O(N^2) algorithms for using mesh computers to determine intersections among line segments and among polygons. Using a different approach, Jeong and Lee (Ref. 15) also give optimal results on mesh computers for the same problem. No N-Cube algorithm is currently available.

(3) Parallel Algorithms for Proximity Problems

Using the shared-memory model, Chow (Ref. 13) showed that the Voronoi diagram can be computed in O(log^2(N log^2N)) time using N processors, where N denotes the number of input points. Aggrawal et al. (Ref. 11) presented an O(log^2N) time parallel algorithm to solve the same problem with O(N log^2N) total space. Chow also presented an O(log^N) algorithm on the Cube-Connected-Cycles using O(log^2N) storage space per processor. La (Ref. 16) showed that O(N^2 log^2N) time is sufficient to compute Voronoi diagrams on an N^2 x N^2 mesh computer, with constant space requirement on each processor. Recently, in (Ref. 15), Jeong and Lee improved the time bound to O(N log^2N), which is already optimal for the mesh computers. Currently no N-Cube algorithm for the Proximity problems is published.

The development of Pyramid computers, originally intended for image processing, pattern recognition, and computer vision applications, also led to the discovery of a cluster of geometric algorithms (Refs. 21-23). A pyramid of size N is defined to have an N^1/2 x N^1/2 mesh connected computer as its base, and log_4N levels of mesh connected computers above (Ref. 21). Assuming the input to be an
4. New Results

We will present new algorithms for the following three problems on the hypercubes:

1. Convex Hull Problem: specify the convex hull for a set of N points.
2. Nearest Neighbor Problem: Find a nearest neighbor for each of the N planar points.
3. Line Intersection Problem: Detect intersection for a set of N input lines.

For input of our algorithms, it is assumed that the N data points have been evenly distributed on the M processors, where $M = 2^k$ for some integer $k$, and the output are also represented in the distributive manner. Our algorithms are based on the divide-and-conquer approach (Ref. 1). Specifically, a problem is solved recursively by subdividing the input data into two subsets which are allocated on two subcubes until a primitive case is encountered, then combining (again recursively) the two partial results by using the communication links between the two subcubes.

1. new Convex Hull Algorithm

The algorithm is based on a new set of decision rules which enable us to determine, in a distributive manner, the common tangents between two convex hulls. The total time complexity of the new algorithm is $O(\log^2 N)$ in the worst case, where $N$, the number of input data points, is equal to $M$, the number of processors available. In the case $M > N$, we also present a generalized algorithm which achieves $O(N \log (N/M))$ time complexity.

2. new Nearest Neighbor Algorithm

We present a new $O(\log^2 N)$ time algorithm for finding the nearest neighbor for each of the N points on the plane. In the k-dimensional case, the algorithm has a time complexity of $O(\log^2 N)$ for $k > 2$. Since it takes only $O(\log N)$ time to find a minimum on the hypercube, the Closest-Pair Problem can also be solved in the same time complexity.

3. new Line Intersection Algorithm

Assuming there are N planar line segments, the worst-case time complexity of our new algorithm for vertical/horizontal lines is $O(\log^3 N)$. For input lines of general orientations, it takes $O(\log^2 N)$ amount of time in the worst case.

Compared to the previous results, the improvements achieved in the new algorithms are significant. The techniques developed here may also be applied to solving other problems of similar character. We will also discuss possible extensions to these results. In particular, two key issues will be addressed:

i. How to distributively represent and manipulate different data types on the hypercube architecture, and

ii. How to synchronize the processors so that maximal parallelism can be achieved.

II. REFERENCES AND BIBLIOGRAPHY

1. General References

i). Shared Memory model:


ii). Linear Arrays (Systolic Algorithms)


iii). Mesh:


iv). Cubes:


v). Pyramids
