An Associative Implementation Of Classical Convex Hull Algorithms

Maher M. Atwah and Johnnie W. Baker
Mathematics and Computer Science
Kent State University
Kent, OH 44242
matwah,jbaker@mcs.kent.edu

Selim Akl
Computing and Information Science
Queen's University
Kingston, Ontario K7L 3N6, Canada
akl@qucis.queensu.ca

Abstract

This paper will present the implementation and comparison of new parallel algorithms for the convex hull problem. These algorithms are a parallel adaptation of the Jarvis March and the Quickhull algorithms. The computational model selected for these algorithms is the associative computing model (ASC) and the multiple associative computing model (MASC). Both models support massive parallelism through the use of data parallelism and constant time associative search and maximum functions. Also, ASC can be supported on many SIMD computers. These algorithms requires $O(n)$ space, $O(\log n)$ (i.e., $O(\log_2 n)$) average running time, and $O(n)$ worst case running time. These algorithms have been compared using random data.

Input Size (n)	Convex Hull Points (h)	log of input points
100	7	6.6
500	8	9
1000	10	10
3000	12	11.6
5000	12	12.3
10000	13	13.3
15000	13	13.9
16000	14	14
20000	16	14.3

Table 1: Number of convex hull points found for n random generated points.

1 Introduction

Given a set S of points in the plane, the *convex hull* of S is the smallest convex polygon for which each point of S is either on the boundary of the convex polygon or in its interior. We assume that no two points in S have the same x or y coordinates and that no three points in S lie on the same straight line as these assumptions make the algorithms easier to describe. However, these algorithms can be easily modified to eliminate the necessity of these assumptions. We also assume that only one point is stored in each PE. This assumption is not necessary as each PE can simulate additional PEs [9]. The randomly generated points used here in evaluating the average running time of an algorithm are assumed to be uniformly distributed points with integer coordinates from a square planar domain. It is shown in [8] (and illustrated in Table 1) that for n randomly generated points the expected number of convex hull points is $\log n$. Therefore it is reasonable to assume that the number of convex hull points is $O(\log n)$ in the average case.

The convex hull plays a central role in the field of computational geometry. This geometric concept finds practical applications in many areas including pattern recognition, image processing, engineering, computer graphics, design automation, and operations research.

Section 2 describes the associative model. Section 3 gives an associative parallel adaptation of the Jarvis March. In Section 4, an associative parallel adaptation of the Quickhull algorithm on the ASC model is presented. Section 5 presents an associative parallel Quickhull algorithm using the MASC model. Section 6 presents the timing results for the new convex hull algorithms given in this paper and in [2] which have been implemented on the WaveTracer (DTC) [10]. In the final section, a summary is given.

2 The Associative Model Of Computation

The associative computing model (ASC) is an extension of the general associative processing techniques developed for the associative STARAN SIMD computer in the 1970's for massively parallel computation. As our algorithms will demonstrate, ASC provides an efficient computational model for algorithms requiring massive parallelism. Details of how this model can be implemented on certain SIMD computers are given in

[6]. Also, a high level language based on ASC detailed in [6] has been installed on the STARAN, ASPRO, WaveTracer, and Connection Machine CM-2.

A brief summary of the features of the ASC model is presented here. Additional information and properties of this model may be found in [5]. ASC consists of an array of cells, each containing a processor and its local memory. Cell memory holds variables used for data-parallel operations. These cells are connected by bus to the instruction stream (IS) which stores a copy of the program being executed and broadcasts program instructions to all active cells. The general ASC model described in [5] allows multiple instruction streams (MASC). It is convenient to assume that variables and constants that need to be globally available to all cells are stored in the memory of the IS and may be broadcast to all active cells. The IS also has the ability to read and store a value from a specific cell. The IS variables are called sequential or global variables and PE or cell variables are called parallel variables. In addition to data-parallel execution, the ASC model supports constant time functions for associative searching and selection, logical operations, and maximum and minimum. Constant time searching permits the simultaneous examination of all active cells and the identification of all those that meet the search criteria. These identified cells are called responders and become the new set of active cells. By altering the criteria, different cells become responders. The IS has the ability to detect the presence of responders in unit time. It is also possible to access each active cell sequentially and to return to the set of cells which were active preceding the search or to activate all cells. The maximum or minimum value of a parallel variable (or the cell address containing that value) can be computed for all active cells in constant time. While not used here, the cells may be connected by means of an interconnection network.

3 Jarvis March

The Jarvis March [4] algorithm computes the convex hull of a set S of planar points by identifying the hull edges. The overall time complexity is $O(nh)$, where h is the number of vertices in the convex hull. In the worst case, h equals n and the complexity is $O(n^2)$.

The Associative Jarvis March algorithm in Figure 1 assumes the ASC model (i.e., only one IS). In this algorithm, the variables *extreme*, *rank*, *slope*, and *maxslope* are parallel PE variables while the rest of the variables are IS variables. The vertices of the convex hull are stored in an ordered list L stored in the IS and in the PEs in a ranking order. The convex hull vertices are also identified in the boolean variable, *extreme*, and their order in the list L is identified in the variable, *rank*. The first part of the algorithm finds the upper convex hull and is given in Figure 1. The second part is similar and finds the lower convex hull. Since the second part can be accomplished by a simple modification to the first part, it is omitted.

Algorithm Associative Jarvis March
Input: A set S of coordinate points.
Output: An ordered list of the vertices of the upper convex hull.

1. Initialize $j = 0$.
2. All PEs are used to compute *xmax* and *xmin*, the maximum and minimum value of the x-coordinate of S.
3. Restrict the PEs to the one whose point(x,y) satisfies $x = xmin$. This PE is marked *extreme* and *rank* is set to j. The IS processor assigns $xcurrent = x$, $ycurrent = y$ and $L[j] = (x, y)$.
4. While $xcurrent < xmax$ (Compute the upper hull of S)
   (a) $j = j + 1$
   (b) Restrict the active PEs to the points (x,y) satisfying $x > xcurrent$; those PEs assign $slope = \frac{y-ycurrent}{x-xcurrent}$
   (c) All active PEs are used to compute *maxslope*, the maximum slope value of all active PEs.
   (d) Restrict the active PEs to the point (x,y) satisfying $slope = maxslope$. This PE is marked *extreme* and assigns $rank = j$. The IS processor assigns $xcurrent = x$, $ycurrent = y$ and $L[j] = (x, y)$.

Figure 1: Associative Jarvis's March Algorithm.

If h is the number of vertices in the convex hull, then the running time of the algorithm in Figure 1 is $O(h)$. Of course, $h = n$ in the worst case and the running time is $O(n)$. Since there is a PE for each point, the cost of this algorithm is $O(nh)$ or $O(n^2)$ in the worst case. If we assume h is on average $O(\log n)$, as justified in Section 1, then this algorithm has $O(\log n)$ average running time and $O(n \log n)$ average cost, which is optimal. Assuming h is $O(\log n)$ for the average case this algorithm has $O(\log n)$ average running time and $O(n \log n)$ average cost, which is optimal.

4 Quickhull - ASC

The Quickhull algorithm is an adaptation of the Quicksort algorithm, a recursive method which uses the divide-and-conquer approach. The original problem is divided into two subproblems, each subproblem

is solved recursively, and the solutions are merged to produce the overall solution.

Let S be a set of n planar points that are stored at most one point per PE. Each point has its two coordinates stored in the PE variables x and y. Also, each PE has two variables called *area* and *pointer*. The rest are sequential variables. These include a FIFO edge list Q which is used to store the endpoints of potential edges of the convex hull as they are located.

Let e be the extreme point of S with the largest x coordinate. Also, let w be the extreme point of S with the smallest x coordinate. The IS keeps a linked list L which is used to store the convex hull points in a clockwise order starting at w. The vertices of the convex hull can be listed in sorted order by starting at w and following the forward pointer to the next vertex until e is reached. The first part of the algorithm finds all the convex hull points above $\overline{we}$ from w to e and is given in Figure 2. The second part finds the convex hull points below $\overline{we}$ from w to e. Since the second part can be accomplished by a simple modification of the first part, it is omitted

Since each iteration for the algorithm in Figure 2 produces a convex hull point, exactly h iterations are required and the running time is $O(h)$, where h is the number of convex hull points. If we assume h is on the average $O(\log n)$, as justified in Section 1, then this algorithm has $O(\log n)$ average running time and $O(n \log n)$ average cost, which is optimal.

5 Quickhull - MASC

If multiple ISs exist, then the Quickhull subproblems can be subdivided among these ISs. Initially, IS_1 will process the first edge $\overline{we}$. When finished, it will place $\overline{re}$ on a job stack and start processing $\overline{wr}$. An idle IS, say IS_2, can pop $\overline{re}$ off the stack and process it. Upon completion, IS_2 will have recursively determined the polygonal chain of convex hull edges from the points above $\overline{re}$ and returned these to IS_1 so that IS_1 can concatenate them with the polygonal chain hull edges above $\overline{wr}$ to obtain the convex hull above $\overline{we}$. This procedure is general and can be repeated recursively at each level.

As justified in Section 1, it is reasonable to assume that in the average case, there are $O(\log n)$ convex hull points. In addition, we assume that in an average case, the remaining unidentified hull points are more or less evenly distributed among the partitions in each recursion level. Since each partition of the problem will identify a unique convex hull point, in k recursion levels, $O(2^0 + 2^1 + ... + 2^k)$ or $O(2^k)$ convex hull points will be identified. Eventually $O(2^k) = O(\log n)$, all convex hull points should be found while k is $O(\log \log n)$. If $O(\log n)$ ISs are available, each level of recursion can be computed in constant time. Then the average running time of

Algorithm Associative ASC-Quickhull
Input: A set S, of points given as (x, y) coordinates.
Output: A linked list L of the vertices of the upper convex hull.

1. All PEs are used to compute $xmax$ and $xmin$, the maximum and minimum values of the x-coordinate of S, respectively.
2. Restrict the PEs to the one whose point(x,y) satisfies $x = xmin$. This PE stores w.
3. Restrict the PEs to the one whose point(x,y) satisfies $x = xmax$. This PE stores e.
4. The IS places $\overline{we}$ into the FIFO list Q.
5. The IS adds w and e to L and sets *pointer* in w to e.
6. While Q contains edges (Compute the upper hull of S)
   (a) Remove $\overline{pq}$ from top of list Q.
   (b) Make the active PEs those whose point are above $\overline{pq}$.
   (c) For each active PE, assign $area$ = (area of triangle prq) where r is the point (x, y) held in that PE.
   (d) All active PEs are used to compute $maxarea$, the maximum area value of all active PEs.
   (e) Restrict the active PEs to the one satisfying $area = maxarea$. If multiple triangles have the same maximum size, the one whose $\angle prq$ is maximal is selected. The IS perform the following:
      i. Add the selected point r to L.
      ii. Sets *pointer* in the PE containing r to q.
      iii. Places two edges $\overline{pr}$ and $\overline{rq}$ into the FIFO list Q.
   (f) Make the active PE the one containing p and reset its *pointer* to r.

Figure 2: Associative Quickhull Algorithm.

this algorithm is $O(\log \log n)$ and the average cost is $O(n \log \log n)$.

6 Implementation Results

The preceding algorithms except for Quickhull-MASC algorithm were implemented on the WaveTracer (DTC) SIMD Computer [1, 2]. The WaveTracer automatically provides virtual processors if more physical processors are required. With each actual processor supporting either k or $k-1$ virtual processors. This causes a constant slowdown in actual running time of approximately $\frac{1}{k}$. The results are shown in Table 2 for sets from 100 to 20000 points in size. These points were obtained by randomly generating their x and y coordinates in the range 0 to 5000. The result in Table 2 is obtained for each input size by taking the mean of the time to compute the convex hull points from over 60 different random generated data sets. Since there is no built in clock for the WaveTracer, the running time for each random generated data set is obtained by timing 100 executions of the algorithm and then dividing by 100. The running time obtained did not include the time for loading the data into the PEs or outputting the results. The performance of these two algorithms was compared to an associative algorithm based on the Graham Scan in [1, 2]. In [1], a graph based on Table 2 indicate that these algorithms have an average running time of $O(\log n)$ or better.

Input Size	Graham Scan	Jarvis March	Quick Hull
100	.21	.14	.17
500	.33	.15	.23
1000	.37	.21	.26
3000	.39	.20	.29
5000	.42	.22	.31
10000	.44	.25	.35
15000	.45	.27	.30
16000	.42	.28	.34
20000	.32	.35	.36

Table 2: Time in Seconds to compute the Convex Hull for n points.

Our implementations require no specific preordering of the vertices, nor does it need sorting as part of the computation. Also, no network is used.

7 Summary

Parallel versions of the Jarvis March and Quickhull Algorithms designed for the associative computing model are presented in this paper. These algorithms assumes n processors and have $O(\log n)$ average case cost and $O(n^2)$ worst case cost. We implemented these algorithm on the WaveTracer (DTC) Computer and ran extensive tests on them. These tests confirm that these are an excellent algorithm for the associative model. One advantage of these algorithms is that they do not require the use of network operations, which are known to be much slower than local operations [3].

References

[1] Atwah, M. "Computing the Convex Hull on the Associative Model." Masters Thesis. Kent State University 1994.

[2] Atwah, M., J. W. Baker and S. Akl. "An Associative Implementation of Graham's Convex Hull Algorithm." Proceedings of the *Seventh IASTED/ISMM International Conference, Parallel and Distributed Computing And Systems.* 273-276, Washington D.C., October 1995.

[3] Gibbons, P.B. "Asynchronous PRAM Algorithms." In J.H. Reif, editor, *Synthesis of Parallel Algorithms.* 22, 957-997, Morgan Kaufmann Publishing, San Mateo, CA 1993.

[4] Jarvis, R. A. "On the Identification of the Convex Hull of a Finite Set of Points in the Plane." *Information Processing Letters.* 2: 18-21, 1973.

[5] Potter, J., J. Baker, A. Bansal, S. Scott, C. Leangsuksun and C. Asthagiri. "ASC - An Associative Computing Paradigm." Special Issue on Associative Processing, IEEE Computer. 27(11): 19- 25, 1994.

[6] Potter, J. *Associative Computing: A Programming Paradigm for Massively Parallel Computers.* Plenum Publishing, New York. 1992.

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

[8] Rényi, A. and R. Sulanke, Über die konvexe Hülle von n zufällig gewählten Punkten, I and II, Zeitschrift für Wahrscheinlichkeitstheorie 2, 1963, 75–84; 3, 1964, 138-147.

[9] Ulm, D., J. W. Baker. "Virtual Parallelism By Self Simulation Of The Multiple Instruction Stream Associative Computer." Proceedings of the *International Conference On Parallel and Distributed Processing.* 3: 1421-1430, Sunnyvale, Calif. August 1995.

[10] . "The MultiC Programming Language." January 6, 1991.