AN ASSOCIATIVE DYNAMIC CONVEX HULL ALGORITHM

MAHER M. ATWAH
Mathematics and Computer Science
Kent State University
Kent, OH 44242
matwah@mcs.kent.edu

JOHNNIE W. BAKER
Mathematics and Computer Science
Kent State University
Kent, USA, OH 44242, USA
jbaker@mcs.kent.edu

Abstract

This paper presents a new parallel algorithm for the dynamic convex hull problem. This algorithm is a parallel adaptation of the Jarvis March Algorithm. The computational model selected for this algorithm is the associative computing model (ASC) which supports massive parallelism through the use of data parallelism and constant time associative search and maximum functions. Also, ASC can be supported on existing SIMD computers.

1 Introduction

The convex hull of a finite set of a set S of n planar points is an important geometric concept. It can be defined as the smallest convex polygon for which each point in 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 algorithm easier to describe. However, the algorithm given in this paper can be easily modified to eliminate the necessity of these assumptions. 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.

The Jarvis March [7] 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 $O(n^2)$.

2 Associative Computing Model

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 algorithm will demonstrate, ASC provides an efficient computational model for algorithms requiring massive parallelism. Details of how this model can be implemented on existing SIMD computers are given in [10]. In particular, a high level language based on ASC detailed in [10] 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 [9]. 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. For our algorithm, only one IS is required, but the more general ASC model described in [9] allows multiple instruction streams. 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 variables and 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 active cells 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 a simple network.

3 Dynamic Convex Hull

In this section we introduce a dynamic convex hull algorithm. This algorithm is based on parallel adaptation of the Jarvis March algorithm [5]. The new algorithm refines the results obtained by Chazelle [6].

This problem can be stated as follows: Given a number of points that are moving in Euclidean space, we want to maintain for this set of points the convex hull. Each of the convex hull algorithms we have examined thus requires all of the data points to be present before any processing begins. In many geometric applications, particularly those that run in real-time, this condition cannot be met and some computation must be done as the points are being received. In other words, we call an algorithm that cannot look ahead at its input off-line, while one that operates on all the data collectively is called on-line.

A dynamic convex hull is needed [12], when a population is to be estimated using statistics [7, 8], or simulating chemical reactions. In addition, a dynamic algorithm is needed in applications such as graphics, air traffic control, and robotics.

To obtain dynamic algorithm for convex hull, we must make substantial modification to the static algorithms presented [5]. We first state the requirements for a dynamic algorithm. Following Chazelle [6], we need to support four operations:

1. Insert a point M.
2. Delete a point M.
3. Report all the vertices of the convex hull in some reasonable order.
4. Determine whether an arbitrary point M lies inside or outside the convex hull.

Note that in operation 1 the point M can be either a new point or a point that is already in the structure. Also, the operation "delete point M" always refers to a vertex of the convex hull.

If we do not need to support deletions, it is quite easy to make our algorithms dynamic since we can continue to discard points which are not extreme points of the hull. However, if points are to be deleted, it is possible that some non-hull points will later become extreme points of the hull, and in this case we can no longer eliminate any points entirely. With a dynamic algorithm, the number of processors needed is determined by the maximum size of P, where P is the number of processors available. In the discussion that follows, we use N to indicate the largest number of points that will be in P at any given time. Since our algorithm is using one point per processor N will be equal to P.

4 The Dynamic Algorithm

This algorithm considers only the upper convex hull; the lower hull is the same as the upper hull with minor modifications. We have to note that when a point is marked for deletion its value is still stored in that PE and it can be retrieved by marking that PE not deleted.

Let S be the current set of n planar point that are stored in the local memory of the PEs with at most one point per PE. Each point p has its two coordinates x and y stored in the PE variables. 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.

A minimum of three points is needed for the algorithm to work. As the first point or points initially are entered they are stored one per processor in the array, the static algorithm given in [5] is used to compute the upper hull. When a point is deleted, its value is simply replaced by **null** in its own processor. New points being entered are assigned to a processor with a **null** value. As long as the total number of points does not exceed N, there will be no overflow.

After that, whenever a points M is inserted into P the following steps takes place:

1. If M is below $\overline{we}$ then mark it for deletion.
2. If there is a point p equal to M then replace M by **null**.
3. If step 1 & 2 fails then
   (a) Find the greatest lower bound point (call it *glb*) and the lowest upper bound point (call it *lub*) from the set of the current convex hull points.
   (b) If M is below $\overline{glb, lub}$ then mark this point for deletion.
   (c) Else
      i. Activate all PE such that the x-coordinate is less than the x-coordinate of M.
      ii. Calculate the slope of all the active PEs with respect to M.
      iii. Restrict the active PEs to the one satisfying $slope = minslope$. This PE is called L and marked *extreme*.
      iv. All the PEs that are below $\overline{LM}$ are marked for deletion.
      v. Activate all PE such that the x-coordinate is greater than the x-coordinate of M.
      vi. Calculate the slope of all the active PEs with respect to M.
      vii. Restrict the active PEs to the one satisfying $slope = maxslope$. This PE is called G and marked *extreme*.
      viii. All the PEs that are below $\overline{MG}$ are marked for deletion.

Note that the greatest lower bound point for a point M is a convex hull point with its x coordinate value is smaller than the x coordinate of M but larger that all the x coordinate values of all the convex hull points that are to the left of M see Figure 2). Also, the lowest upper bound point for a point M is a convex hull point with its x coordinate value is larger than the x coordinate of M but smaller that all the x coordinate values of all the convex hull points that are to the right of M see Figure 2).

Figure 2 gives an example of inserting a point M. The dotted lines represent the old part of the convex hull and the solid lines represent the new hull after M has been inserted. Notice that *glb* and *lub* points are marked for deletion since they fall below $\overline{LM}$ and $\overline{MG}$, respectively.

All the above steps require constant time. So, inserting a point to the upper convex hull cost $O(1)$ time.

Point queries are very simple with this scheme and it is performed as follows:

1. If M is below $\overline{we}$ then it is outside the upper hull
2. Else
   (a) Find the greatest lower bound point (call it *glb*) and the lowest upper bound point (call it *lub*) from the set of the current convex hull points.

(b) If M is below $\overline{glb, lub}$ then it is inside the upper convex hull, otherwise it is outside the upper convex hull.

As point insertion, point query cost $O(1)$.

Deleting a point is different than inserting a point since, if points are deleted, it is possible that some non-hull points will later become extreme points of the hull. So, if there is a request to delete a point M from S the following steps take place (M is a vertex of the convex hull):

1. Replace M by **null**.
2. Find the greatest lower bound point (call it glb) from the set of the current convex hull points.
3. Activate all the PE such that the x-coordinate is greater than the x-coordinate of glb.
4. If any one of these PEs was marked for deletion then mark it not deleted.
5. Run the static algorithm Figure 1 ([5]) on the set of active PEs to recompute the upper hull.

All the steps cost $O(1)$ except step 5, which in the worst case will cost $O(h)$, where h is the current number of the vertices of the convex hull.

Reporting all the vertices of the convex hull in clockwise (or counterclockwise) order cost $O(h)$, where h is the current number of the vertices of the convex hull. Point reporting has the same cost as sorting. First, we activate all the PEs that are marked *extreme*. Then, we pick the point that contain the smallest x-coordinate and so on until all the extreme points are reported.

This algorithm cannot freely mix insertions and deletions with queries and reports. Queries and report requests cannot be entered after a series of insertions and deletions until the new hull has been completely calculated, and all queries and reports must be completed before a batch of insertions and deletions can be entered.

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 March Algorithm.

5 Conclusion

A parallel dynamic convex hull algorithm designed for the associative computing model is presented in this paper. This algorithm refines the results obtained in [6]. In [6] all operations except point reporting requires $O(n^{\frac{1}{2}})$ time and point reporting requires $O(1)$ time. In our algorithm, point insertion and query takes $O(1)$ time and point deletion and reporting take $O(h)$ time, where h is the number of the vertices of the convex hull. In the worst case, h equals n and time reporting and deletion is $O(n)$. One advantage of this algorithm is that it did not require the use of network operations, which are known to be much slower than local operations [11].

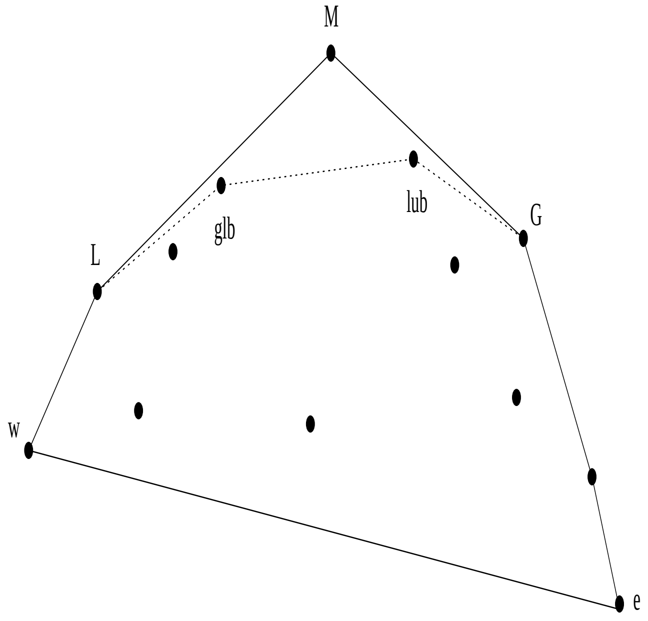

Figure 2: Dynamic Convex Hull: After inserting a point M.

References

[1] S. G. Akl and G. T. Toussaint, Efficient Convex Hull Algorithms for Pattern Recognition Applications, Proceedings of the *4th International Joint Conference on Pattern Recognition,* Kyoto, Japan, 1978, 483-487.

[2] S. G. Akl, A Constant-Time Parallel Algorithm for Computing Convex Hulls, *BIT*, 22, 1982, 130-134.

[3] S. G. Akl. *Parallel Computational Geometry.* Prentice Hall, Englewood Cliffs, New Jersey. 1992.

[4] S. G. Akl and K. A. Lyons, *Parallel Computational Geometry*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[5] M. M. Atwah, J. W. Baker, and S. Akl, An Associative Implementation of Classical Convex Hull Algorithms, *Proceedings of Eighth IASTED International Conference on Parallel and Distributed Computing and Systems*, Chicago, IL, October 1996, 435-438.

[6] B. Chazelle, Computational Geometry on a Systolic Chip, *IEEE Trans. Comp.*, C-33(9), 1984, 774-785.

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

[8] M. H. Overmars and J. Van Leeuwen, Dynamically maintaining configurations in the plane, *Proc. 12th Annual SIGACT Symp.*, Los Angeles, CA. May 1980.

[9] J. Potter, J. Baker, A. Bansal, S. Scott, C. Leangsuksun and C. Asthagiri, ASC: An associative computing paradigm, IEEE Computers, November 1994, 19-25.

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

[11] J. H. Reif, Editor, Asynchronous PRAM Algorithms, *Synthesis of Parallel Algorithms*, 22, 957-997, Morgan Kaufman Publishing, San Mateo, CA 1993.

[12] M. I. Shamos, Computational Geometry, Ph.D. dissertation, Yale University, 1978.