An Associative Implementation Of Graham's Convex Hull Algorithm

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 presents a new parallel algorithm for the convex hull problem. This algorithm is a parallel adaptation of the Graham Scan 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. This algorithm requires $O(n)$ space, $O(n \log n)$ average cost, and $O(n^2)$ worst case cost. The algorithm has been implemented and tested on random data.

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 Graham Scan Algorithm [1] is an important sequential algorithm used for computing the convex hull. It requires $O(n)$ space and has a worst case time of $O(n \lg n)$ (i.e. $n \log_2 n$) due to an initial sorting step which sorts the points in Cartesian coordinates.

The parallel version of the Graham Scan presented here appears to be the first parallel version of this algorithm. Like its sequential version, it also examines three points at a time and checks if the middle point can be eliminated as a possible convex hull point. Also, points initially kept may be eliminated later during backtracking. This algorithm matches the average cost $O(n \lg n)$ of the Graham Scan but has a worst case cost of $O(n^2)$. This algorithm has been implemented on a SIMD computer and tests on random data support the $O(n \lg n)$ average cost.

The remainder of this paper is partitioned as follows. Section 2 describes the associative model. In Section 3, a parallel adaptation of the sequential Graham Scan for the Associative Model [2] is presented. Section 4 presents the timing results for the new convex hull algorithm given in this paper which have been implemented on the WAVETRACER (DTC) [3]. The final section contains a summary.

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 [4]. In particular, a high level language based on ASC detailed in [4] 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 [2]. 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 [2] 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.

Associative Adaptation of the Graham Scan

An associative version of Graham's Algorithm is presented next. This algorithm is like the original Graham Scan in the sense that it examines three points at a time and determines if the middle point can be deleted as a possible convex hull point and that points initially kept may be deleted as a middle point during backtracking later. Note that the Associative Graham Scan checks the candidates for convex hull points on both the right and left sides using the three successive point technique used by the sequential Graham Scan. All point to the left and right of the current x-min or x-max, respectively, in parallel checks itself for deletion with respect to the current x-min or x-max and its past x-min or x-max, respectively. If one or both of the previous x-min or x-max is deleted then the pointer to the previous x-min or x-max in the current x-min or x-max, respectively, is reset recursively to point to the grandparent x-min or x-max, respectively. However, if additional old x-min or x-max points is deleted, the current x-min or x-max updates its previous x-min or x-max pointer to the most recent x-min or x-max point, respectively, that remain undeleted. This provides the backtracking aspect of the sequential Graham Scan Algorithm.

Let S be a set of n planar points that are stored in the local memory of the PEs with at most one point per PE. Each point p has its two coordinates stored in the PE variables x and y. Also, each PE stores the variables *uppermax* and *lowermin* and a two boolean variables called *delete* and *processed*. In addition, the IS stores four variables *min*, *max*, *prevmin* and *prevmax*. The variables *prevmin* and *prevmax* are used to store the previous *min* and previous *max*, respectively. Let *min* and *max* be the points with the current smallest x coordinate and the current largest x coordinate in the set of active PEs, respectively. Also, let P_{min} and P_{max} be the PEs that are holding *min* and *max* as their point, respectively. Generally, we will use P_q to denote the PE that is holding q as its point. If a PE is found to contain *min* as its point (i.e., the PE is P_{min} currently), it sets the variable *lowermin* to *prevmin*. Similarly, the variables *uppermax* is set to *prevmax* for each PE that is found to contain *max* as its point. The variables *lowermin* and *uppermax* are set for each P_{min} and P_{max} directly after *min* or *max* have been found and are reset to another point later only if the point they store get deleted. In this case, the new point stored in *lowermin* or *lowermax*, respectively, is the most recent of the previous *prevmin* or *prevmax* point, respectively, which have not been deleted. This associative version of Graham's Algorithm is presented in Figure 1. This algorithm finds all the upper convex hull points from *min* to *max* and it assumes that there are at least three points to start with. The lower convex hull can be found and computed in reverse order by a simple modification of this algorithm. Note that the variables *min*, *max*, *prevmin*, *prevmax*, *lowermin* and *uppermax* hold both the x and y coordinate for a point. It is convenient to also denote the coordinates of *min* as $(xmin, ymin)$ and of *max* as $(xmax, ymax)$.

The Associative Graham Scan finds *min* and *max* of the remaining points each time it loops and it deletes any point that lies below $\overline{min, max}$ (see Figure 2). Next, all the undeleted points that are to the left of *min* are activated. Each active point is deleted if it lies below $\overline{lowermin, min}$ or below $\overline{lowermin, max}$. As shown in Figure 3, p_1 is deleted since it lies below $\overline{lowermin, min}$. Similarly, all undeleted points that are to the right of *max* are activated. If any active point lies below $\overline{min, uppermax}$ or below $\overline{max, uppermax}$, it is deleted. Figure 4 shows that p_2 is deleted since it lies below $\overline{max, uppermax}$. Note that the variable *lowermin* and *uppermax* maybe different for each active point.

This algorithm (see Figure 1) gives the steps of how the upper convex hull is found. To accomplish point deletion we will use a function called *Delete-p*(p_1, p_2, p_3) which takes three lexicographical sorted points p_1, p_2 and p_3 with $p_1 \leq p_2 \leq p_3$ as input. It returns "true" and deletes p_2 if p_2 lies on or below the line connecting p_1 and p_3; otherwise, it simply return "false". The function works by first finding the two slopes m_1 and m_2 of $\overline{p_1 p_2}$ and $\overline{p_2 p_3}$, respectively and tests if m_1 is less than or equal to m_2. The function *Delete-p*(p_1, p_2, p_3) also returns false when two or more of the three points are the same. The Associative Graham Scan algorithm also shows how to sort the output list L. For the purpose of sorting, each PE will need a pointer *next* so that at the conclusion of the algorithm, each vertex on the upper hull can indicate the next vertex on the upper hull. The list L can then be easily constructed after the algorithm executes. The first element of L is w, the leftmost point in the set S. The next element is the value of the *next* variable of the PE storing w. The processor holding this element will contain the third element in its *next* field. The processor holding the last element in L contains $NULL$ in its *next* field.

To establish correctness, it is only necessary to consider deletions under the lines $\overline{min, max}$, $\overline{lowermin, min}$, and $\overline{max, uppermax}$. This algorithm applies the Graham Scan middle point elimination technique with parallel backtracking to both the points selected as *min* on the left and those selected as *max* on the right. A proof similar to [1, page 904-5], adjusted to replace deletion of points

Procedure Associative Graham Scan (S, H)
Input: A set S, of points given as (x, y) coordinates.
output: A list L, of the vertices of the convex hull.

1. Initialize *processed* and *delete* in each PE to false.
2. While there is an unprocessed PE such that its point p is not deleted
   (a) All active PEs are used to locate min and max, the points p in the active PEs whose x-coordinate has the minimum and maximum value, respectively.
   (b) P_{min} and P_{max} sets $processed = true$.
   (c) If it is first time through the while-loop
      i. P_{min} sets $lowermin = min$, P_{max} sets $uppermax = max$ and P_{max} sets $next = NULL$.
      ii. Else P_{min} sets $lowermin = prevmin$ and P_{max} sets $uppermax = prevmax$.
   (d) Activate all PEs that contain $delete = false$
   (e) The SC broadcasts min and max found earlier to all active PEs.
   (f) Restrict the active PEs to those satisfying $xmin < x < xmax$. Each active PE sets $delete = true$ if p lies below $\overline{min, max}$.
   (g) Restrict the active PEs to those satisfying $x \leq xmin$. Each active PE sets $delete = true$ if p lies below $\overline{lowermin, max}$ or below $\overline{lowermin, min}$.
   (h) Restrict the active PEs to those who satisfy $x \geq xmax$. Each active PE sets $delete = true$ if p lies below $\overline{min, uppermax}$ or below $\overline{max, uppermax}$.
   (i) If P_{min} $delete = true$ then the SC sets $min = prevmin$.
   (j) If P_{max} $delete = true$ then the SC sets $max = prevmax$.
   (k) The SC sets $prevmin = min$ and $prevmax = max$ and broadcasts them to all active PEs.
   (l) Restrict the active PEs to those satisfying $x < xmin$ and $delete = false$. From the set of active PEs select the one that hold the largest x value, call that point r. P_{min} sets $lowermin = r$ and P_r sets $next = min$.
   (m) Restrict the active PEs to those satisfying $x > xmax$ and $delete = false$. From the set of active PEs select the one that hold the least x value, call that point r. P_{max} sets $uppermax = r$ and P_{max} sets $next = r$.
   (n) Activate all the PEs that are unprocessed and have $delete = false$.
3. P_{min} sets $next = max$.

Figure 1: Associative Graham Scan Algorithm.

during backtracking with parallel point deletion, can be used to show that the list L (list R) of undeleted points selected as the value of min (respectively, max) maintain the invariant that the points in L (R, respectively) form the vertices of a convex polygon in clockwise (counterclockwise, respectively) order. Let w-$start$ and e-$start$ be the values of max and min, respectively, at the end of the first pass. After at most $\frac{n}{2}$ passes, there are at most two points, so let w-end and e-end be the values of max and min, respectively. At this stage both w-$start$ and w-end are in L and both e-$start$ and e-end are in R. Moreover, all points in R and L lie on or above $\overline{w\text{–}start, e\text{–}start}$ and on or below the line (or point) $\overline{w\text{–}end, e\text{–}end}$. It follows that the points in $L \cup R$ form the upper convex hull of S.

Note that the Associative Graham Scan Algorithm marks two unprocessed points as "processed" during each pass through the WHILE loop, with the possible exception of the last pass. As a result, in a worst case (e.g., if all points of S are vertices of the convex hull), the WHILE loop will be executed $\frac{n}{2}$ times. Since each pass through the WHILE loop requires constant time, this algorithm has $O(n)$ running time and $O(n^2)$ cost in the worst case. However, it seems reasonable to expect that the Associative Graham Scan will normally delete roughly one-half (or at least some constant fraction) of the remaining unprocessed and undeleted points during each pass through the WHILE loop. Based on this, the average running time of this algorithm is $O(\lg n)$ and its average cost is $O(n \lg n)$, which is optimal.

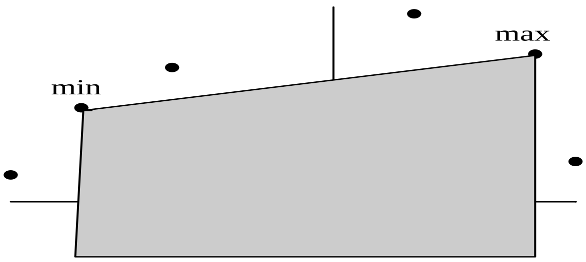

Figure 2: First major step.

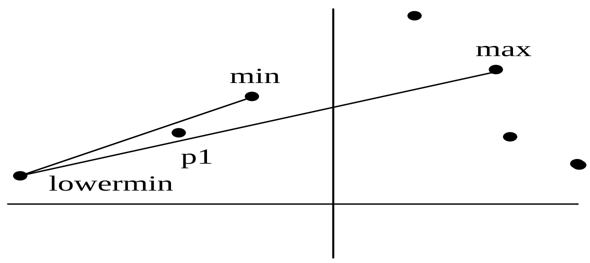

Figure 3: Second major step.

Implementation Results

The preceding algorithm was implemented on the WAVETRACER (DTC) SIMD Computer with 4096

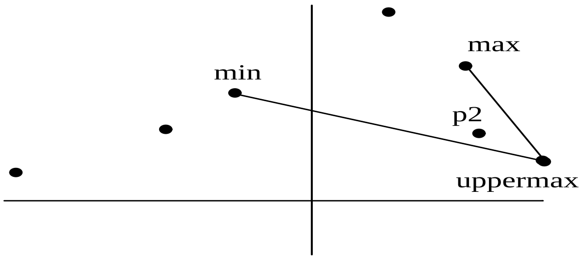

Figure 4: Third major step.

processors [5, 6]. When additional processors are required, the WAVETRACER automatically provides the required number of virtual processors, 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 actual implementation allows us to examine some factors which cannot be determined solely by theoretical analysis. The results are shown in Table 1 for sets from 50 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 1 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 this algorithm is compared to other recently developed associative algorithms for the convex hull in [5, 6]. Notice that if we plot a graph for the timings of the Associative Graham Scan as a function of input size and a graph for the logarithmic graph $y = c \lg n$ with $c = 0.036$ the two graphs closely agree. This was predicted earlier in this paper where it was argued that in the average case the Associative Graham Scan runs in $O(\lg n)$.

Input Size	Graham Scan Time in Sec	Nr. Hull Points	$\log_2$ (Input Size)
100	.21	7	6.6
500	.33	8	9
1000	.37	10	10
3000	.39	12	11.6
5000	.42	12	12.3
10000	.44	13	13.3
15000	.45	13	13.9
16000	.42	14	14
20000	.32	16	14.3

Table 1: Time to compute the Convex Hull for n points.

Summary

A parallel version of the Graham Scan Algorithm designed for the associative computing model with one instruction stream is presented in this paper. This algorithm assumes n processors and has $O(\lg n)$ average case cost and $O(n^2)$ worst case cost. We implemented this algorithm on the WAVETRACER (DTC) Computer and ran extensive tests on it. These tests confirm that this is an excellent algorithm for the the associative model. One advantage of this algorithm is that it does not require the use of interconnection network operations, which are known to be much slower than local operations [7, page 958]. While optimal time is obtained in the average case, its performance degrades less rapidly as the number of convex hull points are increased than associative versions of other classical convex hull algorithms tested. It appears that this is the first parallel version of the classical Graham Scan Algorithm. This algorithm will provide a new algorithm for massively parallel computers that can support the associative model. Moreover, it may be easy to execute this algorithm on other computational models that can support massively parallelism. For example, it is not difficult to see that this algorithm can be computed on a mesh with multiple broadcasts [6]. While in practice, the number of processors in a parallel computer may be too small to store each planar point in a separate processor as required by the associative model, this problem is easily overcome with the use of virtual parallelism [8]. In fact, this solution is supported automatically by the WAVETRACER Computer.

References

[1] Cormen, T. H., C. E. Leiserson, R. L. Rivest. *Introduction To Algorithms.* MIT Press and McGraw-Hill. 1989.

[2] 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.

[3] WAVETRACER. *The MultiC Programming Language.* Preliminary Documentation. January 6, 1991.

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

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

[6] Atwah, M., J. Baker and S. Akl. "Parallel Versions of Classical Convex Hull Algorithms for the Associative Model and Comparisons of their Performances". Paper in progress.

[7] Reif, J. H., editor. *Synthesis of Parallel Algorithms.* M. Kaufmann Publishing, San Mateo, Calif. 1993.

[8] Maresca, M. "Polymorphic Processor Arrays." *IEEE Trans. Parallel Distributed Syst.* 4(5): 490-505, 1993.