Parallel Algorithms for Constructing the Convex Hull in 2-Dimensions

N Asokan
CS 760 Project

1 Introduction

Very little work has been done regarding the development of parallel algorithms to solve problems in computational geometry[2]. Most of the work has been focussed on *parallel random access memory (PRAM)* architectures. A PRAM architecture presupposes that a random access read/write operation (i.e each processing element wants to perform a read/write operation on a specific memory location) can be carried out in $O(1)$ time. In practice, hardware limitations such as bus capacity and fan-in/fan-out restrictions render this impossible. The processing elements and memory modules are connected by an underlying physical network whose interconnections are necessarily limited.

The justification in designing PRAM algorithms is that a PRAM can be simulated on any real architecture. However, often it is possible to obtain lower costs with algorithms designed specifically for the real architecture under consideration.

In this report, two algorithms for computing the convex hull in a 2-dimensional plane on two real architectures, the mesh and the mesh-of-trees are described. The presentation is in somewhat greater detail than in the sources. Some observations are made as to how these could be possibly improved.

2 Description of the Architectures

In both architectures, m^2 processing elements(PEs) are arranged in an $m \times m$ square lattice. The PE in row i and column j is identified as P_{ij}, $1 \le i, j \le m$. It is also identified as P_x, where x is a unique identifier.

The mapping between (i,j) and x is of importance from the point of view of the fundamental operations described later. Usually, it is immaterial from the point of view of higher level algorithms which are built using these fundamental operations. There have been many mapping schemes proposed: row-major ordering, column-major ordering, snake-like ordering etc.

In the mesh architecture, P_{ij} is connected to PEs $P_{i\pm1.j\pm1}$, if they exist. That is, depending on its location, a PE can be connected to 2,3 or 4 neighbours. There are no other interconnections between PEs. In the mesh-of-trees (MOT) architecture, the PEs of a given row (or column) are interconnected to form a balanced binary tree. Figure 1 illustrates these architectures with 16 PEs.

Fig 1. Mesh and Mesh-of-trees

Each PE has its own local memory. However, any PE could read/write on the memory of a neighbour(i.e a PE which has a direct physical connection) in $O(1)$ time. Consequently, a PE can read/write on any other PE given sufficient units of time.

3 Fundamental operations

The number of links in the shortest path between two PEs is the the distance between the two PEs. The largest such inter-PE distance is called the *diameter.* Any operation that calls for the interaction of two PEs is bounded below by the diameter of the architecture. Consequently we have an $O(m)$ lower bound for the mesh and an $O(\lg n)$ lower bound for the MOT, given m^2 PEs.

The following fundamental operations can be used to build algorithms for the specific architectures. D_x in the foregoing refers to a data item in the PE P_x.

- **sorting** Given m^2 data items, distributed one per PE, rearrange them so that the resulting configuration has the data items in ascending order, ie $D_1 \leq D_2 \ldots \leq D_{m^2}$. An $O(n)$ sorting algorithm for the mesh was proposed by Thompson and Kung[7]. This is therefore an asymptotically optimal algorithm. The latest version, with the best constants so far, is given in [6]. An $O(\lg n)$ algorithm for the MOT is implied in [1].

- **broadcasting data within intervals** All PEs in a given interval should receive the same data item, initially stored in one of the PEs in the interval. An interval consists of PEs P_x, with consecutive values of x. It is easy to see that this operation takes $O(d)$ time, where d is the diameter, if there is only one interval containing all the PEs. In [4], it is shown that the same result holds for any number of intervals in the mesh. The same is likely to be true for the MOT as well, though I could find neither a reference nor a proof.

- **Reduction within intervals** Reduction is the combination of data elements using associative operations such as addition, multiplication, boolean AND and OR operations, minimum and maximum. The comments about the costs of broadcasting apply here as well.

- **Concurrent read/write** These operations simulate PRAM capabilities. In concurrent read, each PE contains an address field A_x, containing the address of some PE, and a data field D_x. The result of the operation is to have D_{A_x} in P_x. Concurrent write is similar. Nassimi and Sahni[5] presented algorithms to perform concurrent read/write on hypercube architectures. The same ideas can be extended to any architecture[4]. The dominating component is the sort step. Consequently, the complexity of concurrent read/write is the same as that of sorting.

In addition, it is possible to read from or write to a single PE, presumably from a sequential front-end, in $O(d)$ time.

4 Algorithms

Given a set $S = \{ p_1, p_2, \ldots p_n \}$ of points in a plane, the 2-dimensional convex hull $CH(S)$ is the smallest convex polygon that includes all the points in S. Clearly, the vertices of $CH(S)$ form a subset of S. The object of the algorithm is to produce the vertices of $CH(S)$ in, say anti-clockwise, sequence. The following standard assumptions are made.

- No two points lie on the same horizontal or vertical line.
- No three points lie on any straight line.

4.1 MOT solution

Akl[1] provides an algorithm for MOT architectures to solve this problem. It is based on the following observations.

- Consider the quadrilateral formed by the points $p_{xmin}, p_{ymin}, p_{xmax}, p_{ymax}$ with the smallest x co-ordinate, smallest y co-ordinate, largest x co-ordinate and the largest y co-ordinate respectively. The vertices of this quadrilateral are all in $CH(S)$ and no point contained within the quadrilateral can be in $CH(S)$.[1] Thus we can divide the remaining potential points into four regions.
- A segment $\vec{p_ip_j}$ is an edge of $CH(S)$ if all other points of S lie to the left of $\vec{p_ip_j}$.
- If $\vec{p_ip_j}$ is an edge of $CH(S)$, then the angle $\vec{p_ip_j}$, which $\vec{p_ip_j}$ makes with the positive(or negative) x-axis is the smallest of all such angles $\vec{p_ip}$, where p is a point in the same region as p_i and p_j. Whether to consider the positive x-axis or the negative x-axis depends on the region.

[1]It is possible that the quadrilateral may degenerate into a triangle or a line; but the observation remains valid.

Figure 2 illustrates these observations.

Fig 2. Forming regions and finding the smallest angle.

We now have the following algorithm based on the above:

Assume that the n points are distributed on the n^2 PEs such that P_{ij} contains the co-ordinates of p_j. This can be easily achieved in $O(\lg n)$ time using a broadcast within intervals operation.

- The first four rows compute p_{xmin}, p_{ymin}, p_{xmax}, p_{ymax} in $(O \lg n)$ time. *(reduction in intervals)*
- Broadcast the co-ordinates of these four points to all PEs in row 1 in $O(\lg n)$ time *(broadcast within intervals)*
- Those PEs in row 1 that correspond to these vertices produce a 1 output in constant time.
- All other PEs n row 1 check to see if their point lies within the quadrilateral. If it does, produce a 0 output. If it doesn't, determine the regions that it falls in.
- All PEs in row 1 that has not produced an output, broadcast their region information to all PEs in their columns in $O(\lg n)$ time. *(broadcast within intervals)*

- If the status of p_i is not decided yet, then the PE p_{ij} in row i, in constant time, compute the angle $\vec{p_i p_j}$ makes with positive x-axis(if p_i, p_j are in regions 1 or 2) or negative x-axis(if p_i, p_j are in regions 3 or 4)

- In $O(\lg n)$ time, determine the smallest of these angles. *(reduction in intervals)*

- In $O(\lg n)$ time broadcast the equation of $\vec{p_i p}$, where p is the point that corresponds to the minimum in the last step. *(broadcast within intervals)*

- In constant time, determine if p_j falls to the left of $\vec{p_i p}$.

- Reduce the results of the last step in $O(\lg n)$ time to see if all points fall to the left of $\vec{p_i p}$. If they do, then P_{1i} produces a 1 output; else it produces a 0 output.

- Now that all vertices of $CH(S)$ have been identified, choose an arbitrary point x within or on the quadrilateral and broadcast its coordinates to all the PEs in row 1 in $O(\lg n)$ time. *(broadcast within intervals)*

- All PEs P_{1j} in row 1 compute the angle $\vec{p_j p}$ makes with the positive x-axis and sort the data items on these angle values in $O(\lg n)$ time. *(sorting)*

4.1.1 Analysis

The time complexity of the algorithm is $O(\lg n)$. Even though this is optimal, we used $O(n^2)$ PEs and hence the cost of the algorithm, expressed as a *processor-time product*, is $O(n^2 \lg n)$, which is not optimal because the best sequential algorithm takes $O(n \lg n)$ time.

4.2 Mesh solution

Miller and Stout[4] presented an algorithm to perform this computation in $O(\sqrt{n})$ time. Their algorithm is based on the following observations about merging the convex hulls of two sets X and Y that are divided by a vertical line.

Fig 3 Merging convex hulls.

Let $t, u(t', u')$ be the points with the extreme x-coordinates in $CH(X)(CH(Y))$. The object is to determine the point $p(p')$, which is part of the *upper bridge* between the two hulls.

- pp' must lie above tu.
- if $x(x')$ and y (y') are points the immediately succeed and precede $p(p')$ in $CH(X)(CH(Y))$, then $\vec{yp}(\vec{y'p'})$ has at least some points of $CH(Y)(CH(X))$ above it and $\vec{px}(\vec{p'x'})$ has **all** points of $CH(Y)(CH(X))$ below it.

The algorithms is then as follows:

- Sort the points on x-coordinates.
- Partition the PEs into four equal blocks A, B, C, D based on the above sorting.
- Compute $CH(A)$, $CH(B)$, $CH(C)$ and $CH(D)$ recursively.
- Merge $CH(A)$ and $CH(B)$ to yield $CH(E)$; merge $CH(C)$ and $CH(D)$ to yield $CH(F)$.

- Merge $CH(E)$ and $CH(F)$ to give $CH(S)$.

Merging of two convex hulls $CH(X)$ and $CH(Y)$ is done using a *parallel binary search* technique as follows:

- 1. Find the co-ordinates and relative positions n_t, n_u of the points with the extrem x-coordinates, T and U (T' and U'). *(reduction in intervals)*
- 2. Set t to T and u to U (t' to T' and u' to U')
- 2. Broadcast them to all PEs of $CH(X)(CH(Y))$. *(broadcast in intervals)*
- 3. Identify the potential candidate $p(p')$ with relative position $\lceil \frac{p_t + p_u}{2} \rceil$.
- 4. Identify points x and y.
- 5. Broadcast the equations of $\vec{yp}(\vec{y'p'})$ and $\vec{px}(\vec{p'x'})$ to the PEs of $CH(Y)(CH(X))$. *(broadcasting in intervals)*
- 6. Every PE of $CH(Y)(CH(X))$ determine if their point falls (a) below $\vec{px}(\vec{p'x'})$ and (b) above $\vec{yp}(\vec{y'p'})$.
- 7. Determine if *all* points of $CH(Y)(CH(X))$ fell below $\vec{px}(\vec{p'x'})$ and at least *some* points of $CH(Y)(CH(X))$ fell above $\vec{yp}(\vec{y'p'})$. *(reduction in intervals)*
- 8. If both conditions in 7 were true, then the right $p(p')$ is found; go to step 11.
- 9. Else, if some points of $CH(Y)(CH(X))$ were above $\vec{px}(\vec{p'x'})$ then set $u(u')$ to $x(x')$. Else set $t(t')$ to $y(y')$
- 10. Go to step 2.
- 11. Once $p(p')$ and $q(q')$ have been identified, the new relative positions can be computed in $O(\sqrt{n})$ time. (I.e PEs of $CH(Y)$ read the previous position of q and update relative positions of q' through p'. PEs of $CH(X)$ then read the new position of p' to update the positions of p through T.

4.2.1 Analysis

Let $T(n)$ be the time complexity. Supposing that the indexing scheme used was *proximity ordering*, as illustrated in Figure 4, then A, B, C, D correspond to smaller meshes. Thus the time taken for the computation of the convex hulls in all these partition is $T(\frac{n}{4})$.

Fig 4 Proximity ordering.

Now, in each iteration of the parallel binary search, half the points of $CH(X)$ and half the points of $CH(Y)$ are eliminated. Thus if we can compact the relevant points after every iteration into a tighter square, we could reduce the distance of data movement for the subsequent iteration. In the i^{th} iteration, the algorithm would operate on $O(\frac{n}{2i})$ pieces of relevant data, which can be compressed into a mesh of diameter $O\left(\sqrt{\frac{n}{2i}}\right)$. Therefore the cost of the binary search is:

$$\sum_{i=0}^{O(\lg n)} O\left(\left(\sqrt{\frac{n}{2i}}\right)\right) = O(\sqrt{n})$$

Which yields the recurrence relation:

$$T(n) = T\left(\frac{n}{4}\right) + O(\sqrt{n})$$

the solution of which is $T(n) = O(\sqrt{n})$. The time is optimal, however the cost, $O(n\sqrt{n})$ is not.

5 Comments

The major drawback in the MOT solution is the number of processors. Notice that most PEs remain idle because we are likely to remove a fair number in the first step. Still we need $O(n^2)$ PEs to compute the minimum angle.

If we continue the kind of eliminations in step 1, we may arrive at a better algorithm. Consider the following algorithm based on a sequential algorithm by Greenfield[3]. It suffices to compute the "upper hull". The "lower hull" can be computed likewise. Assume that we have only n PEs, in the form of a balanced binary tree.

- 1. Sort the records according to x-coordinates.
- 2. Set interval i=0 from p_1 to p_n. Produce a 1 output in the first and the last PEs.
- 3. Set p^0_{min} to p_1 and p^0_{max} to p_n.
- 4. Broadcast the equation of $\overrightarrow{p^i_{min}p^i_{max}}$ to all PEs in interval i.
- 5. Tag all PEs in interval i that are to the left of $\overrightarrow{p^i_{min}p^i_{max}}$.
- 6. All untagged PEs produce a 0 output.
- 7. All tagged PEs compute the perpendicular distance of their points to $\overrightarrow{p^i_{min}p^i_{max}}$.
- 8. Determine the largest distance, computed in 7, in each interval.
- 9. Determine the PEs p^i_m corresponding to the largest distance.
- 10. Produce a 1 output in those PEs identified in the last step.
- 11. Set interval $2i$ from p^i_{min} to p^i_m and interval $2i+1$ from p^i_m to p^i_{max}, setting the p_{min} and p_{max} values for the new intervals accordingly.
- 12. If any interval contains more than 2 points, go to step 4.
- 13. Select a point x inside the hull, compute the angle that $x\vec{p}_i$ makes with the positive x-axis for all p_i in the hull and sort the hull points on these angles.

Figure 5 illustrates the above algorithm.

Fig. 5 - A tree algorithm

Most of the steps involve operations within intervals. If these operations take $O(\lg n)$ time as conjectured, then the algorithm would run in $O(\lg n)$ time, which would correspond to a cost of $O(n \lg n)$.

In any case, an operation within intervals can be simulated by performing a global operation as many times as there are intervals. In the worst case, the above algorithm would make $h(\leq n)$ iterations where h is the number of points on the convex hull. This would correspond to a running time of $O(h \lg n)$ and a cost of $O(nh \lg n)$ which is no worse than the algorithm presented in [1].

References

[1] Akl, Selim "The Design and Analysis of Parallel Algorithms", Prentice Hall 1989

[2] Aggarwal, Alok "Research Topics in Computational Geometry", EATCS 39, Oct 1988

[3] Greenfield, Jonathan "Quickhull: A Planar Convex Hull Algorithm" Syracuse University Technical Report 1990

[4] Miller, Russ and Stout, Quentin "Mesh Computer Algorithms for Computational Geometry", *IEEE Trans. Comput.*, 38(3) March 1989 pp 321-340

[5] Nassimi, D and Sahni, S "Data Broadcasting in SIMD Computers", *IEEE Trans. Comput.*, 30(1) 1981 pp 101-107

[6] Park, Arvin and Balasubramanian, K "Reducing Communication Costs for Sorting on Mesh-connected and Linearly-connected Parallel Computers", *JPDC* 9, 1990 pp 318-322

[7] Thomson, C.D. and Kung, H.T. "Sorting on a Mesh-connected Parallel Computer", *CACM* 20(4) Apr 1977 pp 263-271