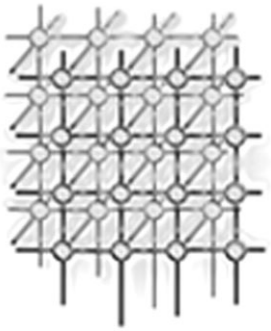
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2006; **18**:1595–1612
Published online 12 January 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1007



Parallel divide-and-conquer scheme for 2D Delaunay triangulation

Min-Bin Chen[1,*,†], Tyng-Ruey Chuang[2] and Jan-Jan Wu[2]

[1]*Department of Management Information Systems, Chung Kuo Institute of Technology, Mucha, Taipei 116, Taiwan*
[2]*Institute of Information Science, Academia Sinica, Nankang, Taipei 115, Taiwan*

SUMMARY

This work describes a parallel divide-and-conquer Delaunay triangulation scheme. This algorithm finds the affected zone, which covers the triangulation and may be modified when two sub-block triangulations are merged. Finding the affected zone can reduce the amount of data required to be transmitted between processors. The time complexity of the divide-and-conquer scheme remains $O(n \log n)$, and the affected region can be located in $O(n)$ time steps, where n denotes the number of points. The code was implemented with C, FORTRAN and MPI, making it portable to many computer systems. Experimental results on an IBM SP2 show that a parallel efficiency of 44–95% for general distributions can be attained on a 16-node distributed memory system. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: Delaunay triangulation; parallel algorithm; unstructured mesh generation

1. INTRODUCTION

Meshes are commonly employed in finite-element and finite-volume analysis in many fields such as computational solid mechanics, computational fluid dynamics, computational electro magnetics and computer graphics. Large point set problems are frequently solved using parallel computers, since their large computation requirement makes the mesh generation beyond the storage and power of a single processor. Therefore, parallel computers are needed to create the mesh of a very large

*Correspondence to: Min-Bin Chen, Department of Management Information Systems, Chung Kuo Institute of Technology, Mucha, Taipei 116, Taiwan.
†E-mail: cmb@mail.ckitc.edu.tw

Contract/grant sponsor: National Science Council; contract/grant numbers: NSC 91-2213-E-163-001, NSC 88-2213-E-001-007, NSC 88-2213-E-001-004 and 890002291-9

Received 27 February 2004
Revised 17 May 2005
Accepted 15 June 2005

data set. Automatic mesh generation has two main classes of methods—Delaunay triangulation and the advancing front method. Delaunay triangulation's max–min angle criterion makes the triangulation a well-shaped mesh. Many Delaunay triangulation sequential algorithms have been presented including the Bowyer–Watson algorithm, quick hull, sweep line, incremental construction and divide-and-conquer (D&C). Among these algorithms, the D&C scheme is the most efficient and powerful for generating Delaunay triangulation [1]. This study uses the D&C scheme in the parallel implementation.

Past literature has included some parallel Delaunay mesh generation methods. Said *et al.* [2] discretized the inter-domain boundary leading to independent grid generation. A grid must exist in the inter-domain boundary between sub-domains. Cignoni *et al.* [3] proposed the parallel implementation of the incremental construction algorithm, which uses the 'marriage before conquest' strategy. Blelloch *et al.* [4] also applied this strategy. Blelloch *et al.*'s algorithm identifies the median line between sub-domains by the convex hull algorithm, then triangulates sub-domain using the D&C algorithm. The parallel efficiency of Blelloch *et al.*'s algorithm at 128k points is 48–90% on an IBM SP2 with eight nodes.

Details of the D&C Delaunay algorithm are reported in [5–8]. This algorithm splits the original point set into small data sets each containing two or three points, then recursively merges the triangulation of small data sets to create the complete Delaunay triangulation. The major complexity of this algorithm is in the merge phase. The merge algorithm performs the merge operation between the upper and lower common tangent of the two sub-convex hulls. D&C's merge operation only works on the neighbor triangulation of the merging area, which has an efficient data structure [9]. However, for the parallel Delaunay triangulation, the continuous operation on neighbor triangulation produces continuous data exchange between processors, which is the main problem in parallelizing D&C schemes [10].

An effective means of preventing massive data exchange between processors is to find the neighbor triangulations which may be modified during the merge operation, and then only exchange those neighbor triangulations. This neighbor triangulation on each processor is called the 'affected zone'. This study devises a novel algorithm to find the affected area. The algorithm's associated time complexity is $O(n)$, where n denotes the number of points assigned to a processor. This work also presents an error-prone line circle test in constructing the affected region.

The rest of this paper is organized as follows. Section 2 presents an overview of the proposed parallelization method. Section 3 presents the theory and the algorithm to determine affected zone. Section 4 discusses the line circle test. Section 5 reports the experimental results with various distributions of the point sets. Section 6 draws conclusions.

2. PARALLEL D&C DELAUNAY TRIANGULATION SCHEME

The D&C Delaunay triangulation algorithm is $O(n \log n)$. This method was chosen because it efficiently merges two triangulations. During the merge operation, the algorithm must access the neighbor triangulation in the merge area, and therefore requires successive exchanges of data between processors to merge between them. The proposed algorithm resolves this difficulty by estimating the affected areas of two triangulations, and then exchanging and merging them. The next section shows the algorithm and the theoretical properties of affected zone. Figures 1–3 depict the parallel merge of block triangulations by decomposing the domain into four sub-domains. The block triangulation data structure is the same as that of Shewchuk's algorithm [9] introduced in Section 3.

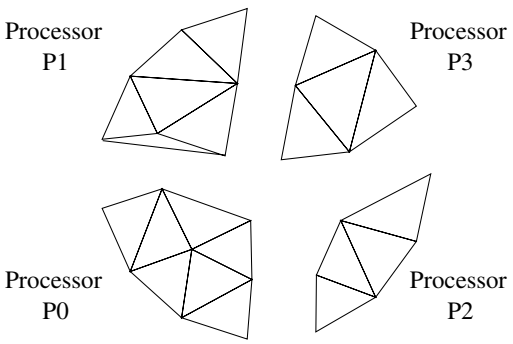


Figure 1. Delaunay triangulation of each block.

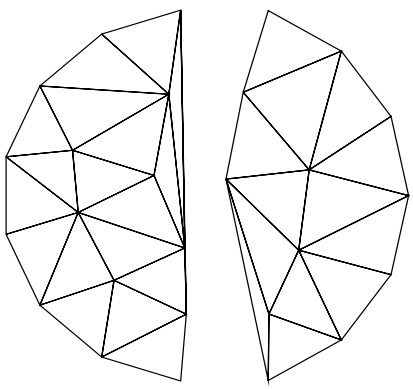


Figure 2. Processors P0 and P2 merge their triangulation with the triangulation from P1 and P3.

At the beginning of the parallel scheme, the point set must be partitioned. A parallel partitioning method of Hoare's median finding algorithm [11] is designed to partition the point set into 2D blocks. In the merging phase of the parallel scheme, the processors in the same column merge parts of the x-direction cut (Figure 1). Only the processors which merge across columns are needed to merge the full y-direction cuts (Figure 2).

This parallel algorithm assumes that the total number of processors are powers of two, and configure them into a 2D array. As shown in Figure 4, the point set is partitioned in the x-direction and distributed to the first processor of each column. Then, the points of every column are partitioned in the y-direction and distributed to the other processors. Every processor sorts its points, triangulates the point set and generates the affected zone. Then, the combination of the triangulations are performed by merging the

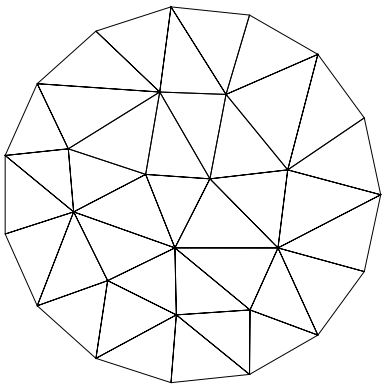


Figure 3. Processor P0 merge its triangulation with the triangulation from P2.

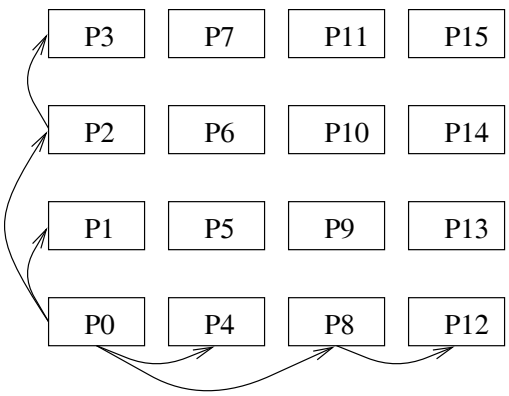


Figure 4. Processors are configured into a 2D array.

affected zones. The merging order is the reverse of parallel partitioning point sequence, therefore the merge needs $O(\log P)$ phases. This method is conducted as follows.

1. Configure the P processors (where P is a power of two) as a 2D array (Figure 4).
2. Partition the whole point set to the first processors of each column.
3. Partition the point set on the first processor to the other processors on its column.
4. Triangulate each block by applying D&C on individual processors.
5. Create the affected zone on individual processors.
6. Merge the affected zones in the reverse order of partitioning the point set and transmit the modified triangles back to the associated processors.

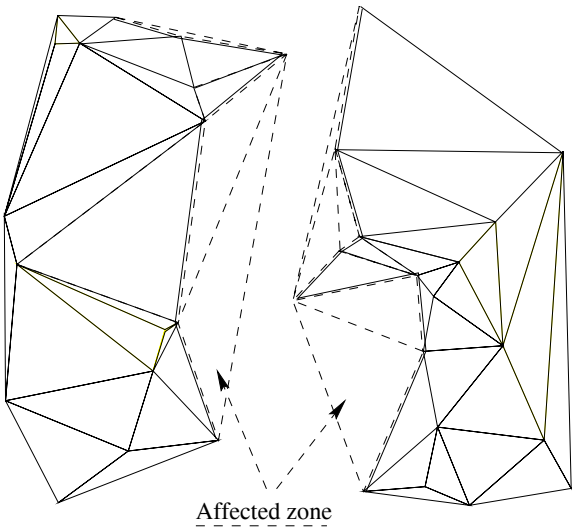


Figure 5. The affected zones of two triangulations.

3. AFFECTED ZONE

The affected zone is the triangulation region (Figure 5) that may be modified when two sub-Delaunay triangulations are combined. Triangulations can be combined between processors by merging the affected regions.

3.1. The data structure of the affected zone

For efficiency, a data structure similar to that of Shewchuk's [9] (Figure 6) was employed for the affected zone. Therefore, the affected zone has two types of triangles, real and ghost zone triangles. Each triangle includes eight pointers, of which three point to nodes, three point to neighbor triangles, one points to the processor containing the triangle, and the last pointer points to the index of the triangle on the processor. The data structure of the block triangulation, like Shewchuk's data structure, only includes the first six pointers. Since one neighboring triangle pointer in the boundary real triangle points to a null triangle outside, a ghost triangle is created instead of a null triangle. A ghost zone triangle has the same data structure as a real zone triangle. The three node pointers of the ghost zone triangle point to one null node (negative or zero value) and two boundary nodes, and the other three triangle pointers point to the boundary triangle and two neighbor ghost triangles. When the two affected zones are merged on the other processor, the last two pointers can be used to address and remove the non-Delaunay triangles on the associated processor.

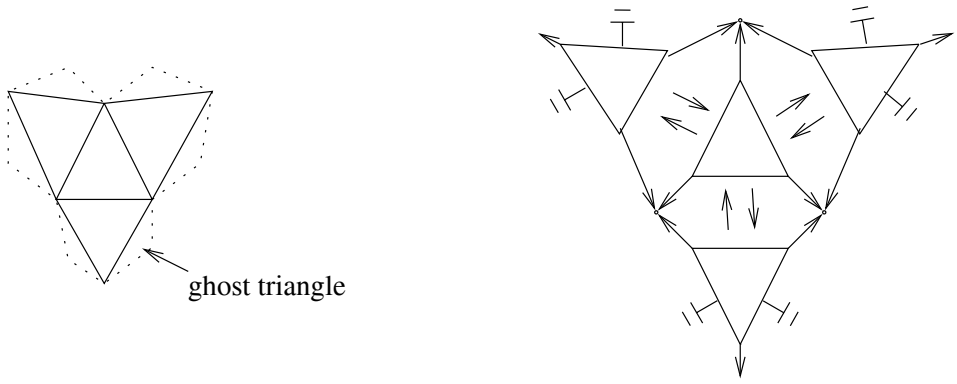


Figure 6. (a) A triangulation with its ghost triangles and (b) its data structure.

3.2. The affected zone theory and algorithm

Before discussing how to find the affected area, two Delaunay triangulation characteristics required to identify the affected region are outlined. The first property is the well-known circumcircle criterion, often applied to construct a Delaunay triangulation. The second rule [6] states that, for a specified direction from a node, the circumcircles of the node's neighboring triangles move monotonically from the farthest circles in the reverse direction.

Fact 1. *A triangulation is Delaunay if and only if the circumcircle of every interior triangle is point-free; that is, none of the given sites are constrained in the circle.*

Fact 2. *Let* $N_1, N_2, \ldots, N_k (k \geq 1)$ *be some Delaunay neighbors of a site* P *and lies above a line* L *in counterclockwise order as in Figure 7. The circumcircle of triangle* $PN_iN_{i+1}, i = 1, 2, \ldots, k-1$, *move monotonically to the left.*

Two problems exist in creating the affected zone. Which triangles should be incorporated into the affected area? How are these triangles identified? Lemma 1, presented below, specifies the triangles to be included in the affected region. Lemma 2 presents the rule for searching the triangles specified in Lemma 1. To compute the union of two block triangulations, the points of the right(left) block are added to left(right) block triangulation together. The two lemmas address the condition that adds the right block points to the left block triangulation, and this condition is the same for the left block points added to the right block triangulation.

Lemma 1. *For a given Delaunay triangulation, new points are added to form a new Delaunay triangulation. Assuming a specified line outside the given triangulation, the newly added points are on different sides of the line. In the given triangulation, the interior triangles whose circumcircles touch the line may not be Delaunay triangles for the new point set. That is, these triangles are indeterminate.*

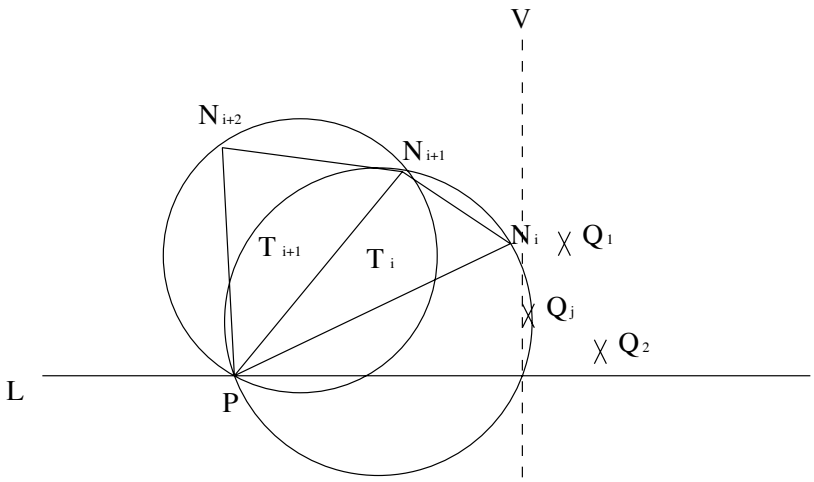


Figure 7. The neighboring circumcircles of a node of Delaunay triangulation monotonically move towards the reverse of a specified direction (right-hand side is the specified direction here).

Proof. According to Fact 1, none of the circumcircles of the interior triangles contain any points in the Delaunay triangulation. Since the newly added points are outside the line, the interior triangles whose circumcircles do not touch the line do not contain these new points, and, therefore, are still point-free. Only the triangles whose circumcircles touch the line may not be Delaunay triangles for the new point set. □

Lemma 1 defines the indeterminate triangles to be included in the affected region. Then, the following Lemma 2 demonstrates the convergence of the affected zone searching algorithm.

Lemma 2. *For a point P in the left block triangulation, let its neighbor triangles* $T_1, T_2, \ldots, T_k (k \geq 1)$ *lie in counterclockwise order (Figure 7). Let* $Q_1, Q_2, \ldots, Q_k$ *denote the point set of the right block, and let line V represent the left-most line of the right block. If the circumcircle of triangle* T_{i+1} *does not touch line V, then the triangle* $T_{i+1}, T_{i+2}, \ldots, T_k (k \geq i+1)$ *is point-free from the point set Q.*

Proof. As in Fact 2, the circumcircles of a point move monotonically to the left. This result follows accordingly. □

Now, the indeterminate triangles can be searched inwardly from the triangles on the convex hull. The affected region can be constructed by discovering all the triangles that pass the line-incircle test. The line-incircle test is described in Section 4. Figure 8 illustrates a simple inward searching process.

The affected zone has three parts, the affected triangle set, the real affected-zone triangle set and the ghost triangle set. The affected triangle set is the set of interior ghost triangles of the affected zone. The real affected-zone triangle is the indeterminate triangle. The real affected-zone triangle contains the same first six pointers as the same position of the triangle of block triangulation. The ghost triangle set is the set of exterior ghost triangles with a zero-value null node. Figure 9(a) depicts an initially empty

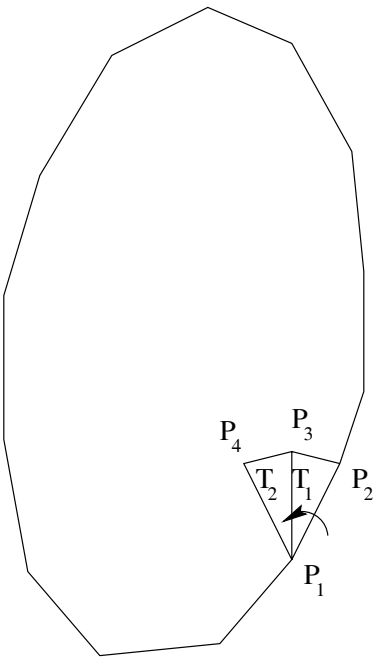


Figure 8. The construction of affected zone begins from the test of neighbor triangles of the sites on convex hull (test T_1 first, then T_2).

affected area and its affected and ghost triangle sets. The triangle in the affected triangle set includes the same six data as the same position triangle of block triangulation except the negative value of the null node. These data are applied to perform the line-incircle test on the triangles. When the triangle of the affected triangle set passes the line-incircle test, it becomes a real affected-zone triangle. The affected triangle set is then updated as in Figure 9(b). If the triangle of affected triangle set does not pass the line-incircle test, this triangle becomes a ghost triangle by setting its null node to zero. Every triangle in the affected triangle set performs the line-incircle test one-by-one in clockwise or counterclockwise order.

If the triangles of the affected triangle set (such as T_3, T_4, T_5 or T_6 in Figure 10) turn into real affected-zone triangles, three cases may result.

1. The new triangle does not contact the interior boundary of the real affected-zone triangle set (T_4).
2. The new triangle shares an edge with the interior boundary of the real affected-zone triangle set (T_3 and T_5).
3. The new triangle shares a point with the interior boundary of the real affected-zone triangle set (T_6).

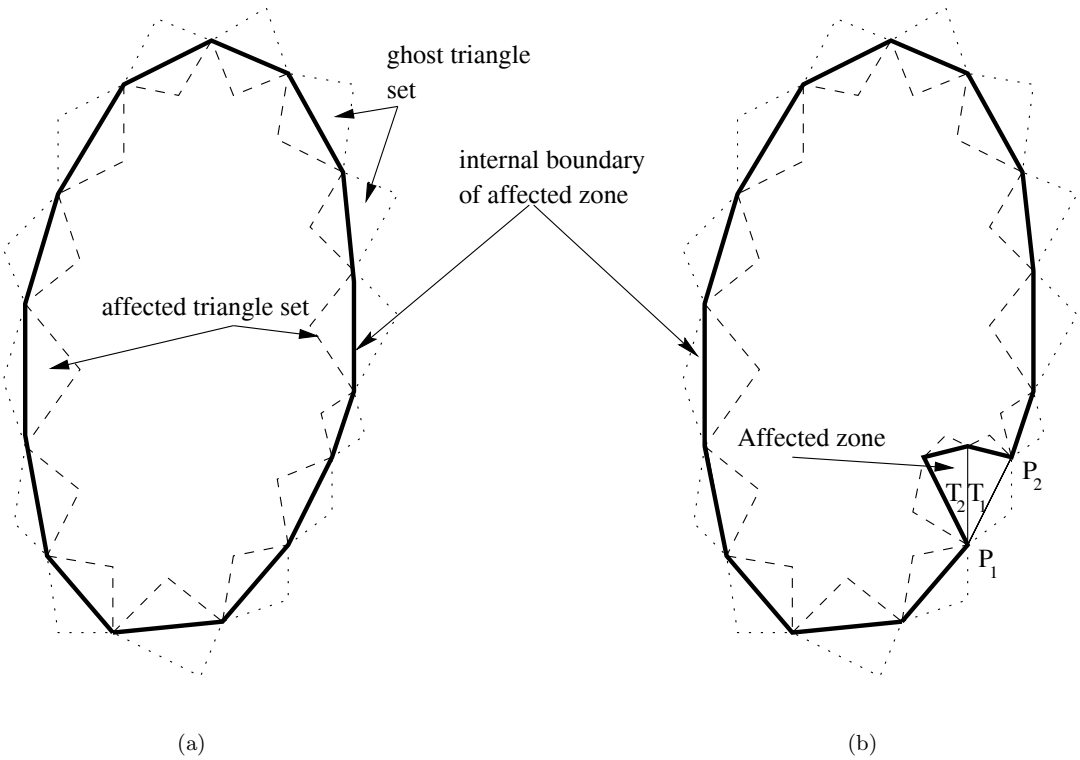


Figure 9. (a) At the initial stage of affected zone algorithm, only the affected triangle set and ghost triangle set exist. (b) The neighbor triangle of affected triangle set around P_1 performs a line-incircle test. After twice passing the test, T_1 and T_2 are the real affected zone triangles in the affected region.

The first case is the most general. For this instance, after T_4 becomes the real affected-zone triangle, two new triangles are added to the affected triangle set. For the second case, $T_3(T_5)$ becomes the real affected-zone triangle, and the other affected triangle on the contacted edge becomes the new affected triangle. Therefore, no new triangle is generated in this case. In the third case, T_6 becomes the real affected-zone triangle, and the affected triangle set is cut into two sets, with a new triangle created in each new sets. The affected zones are generated on each set separately.

ALGORITHM AFFECTED_ZONE

1. Initialize the affected triangle set and the ghost triangle set.
   Choose one triangle in the affected triangle set as the active triangle (Figure 9(a)).
2. Verify whether the circumcircle of the active triangle will touch the dividing line between two block triangulations.

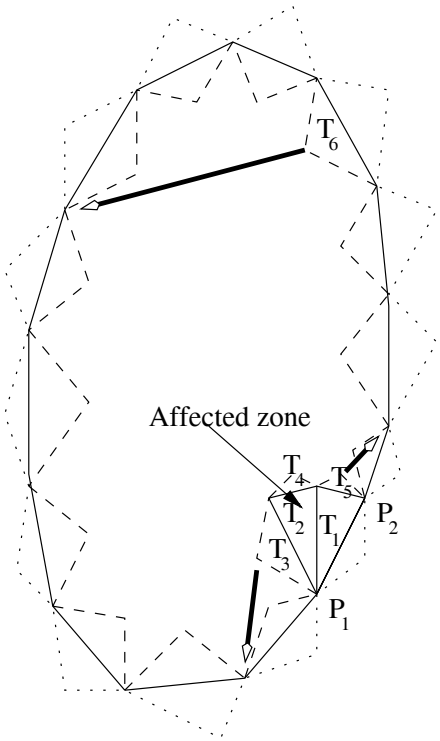


Figure 10. There are three cases when the triangles in the affected triangle set turn into the real affected zone triangles: (i) T_4 does not contact with any triangle of the real affected zone set; (ii) T_3 and T_5 share an edge with the interior boundary of the real affected zone triangle set; (iii) T_6 is the triangle that share a point with the interior boundary of the real affected zone triangle set.

3. IF (the circumcircle of active triangle does not contact the dividing line) THEN
   Turn the active triangle into a ghost affected zone triangle and check whether the affected set is empty. If the affected triangle set is not empty, move to the next active triangle and go to step 2. However, if this affected triangle set is empty and another affected triangle set exists, select one triangle in the next set as the active triangle and go to step 2. If this affected triangle is empty and no other affected triangle set is left, then stop.
   ELSE
   Determine whether the null node of the active triangle (the unsigned value of null node is used) is a point on the interior boundary of the real affected-zone triangle set, and go to step 4.
4. IF (the null node of active triangle is not a point on the interior boundary of the real affected-zone triangle set) THEN
   Change the active triangle into a real affected-zone triangle, add two new triangles to the affected triangle set, then set one of the new triangles as the active triangle and go to step 2.
   ELSE

Verify whether the left or right edge connected to the null node of active triangle is an edge on the internal boundary of the real affected zone. Then, go to step 5.

5. IF (one of the edges connected to the null node of active triangle is the next neighbor edge, e.g. T_5 in Figure 10) THEN

   IF (this affected triangle set only has three affected triangles) THEN

   Turn the active triangle into a real affected-zone triangle, then end this set and search for another affected set.

   IF (another affected triangle set is discovered) THEN

   Choose one triangle in the next affected triangle set as the active triangle and go to step 2.

   IF (no other affected triangle sets exist) THEN

   Stop the AFFECTED_ZONE algorithm.

   IF (this affected triangle set has more than three triangles) THEN

   Convert the active triangle to a real affected-zone triangle. Transform the next triangle in the affected triangle set to an active triangle, and update the associated data, then go to step 2.

   ELSE IF (one of the edges connected to the null node of active triangle is the previous neighbor edge, e.g. T_3 in Figure 10) THEN

   Convert the active triangle into a real affected-zone triangle. Turn the previous triangle in the affected triangle set into active triangle and update the associated data is updated, then go to step 2.

   ELSE

   Change the active triangle into an affected-zone triangle. The affected triangle set is split into two sets and two new affected-zone triangles are added in each set with the appropriate data. Then, select one of the triangles in one of the two sets as the active triangle, and go to step 2.

The above discussion demonstrates that the proposed algorithm can discover all of the triangles which may be changed when two Delaunay triangulations are combined. Since every triangle is tested at most once, the time complexity of the affected-zone algorithm is $O(n)$. Thus, the D&C approach can be applied to merge the affected zones from different processors while reducing the amount of data required to be transmitted between processors.

4. ISSUE IN THE SCHEME

This section discusses the robust line-incircle test for finding the affected area. The algorithm for calculating the affected area involves a line-incircle test, i.e. an incircle test of one circumcircle and the nearest straight line of another triangulation (Figure 11). If the line is inside the circumcircle of the triangle, then the triangle passes the line-incircle test.

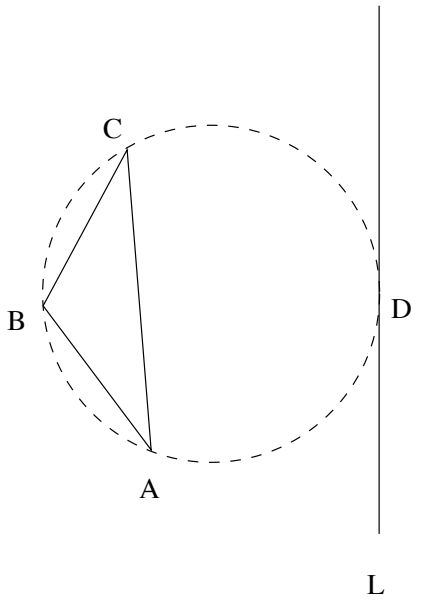


Figure 11. The line-incircle test.

Lemma 3. *In the line-incircle test, let* (x_A, y_A), (x_B, y_B), (x_C, y_C) *be the coordinates of point* A, B, C *on the tips of triangle and* x_D *is the x-coordinate of the line:*

$$\alpha = \begin{vmatrix} y_A & x_A^2 + y_A^2 & 1 \\ y_B & x_B^2 + y_B^2 & 1 \\ y_C & x_C^2 + y_C^2 & 1 \end{vmatrix}$$

$$\gamma = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix}$$

$$\beta = \begin{vmatrix} x_A & x_A^2 + y_A^2 & 1 \\ x_B & x_B^2 + y_B^2 & 1 \\ x_C & x_C^2 + y_C^2 & 1 \end{vmatrix}$$

$$\delta = \begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 \\ x_B & y_B & x_B^2 + y_B^2 \\ x_C & y_C & x_C^2 + y_C^2 \end{vmatrix}$$

Then, if

$$\Delta = \beta^2 - 4\gamma(\gamma x_D^2 + \alpha x_D - \delta) > 0$$

the triangle passes the line-incircle test.

Proof. As in Guibas and Stalfi's paper [6], if

$$\begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{vmatrix} = 0 \tag{1}$$

A, B, C, D are cocircular. Assume point D be the contact point of the circumcircle with the line, so the y-coordinate of D is unknown. Then, we expand the above determinant of array by the last row to be

$$-x_D\alpha + y_D\beta - (x_D^2 + y_D^2)\gamma + \delta = 0 \tag{2}$$

or, equivalently,

$$\gamma y_D^2 - \beta y_D + \gamma x_D^2 + \alpha x_D + \delta = 0 \tag{3}$$

From the previous equation, the circumcircle touches the line L if and only if the determinant $\Delta > 0$. Then this lemma is proved. □

5. EXPERIMENTAL STUDIES

5.1. Primary results

The algorithm was executed on an IBM SP cluster system with P655 Node. The IBM SP has a dedicated processor pool of 16 parallel execution nodes. The main program was written in C for its flexibility in memory allocation and management. The Delaunay triangulation and affected zone were written in Fortran 77. The code was compiled using IBM's mpxlf and mpcc compilers with standard code optimization. Data were moved between processors with library calls to MPI (Message Passing Interface) library functions. The program was executed under IBM's poe (parallel operating environment) on top of the AIX 5 operating system. The parallel efficiency values were compared with the uniprocessor running time, which is nearly the same or better than that of the well-known sequential code in Netlib, package Triangle (see Table I).

The code was executed using three different point distribution; uniform, kuzmin, and line singularity distributions. Owing to the memory limitation of a single processor, Table II lists the timings of triangulating 1M to 16M points. All of the timings represent the average CPU time of 20 runs on each case.

The ideal testing case must represent real-world problems. Therefore, both uniform and non-uniform distributions were selected to test the efficiency of the parallel code. Three different distributions were chosen as indicated in Figure 12 and they are described below.

Table I. Elapsed time of package Triangle [9] with command 'triangle -NEX' and our algorithm on uniprocessor IBM SP2 system.

	Uniform		Kuzmin		Line	
Number of points	Triangle	Ours	Triangle	Ours	Triangle	Ours
1M	5.53	5.32	5.37	5.89	5.65	5.89
16M	203.72	105.42	199.80	111.08	215.03	119.08

Table II. Elapsed time (excluding I/O) and relative speedup with three different distributions.

	Number of processors				
Data size	1	2	4	8	16
Uniform distribution					
1M	5.32 (1.00)	2.85 (1.87)	1.51 (3.52)	0.94 (5.66)	0.70 (7.60)
2M	11.34 (1.00)	5.95 (1.91)	3.15 (3.60)	1.95 (5.82)	1.47 (7.71)
4M	25.04 (1.00)	13.12 (1.91)	6.83 (3.67)	4.12 (6.08)	2.99 (8.37)
8M	53.57 (1.00)	27.24 (1.93)	14.56 (3.68)	8.78 (6.10)	6.36 (8.42)
16M	105.42 (1.00)	53.51 (1.97)	28.96 (3.64)	17.31 (6.09)	12.48 (8.44)
Kuzmin distribution					
1M	5.86 (1.00)	3.13 (1.87)	1.64 (3.57)	1.01 (5.80)	0.78 (7.51)
2M	12.20 (1.00)	6.49 (1.88)	3.36 (3.63)	2.07 (5.89)	1.59 (7.67)
4M	25.80 (1.00)	13.83 (1.87)	7.23 (3.57)	4.43 (5.82)	3.26 (7.91)
8M	55.18 (1.00)	29.56 (1.87)	15.49 (3.56)	9.41 (5.86)	6.91 (7.99)
16M	119.08 (1.00)	63.55 (1.87)	32.98 (3.61)	19.77 (6.02)	15.06 (7.91)
Line singularity distribution					
1M	5.89 (1.00)	2.95 (2.00)	1.70 (3.46)	1.04 (5.66)	0.84 (7.01)
2M	12.05 (1.00)	6.17 (1.95)	3.53 (3.41)	2.12 (5.68)	1.73 (6.97)
4M	25.99 (1.00)	13.28 (1.95)	8.15 (3.19)	4.45 (5.84)	3.37 (7.71)
8M	55.10 (1.00)	29.35 (1.88)	18.39 (3.00)	9.48 (5.81)	6.93 (7.95)
16M	111.08 (1.00)	58.56 (1.89)	33.08 (3.36)	19.06 (5.83)	14.05 (7.91)

- *Uniform distribution*. The points are generated randomly in a unit square. It forms a common base to compare with related work. The timing results can be used to contrast those of non-uniform distributions as well.
- *Kuzmin distribution*. It is used by astrophysicists to model the distribution of stars in a flat galaxy. This distribution is highly non-uniform whose density of point falls quickly as r increases. Its accumulative probability function is

$$M(r) = 1 - \frac{1}{\sqrt{1 + r^2}}$$

The radius r is generated by the inverse of the accumulative probability function $r = M^{-1}(X)$ where X is a random number. Then, the point on the circle of radius r can be picked uniformly.

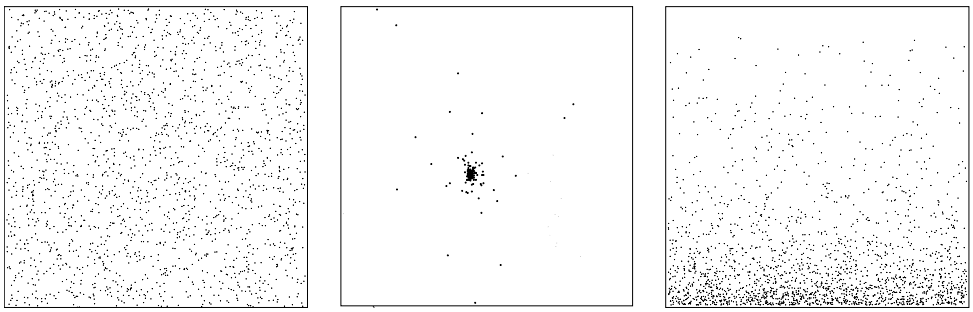


Figure 12. Uniform, kuzmin, line singularity distributions.

- *Line singularity distribution*. This distribution is like the plasma in electromagnetism. Its density is high along a line segment and falls as far away from the line. The points can be generated with the following transformation:

$$(x, y) = \left(\frac{b}{u - bu + b}, v \right)$$

  u and v are two independent, uniform random numbers in range [0, 1]. b is a constant, we set $b = 10.0$ in these experiments.

As Table II demonstrates, the speedup was not linear, because of the partition and communication overheads. The speedup decreased as the number of processors rose, because more levels of recursion were required. This table also reveals that large problems yielded better speed increases than the small problems, since the collective communication of large quantities of data is more efficient than that of small data. The diagonal elements (top-left to bottom-right), which compute the same number of points on each processor, demonstrate that the overhead increased when more levels of recursion were required.

Figure 13 illustrates the parallel efficiency of each processor, defined as the speedup divided by the number of processors, for the uniform and kuzmin distributions. As more processors are added, per-processor performance falls as more levels of recursion are needed.

Figure 14 presents the scalability of our implementation on various numbers of processors. For clarity, the timings of the uniform distribution were compared with those of the opposite non-uniform distribution, i.e. the kuzmin case. From the figures, the performance scales as the problem size becomes larger.

Table III shows the relative costs of the various substeps at processor 0, where every processor triangulates the same number of points but processor 0 is last of the triangulations to be updated. The partition and merge timings incorporate all the partitioning and merging work in processor 0. The 'other' timing in the table is the time spent executing other code, such as changing the index, waiting for or receiving an affected zone, and sending or receiving triangles to be deleted. The sort and triangulation timings remained fairly reasonable as the problem size increased.

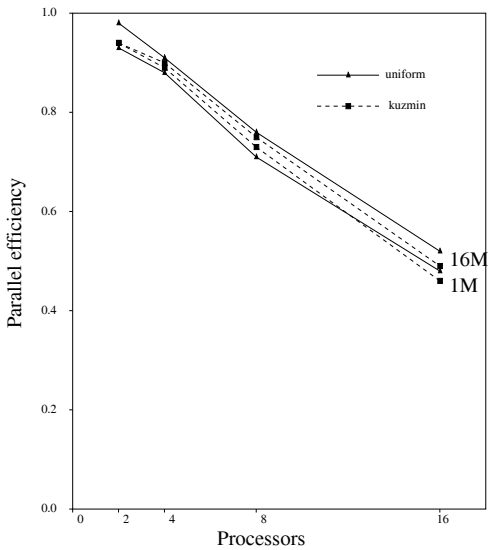


Figure 13. Parallel efficiency of uniform and kuzmin distributions.

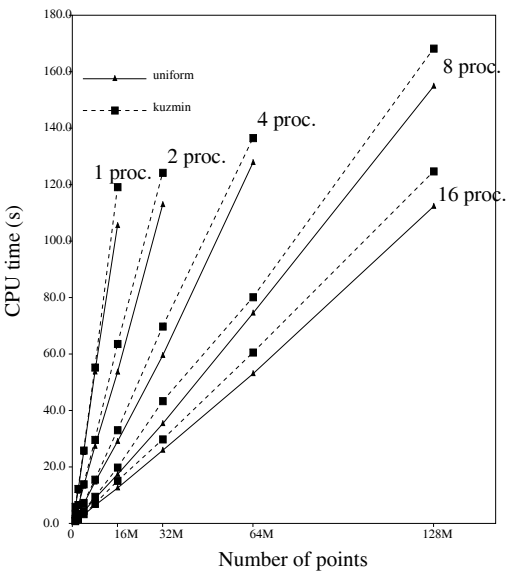


Figure 14. The timings on a fixed number of processors.

Table III. The time and percentage at each substep of processor 0 with three different distributions.

Data	Processors	Partition	Sort	Triangulation	Affected zone	Merge	Other	Total
Uniform distribution								
1M	1	—	0.63 (12%)	4.69 (88%)	—	—	—	5.32
2M	2	0.15 (2.5%)	0.61 (10%)	4.81 (81%)	0.03 (0.5%)	0.002 (0.03%)	0.33 (5.6%)	5.94
4M	4	0.43 (6.3%)	0.64 (9.4%)	4.73 (69%)	0.13 (1.9%)	0.003 (0.04%)	0.88 (13%)	6.82
8M	8	1.10 (13%)	0.63 (7.2%)	4.77 (54%)	0.20 (2.3%)	0.003 (0.03%)	2.07 (23%)	8.78
16M	16	2.94 (23%)	0.67 (5.4%)	4.95 (40%)	0.61 (4.9%)	0.004 (0.03%)	3.30 (26%)	12.46
Kuzmin distribution								
1M	1	—	0.64 (11%)	5.23 (89%)	—	—	—	5.87
2M	2	0.14 (2.2%)	0.61 (9.4%)	5.4 (83%)	0.02 (0.3%)	0.007 (0.1%)	0.31 (4.8%)	6.49
4M	4	0.72 (10%)	0.64 (8.9%)	5.17 (72%)	0.02 (0.3%)	0.016 (0.2%)	0.67 (9.3%)	7.23
8M	8	1.55 (16%)	0.63 (6.6%)	5.14 (54%)	0.04 (0.4%)	0.033 (0.3%)	2.11 (22%)	9.50
16M	16	3.57 (24%)	0.63 (4.2%)	5.53 (37%)	0.11 (0.7%)	0.030 (0.2%)	5.15 (34%)	15.05
Line singularity distribution								
1M	1	—	0.64 (11%)	5.14 (89%)	—	—	—	5.78
2M	2	0.19 (3.1%)	0.63 (10%)	4.88 (79%)	0.005 (0.08%)	0.004 (0.07%)	0.44 (7.2%)	6.15
4M	4	0.44 (5.4%)	0.65 (8.0%)	5.3 (65%)	0.019 (0.2%)	0.004 (0.05%)	1.75 (21%)	8.15
8M	8	1.38 (15%)	0.73 (7.7%)	5.26 (56%)	0.017 (0.2%)	0.005 (0.05%)	2.07 (22%)	9.46
16M	16	3.00 (21%)	0.65 (4.6%)	5.24 (37%)	0.05 (0.4%)	0.006 (0.04%)	5.11 (36%)	14.06

The linear time of the affected region demonstrates the algorithm's $O(n)$ nature. The partition, merge, 'other' timings rose as the problem size increased and more levels of recursive steps are involved. More levels of recursion increase the overhead, reducing the parallel efficiency.

5.2. Memory requirements

The main memory overhead on each processor derives from replicating the affected zone on processors. The memory requirement for recording deleted triangles is insignificant with respect to that of the affected region. The data of the affected zone include points, triangulation and the point index on the original processor. Since two zones are merged on the parent processor, the parent processor needs data for both affected zones. Processor 0 conducts the final stage merge, and therefore has the largest affected zone. The memory overhead on points and triangles is frequently below 30% of the memory needed for block points and Delaunay triangulations.

6. CONCLUSION

This study has described a parallel D&C Delaunay triangulation scheme, which utilizes the affected zone to reduce the amount of data transmitted when combining triangulations on different processors. The resulting program can be implemented in FORTRAN 77, C and MPI, and is easily ported to various platforms. Performance results show that speedups are similar on various distributions.

REFERENCES

1. Su P, Drysdale RLS. A comparison of sequential Delaunay triangulation algorithms. *Computational Geometry: Theory and Applications* 1997; **7**(5/6):361–385.
2. Said R, Weatherill NP, Morgan K, Verhoeven NA. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering* 1999; **177**:109–125.
3. Cignoni P, Montani C, Scopigno R. DeWall: A fast divide and conquer Delaunay triangulation algorithm in E^d. *Computer-Aided Design* 1998; **30**:333–341.
4. Blelloch GE, Hardwick JC, Miller GL, Talmor D. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica* 1999; **24**:243–269.
5. Lee DT, Schachter BJ. Two algorithms for constructing the Delaunay triangulation. *International Journal of Computer and Information Sciences* 1980; **9**:219–242.
6. Guibas L, Stolfi J. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics* 1985; **4**:74–123.
7. Dwyer RA. A faster divide and conquer algorithm for constructing Delaunay triangulation. *Algorithmica* 1987; **2**:137–151.
8. Preparata FP, Shamos MI. *Computational Geometry: An Introduction*. Springer: Berlin, 1985.
9. Shewchuk JR. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. *Applied Computational Geometry: Towards Geometric Engineering* 1996; **1148**:203–222.
10. Chen MB, Chuang TR, Wu JJ. Parallel 2D Delaunay triangulations in HPF and MPI. *Proceedings of the International Parallel and Distributed Processing Symposium*, San Francisco, CA, 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
11. Hoare CAR. Algorithm 63 (partition) and algorithm 65 (find). *Communications of the ACM* 1961; **4**:321–322.

