A PARALLELED DELAUNAY TRIANGULATION ALGORITHM FOR PROCESSING LARGE LIDAR POINTS

Yuanlong Song [1], Ming Li [2,3] *, Xiaojia Liu [2]

[1] Aerial Photogrammetry and Remote Sensing Group Co., LTD, Xi'an, China- SYL_ARSC@outlook.com
[2] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China -lxj@bit.edu.cn
[3]National Geomatics Center of China, Beijing 100830, China- liming@ngcc.cn

Commission III, ICWG III/IVa

KEY WORDS: Paralleled Delaunay Triangular Network; LiDAR; Point Cloud; MapReduce; Multi-core Computer.

ABSTRACT:

LiDAR is an important data source for disaster prevention and mitigation, its advantages include speediness, penetration, initiative, high-density and high-precision, high efficiency, information-richness. In this paper, we address the design and implementation of a practical parallel algorithm for Delaunay triangulation that works on massive point cloud data acquired by airborne LiDAR. The algorithm is based on the divide and conquer algorithm, Bowyer-Watson algorithm and triangulation-growth and can be easily extended to multi-core or cluster environment. It first divides the point cloud data into blocks at the MapReduce's Map phase; then the triangulation network of each block is simultaneously constructed using Bowyer-Watson algorithm on different CPU cores; at the reduction phase, the triangular network of each block is merged and optimized using triangulation-growth and Local Optimization Procedure (LOP). Experimental results show that the speed of the paralleled triangulation algorithm can be improved significantly compared to the sequential algorithm on multi-core desktop computer. The speedup depends on partition of the dataset and the number of CPU cores used, and is usually 2-3 times that of sequential algorithms.

1. INTRODUCTION

With the emerging of a variety of new data acquisition and data processing methods such as airborne/terrestrial laser scanning system, image matching from stereo image pairs, it is now possible to create very accurate and detailed 3D surface model of ground objects using point cloud data. Point cloud data has been widely used in terrain mapping, disaster monitoring and emergency rescue work. To process these highly dense discrete 3D point cloud data, the Delaunay triangulation is widely used, *e.g.*, in automatic filtering, airborne strip adjustment, line feature and breakline extraction, Digital Elevation Model (DEM) generation and terrain visualization, *etc*. However, with the rapidly increased point density and data volume, traditional sequential Delaunay triangulation method could no longer meet the performance requirement nowadays, especially involving current multi-core or clustered computational power. In this situation, a real-time paralleled Delaunay triangulation method has the potential to efficiently use the multi-core and clustered computers to fast construct a large triangulation network.

Traditional Delaunay triangulation algorithm can be divided into three classes according to the construction method. The divide and conquer algorithm proposed by Shamos and Hoey (1975), the triangulation growth algorithm proposed by Green and Sibson (1978), and incremental insertion algorithm proposed by Lawson (1977). A triangulation algorithm based on edge-pointer search and region-division is proposed by Zhang (2021), designed to reflect the positional relationship between triangles. A novel hybrid scheme of large scale reconstruction was proposed by Xue (2020), fully ensures the integrity of the reconstructed scene and is able to reconstruct large scale scene in single computer. Liu (2018) research proves that Delaunay triangulation maximizes minimal angle due to good geometric properties.

The divide and conquer algorithm recursively draws a line to split the vertices into two sets until each set includes less than three points. The Delaunay triangulation is then applied to each set, followed by merging the triangle of each sets along the splitting line. Using some tricky technique, the merge operation can be done within the time complexity of $O(N)$, resulting in a total time complexity of $O(NlogN)$. However, the space complexity of the algorithm is relatively high and takes more memory than that of other algorithm, especially when the data size is large.

The triangulation growth algorithm starts from arbitrary point in the points set as an initial point, and searches through all the points to find the nearest. An initial edge is built by linking these two points. The third point is searched according to the in-circle and max-min angle properties of Delaunay triangulation. After that, the two edges of the newly created triangle are used as the initial edges to build subsequent Delaunay triangles. These procedures are recursively repeated until all points are inserted into the triangular network.

The incremental insertion algorithm repeatedly adds one vertex at a time and retriangulates the affected parts of the graph. When a vertex v is added, we split the triangle that contains v into three, and then applies the flip algorithm. A typical incremental algorithm is Bowyer–Watson algorithm (Bowyer, 1981; Watson, 1981). It adds points, one at a time, to a valid Delaunay triangulation of a subset of the desired points. After every insertion, any triangles whose circumcircles contain the

* Corresponding author

new point are deleted, leaving a star-shaped polygonal hole which is then re-triangulated using the new point.

Currently a few paralleled Delaunay triangulation algorithms have been proposed. Most of them are based on the incremental algorithm or divide and conquer algorithm (Chow, 1981; Aggarwal *et al*., 1988; Cole *et al*., 1990; Teng *et al*., 1993; Chew *et al*., 1997). Blelloch *et al*. (1999) proposed a parallel algorithm for constructing Delaunay triangular network by developing a variant of the Edelsbrunner and Shi 3D convex hull algorithm, Christos *et al*. (2005) proposed multi-granularity parallel algorithm, Chernikov and Chrisochoides (2004, 2006) proposed a parallel two-dimensional constraint Delaunay network building method based on Bowyer-Watson algorithm. Isenburg *et al*. (2006) proposed a streaming computation of Delaunay triangulation for massive points. These algorithms provide feasible approach for paralleled Delaunay Triangulation construction from general point sets. However, few paralleled Delaunay Triangulation method optimized by considering the massive near uniformly distributed LiDAR point cloud data has yet been addressed and could be easily extend to cluster or cloud computing environment.

In this paper, a paralleled 2D Delaunay Triangulation method is proposed for processing massive point cloud data using a MapReduce paralleled computing model. The model is firstly introduced by Google as its enterprise cloud computing infrastructure and search engine (Dean and Ghemawat, 2004). Nowadays, it is becoming more and more popular to use this technique to process high throughout and massive data tasks on the internet.

The rest of the paper is organized as follows. In Section 2, we briefly describe the general idea of the proposed algorithms; we then introduce the detailed steps to implement the method, which is followed with the pseudo code of the algorithm. In Section 3, we present the experimental result by evaluating our algorithm and comparing with traditional sequential algorithm on a multi-core desktop computer. Section 4 concludes the paper.

METHOD

1.1 General Idea

The paralleled Delaunay triangulation method presented in this paper is based on the divide and conquer algorithm in combination with Bowyer-Watson algorithm and triangulation-growth. The basic idea of the method is to firstly divide the point cloud data into blocks at the MapReduce's Map phase; then the triangulation network of each block is simultaneously constructed via using Bowyer-Watson algorithm by different CPU cores or computer clusters; at the Reduce phase, the triangular network of each block is merged by using Local Optimization Procedure proposed by Lawson (1977). In this way, the paralleled computing of massive point cloud data is distributed to many processors, or computer clusters. Since the computing, memory, I/O, and other resources are in distributed environment, the process of massive data could be effective and highly scalable.

1.2 Procedure

The general procedure of the algorithm is shown in Figure 1, which is described in detail as following.

1.2.1 The point cloud partitioning

A paralleled algorithm should firstly consider how to divide a task into a set of concurrently executable tasks. In our algorithm, a task partitioning procedure is firstly applied at the Map phase to separate the point cloud data set into different blocks with roughly equivalent data volume according to its geographic coverage.

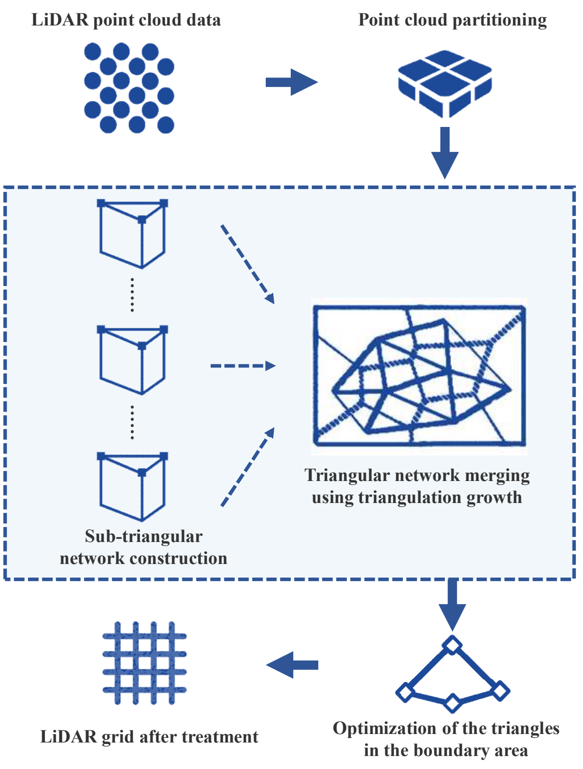
LiDAR point cloud data
Point cloud partitioning
Triangular network merging using triangulation growth
Sub-triangular network construction
LiDAR grid after treatment
Optimization of the triangles in the boundary area

Figure 1. The overall procedure of the paralleled Delaunay triangulation algorithm

Currently there are three different point cloud data partition methods. Partition into strips, partition into blocks with equal area, and partition based on quad-tree, as shown in Figure 2. If the grid partitioning or quad-tree partitioning method is applied, at least two border lines will be created for each generated blocks, as shown in Figure 2b, and Figure 2c. This will cause additional overhead in the procedure of triangular network merging and thus reduce the performance. Considering that most point cloud data collected by airborne LiDAR or mobile laser scanning system are stored sequentially as scan line by scan line for a flight or moving path, the point cloud block data created by grid partitioning or quad-tree partitioning method will also result in a lower sequential Delaunay triangulation performance since it will need much more time to find the neighbourhood vertexes. In this paper, we prefer to partition the point cloud data into equal space vertical or horizontal strips, most probably along with the flight path direction. In this way, at most two adjacent strips need to be considered when merging triangular networks, which simplifies the merging process and improves performance.

Another importance factor should be considered is the data volume of each block. Since the overall time complexity of the sequential Delaunay triangulation algorithm used is O(*Nlog(N)*), the performance of the triangulation on a small number of points will be higher than that on a large number of points. On the contrary, small point set size for each block will create more blocks thus taking more time to merge the triangle network created by the blocks. Therefore, a trade-off of the point sets size should be considered. In our experiment, a size of 1,000,000 points for each block seems to be an optimal choice.

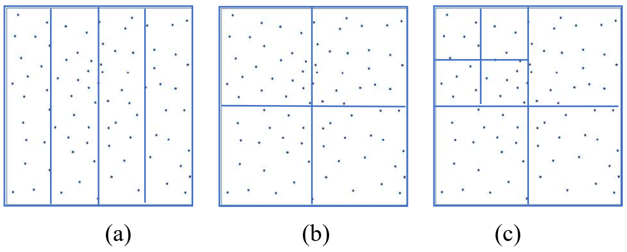

Figure 2. Possible point cloud partition strategies:
(a) Partition into equal space vertical or horizontal strips;
(b) Partition into blocks with equal area;
(c) Partition based on quad-tree.

1.2.2 Sub-triangular network construction with Bowyer-Watson Delaunay triangulation algorithm

After the point cloud has been partitioned into a variety of blocks with approximately equal number of points, the triangulation network of each block is simultaneously constructed using Bowyer-Watson Delaunay triangulation algorithm in each task at the Map phase. After the Delaunay triangular network of all blocks have been constructed, the results are returned to the Reduce function for merging.

1.2.3 Merge the triangular network of each block using triangulation growth algorithm

At the Reduce phase, the triangular network of each block is merged using the triangulation-growth algorithm after every task of Step 2 has finished. This step can be described as following.

(1) Extract boundary of two adjacent blocks
In order to merge the two Delaunay triangular networks of the adjacent blocks, we firstly need to extract the boundaries of these two triangular networks. Since the edge on the boundary has only one adjacent triangle, we can find out all edges which have only one adjacent triangle by traversing the triangular networks, and then save each edge as directional edge counter clock-wisely. In this way, the endpoints of these edges are extracted as the boundary points of the underlying triangular network (see Figure 3a and 3b).

(2) Generate the new triangles on the boundary of two adjacent blocks
To merging two adjacent triangular networks, the triangulation-growth algorithm is employed to generate the new triangles on the boundary with the boundary point set obtained in step (1). It starts from choosing an arbitrary boundary edge as an extensible edge and finding out the optimal point in its adjacent boundary point sets to form a Delaunay triangle. The newly created directional edges of the Delaunay triangle are added into the extensible edges for further processing. This procedure can be illustrated as Figure 3c, 3d, and 3e.

To choose the optimal point, four rules are applied: i) The point is on the right side of an extensible edge; ii) The new triangle created by this point and the two endpoints of an extensible edge fulfills the in-circle property; According to the law of cosines (see Eq.1), the point which has minimum Cos(C) is the point that meets Delaunay rule; iii) The newly built extensible edge is not oriented in the opposite direction; iv) The newly built extensible edge is not intersected with any boundary edge (except tangency).

$$Cos(C) = \frac{(A^2 + B^2 + C^2)}{2 \times A \times B} \tag{1}$$

According to the above-mentioned rule, we can find out the optimal points from the boundary point set until all the extensible edges complete finding their optimal points, as shown in Figure 3.

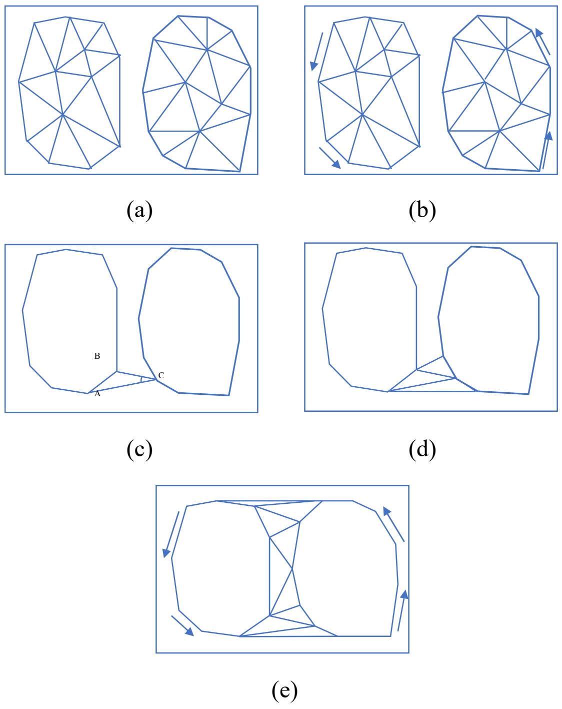

Figure 3. The general procedure of creating the new triangles on the boundary of two adjacent blocks:
(a) Original Delaunay triangular network of two adjacent blocks;
(b) Boundary extraction;
(c) Choose an arbitrary boundary edge as an extensible edge and search the optimal point in its adjacent boundary point sets to form a Delaunay triangle;
(d) Add the newly created directional edges of the Delaunay triangle into the extensible edges list;
(e) The newly created boundary triangles of two adjacent blocks.

1.2.4 Optimizing triangles at the boundaries

In step 3, a merged triangular network from two adjacent blocks is established by creating the boundary triangles between two adjacent boundaries. Since only the boundary points of the two triangle networks are used during iteration, the newly created boundary triangles are sometimes long and narrow triangles, which do not satisfy the Delaunay triangulation rule considering the adjacent interior points within two adjacent blocks. Therefore, the LOP algorithm is applied to optimize the newly built triangles and the interior triangles inside the boundaries according to the in-circle property of Delaunay triangulation (Figure 4). The newly created triangles and its adjacent triangles are recursively optimized, and eventually the merged Delaunay triangular network of the two adjacent blocks is created.

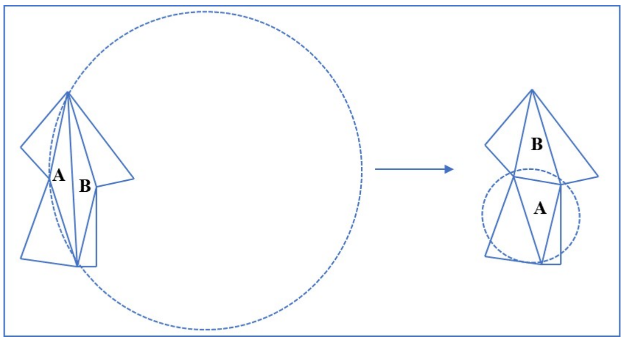

Figure 4. Illustration of the optimization procedure of the boundary triangles using LOP algorithm.

1.2.5 Construction of the global Delaunay triangular network

Repeat step 3 and 4 until the triangular networks of all blocks are merged so that a whole seamless Delaunay triangular network is built.

1.3 Implementation

The main procedure of the algorithm can be expressed as the pseudo code as follows.

```
BEGIN
Step1: Divide the point cloud data into multiple blocks and add to the data list (better if it is a multiple of the CPU cores)
Step2: WHILE (the data list is not empty) DO
    Distribute a block of the data to a free computing unit (thread) to complete sub-triangular network construction. This is called the Map phase.
    ENDWHILE
    Get ordered set of sub-triangular networks. This is called the Reduce phase.
Step3: Put the sub-triangular networks into stack, and merge the sub-triangular networks
  WHILE (stack is not empty) DO
      Counter++
      IF counter equals 2
      THEN merge sub-triangular networks (as shown below)
      Add the merged sub-triangular networks into stack
      Reset counter
  ENDWHILE
END
```

The sub-triangular networks merging algorithm can be expressed as following.

```
BEGIN
Step1: Get boundary edge and point set from the two sub-triangular networks' boundary points and edges
Step2: WHILE (edge set is not empty) DO
      IF one of the optimal point in the point set meets the condition
      THEN save the triangle, add the two newly built edges into the edge set
      ELSE return
    ENDWHILE
Step3: Optimize the newly built triangle by LOP algorithm
END
```

EXPERIMENTAL RESULT

In order to verify the correctness and validity of the algorithm, the algorithm is implemented using C++ programming language on Windows Vista operating system. A desktop computer with 8 CPU cores and 4 gigabyte memory is used for testing. Each core speed is 2.1GHz. The experimental data is located in Huanghua county, HeBei province of China, collected by Chinese Academy of Surveying and Mapping in 2011, using ALS60 airborne laser scanner from Leica Geosystems. The relative flight height is about 900m, the average point density is about 0.67m.

Figure 5 shows the Delaunay triangulation construction of a small size of point cloud in the experimental area. The point cloud contains 2,135,953 points. It is firstly partitioned into two vertical strips. Figure 5a shows the triangular network after an initial merging of the two strips. Figure 4b is a zoomed-in of the area *A* in Figure 5a. From figure 5b we can clearly see that the initial merge of the triangular network is not a truly Delaunay triangular network since it does not meet the in-circle and max-min angle properties. Figure 5c shows the optimized triangular network using LOP algorithm. Figure 5d is a zoomed-in of the area *A* in Figure 5c. From figure 5d we can see that the optimized triangular network is now a true Delaunay triangular network and meet the in-circle and max-min angle properties.

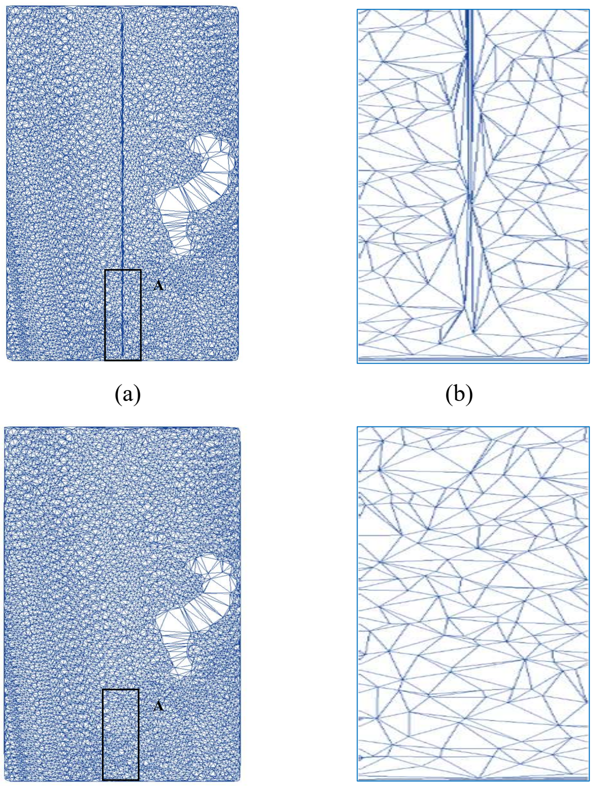

Figure 5. Delaunay triangulation construction of a small size of point cloud in the experimental area:
(a) The two initially merged triangular networks;
(b) Zoomed-in of the area *A* in (a);
(c) The final optimized Delaunay triangular network from (a);
(d) Zoomed-in of the area *A* in (c).

Figure 6 shows the Delaunay triangulation construction of a relatively large size of point cloud in the experimental area. The point cloud contains 9,986,520 points. It is firstly partitioned into four vertical strips. Figure 6a shows the triangular network after an initial merging of the four strips. Figure 6b shows the optimized triangular network using LOP algorithm.

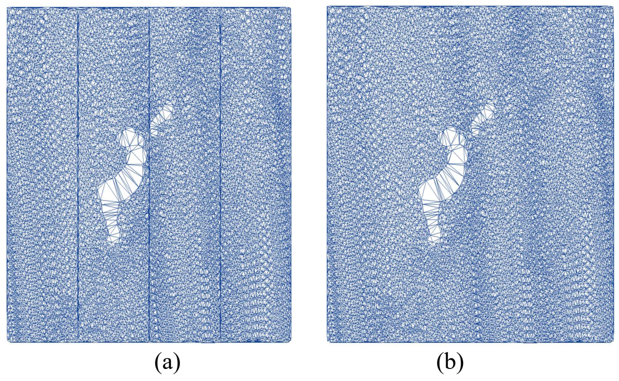

Figure 6. Delaunay triangulation construction of a relatively large size of point cloud in the experimental area:
(a) The four initially merged triangular networks;
(b) The final optimized Delaunay triangular network from (a).

In both cases, the optimized Delaunay triangular networks are successfully constructed finally using the paralleled algorithm. The performance of the algorithm is compared with the sequential Delaunay triangulation algorithm developed by Isenburg (2011). It has been considered as one of the fastest Delaunay triangulation algorithm of the world. 5 point cloud data sets with different level of point numbers are used to evaluate the performance. 2-8 threads are used to the test the efficiency.

The performance comparison result between the paralleled Delaunay algorithm (PA) and the sequential algorithm (SA) is shown in Table 1.

Number of Points	Speedup (seconds)			
	SA*	PA (2 threads)	PA (4 threads)	PA (8 threads)
2,135,953	5.335	3.369	2.534	2.028
4,360,160	13.977	8.112	5.272	4.602
6,010,527	17.005	11.281	7.0849	5.676
9,986,520	40.123	35.256	19.516	23.79
21,214,412	failed	286.321	263.131	241.301

Table 1. Performance comparison between the Paralleled Delaunay Algorithm (PA) and the Sequential Algorithm (SA)

From Table 1, we can see that the paralleled algorithm can improve the speed of Delaunay triangulation of LiDAR point cloud significantly. For example, for a data set with 2,135,953 points, the sequential algorithm takes 5.335 seconds while the paralleled algorithm with 4 threads takes only 2.534 seconds. For a data set with 9,986,520 points, the sequential algorithm takes 40.123 seconds while the paralleled algorithm with 4 threads takesonly 19.516 seconds.

The memory mapping technique is used to process massive point cloud data when the data volume exceeds the computer memory. In this case, the traditional sequential algorithm cannot complete network building due to the failure of memory allocation when building and merging the triangular networks. With memory mapping, the built sub-triangular networks are stored as disk buffers, and the merging of triangular network is directly operated on disk buffers. This is very fast for medium size of point sets compared to hard disk file I/O. However, with the increased size of point sets, the segment error of the operating system becomes more and more frequent when doing the triangulation network merging and optimization. This significantly reduces the triangulation performance, which can be observed in a data set with 21,214,412.

Further analysis of the experimental results shows that the point number of each partitioned blocks and the merging method of sub-triangular network are two critical factors that affect the algorithm efficiency. The optimal speed comes out when about 1,000,000 points are contained in each block. At present, the speed in sub-triangular networks construction phase is almost linear. However, since we use a sequential method in sub-triangular networks merger phase, resulting in a non-linear decrease of the overall speed of the algorithm. The merging time will increase if the number of sub-triangular networks is too large for a data set, resulting in decreased overall performance of the algorithm. This situation can be observed for a data set with 9,986,520, where the paralleled algorithm with 4 threads cost 19.516 seconds while the paralleled algorithm with 8 threads cost 23.79 seconds.

CONCLUSIONS

In this paper, a simple and scalable paralleled Delaunay Triangulation method is proposed for processing massive point cloud data using a MapReduce paralleled computing model. The paralleled Delaunay triangulation algorithm is based on the divide and conquer algorithm, in combination with Bowyer-Watson algorithm and triangulation-growth algorithm. It first divides the point cloud data into blocks at the MapReduce's Map phase; then the triangulation network of each block is simultaneously constructed via using Bowyer-Watson algorithm by different CPU core; At the Reduce phase, the triangular network of each block is merged and optimized using triangulation-growth and LOP algorithm. The application of this algorithm can effectively improve the speed of terrain construction and provide more efficient data services for emergency relief.

Experimental results show that the speed of the paralleled triangulation algorithm can be improved significantly compared to the sequential algorithm on multi-core desktop computer. The speedup depends on partition of the data set and the CPU cores used, usually from 2-3 times compare to the sequential algorithm.

Further study will focus on paralleling the merging and optimization procedure and migrating the algorithm to computer clusters environment.

ACKNOWLEDGEMENTS

The work is funded by National Natural Science Fund of China (62076027).

REFERENCES

Aggarwal, A., Chazelle, B., Guibas, L.,ÓDúnlaing C. C, and Yap C., 1988. Parallel computational geometry. *Algorithmica*, 3(3), pp. 293–327.

Blelloch, G., Hardwick, J., Miller, G., Talmor, D., 1999. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica*, 24, pp. 243–269.

Bowyer, A., 1981. Computing Dirichlet tessellations. *The Computer Journal*, 24(2), pp. 162-166.

Chernikov, A. N. and Chrisochoides, N. P., 2004. Parallel guaranteed quality planar Delaunay mesh generation by concurrent point insertion. In *14th Annual Fall Workshop on Computa-tional Geometry*. MIT, pp. 55–56.

Zhang, J. and Tian, H. M., 2021.A triangulation algorithm based on edge-pointer search and region-division. *Acta Automatica Sinica*, 47(1),pp.100-107

Xue, J. S. and Yi, H., 2020.A hybrid multi-View 3D reconstruction method based on scene graph partition. *Acta Automatica Sinica*, 46(4),pp.782-795.

Liu, H. Q. and Yu, J. B., 2018. A bidimensional local mean de composition algorithm. *Journal of Computer-Aided Design & Computer Graphics*,30(10),pp.1859−1869.

Chernikov, A. N., and Chrisochoides N. P., 2006. Generalized Delaunay Mesh Refinement: From Scalar to Parallel. In: Proceedings of the 15th International Meshing Roundtable, September 2006, Springer-Verlag, pp.563-579.

Chew, L. P., Chrisochoides, N., Sukup, F., 1997. Parallel Constrained Delaunay Meshing. In: *Proceedings of 1997 Joint ASME/ASCE/SES Summer Meeting, Special Symposium on Trends in Unstructured Mesh Generation*, Northwestern University, Evanston, IL, 29 June–2 July 1997, pp. 89-96.

Chow A., 1981. Parallel Algorithms for Geometric Problems. Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, IL.

Christos, D. A., Ding, X. N., Andrey, N. C., 2005. Multigrain parallel Delaunay mesh generation: challenges and opportunities for multithreaded architectures. In: *Proceedings of 19th ACM International Conference on Supercomputing*. ACM , New York, pp. 367-376.

Cole, R., Goodrich, M. T., andÓDúnlaing C.C., 1990. Merging free trees in parallel for efficient Voronoi diagram construction. In: *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, July 1990, pp. 32–45.

Dean, J. and Ghemawat, S., 2004. MapReduce: Simplified Data Processing on Large Clusters. In: *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation* (*OSDI'04*). USENIX Association, Berkeley, CA, pp. 137–150.

Green, P. J., Sibson, R., 1978. Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21(2), pp. 168-173.

Isenburg, M., 2011. LAStools: converting, filtering, viewing, gridding, and compressing LIDAR data. http://www.cs.unc.edu/~isenburg/lastools/
Isenburg, M., Liu, Y., Shewchuk, J., Snoeyink, J., 2006. Streaming Computation of Delaunay Triangulations, *Proceedings of SIGGRAPH'06*, pp. 1049-1056.

Lawson, C. L., 1977. Software for C1surface interpolation. Rice J., 1977. *Mathematical Software III*. Pasadena. California Institute of Technology, California, pp. 161-194.

Shamos, M. I., Hoey, D., 1975. Closest-point problems. In: *Proceeding of the 16th Annual IEEE Symposium on Foundation of Computer Science*. Los Angeles. IEEE, California, pp. 151-162.

Teng, Y. A., Sullivan, F., Beichl, I., and Puppo, E., 1993. A data-parallel algorithm for three-dimensional Delaunay triangulation and its implementation. In: *Supercomputing 1993*. ACM: Providence, RI, pp. 112–121.

Watson, D. F., 1981. Computing the n-dimension Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2), pp. 167-172.