Parallel Algorithm for Building Extraction from LiDAR Data

Hyo Jong Lee
Div. of Computer Science and Engineering
Center for Advanced Image and Information Technology
Chonbuk National University, Jeonju, Korea

Abstract - The data presented by Light Detection and Ranging (LiDAR) systems are in a dense and accurate three dimensional pattern without point classification, such as trees, roads, and buildings. Extraction of boundary points is essential for recognizing buildings. However, it is complicated to process the LiDAR data due to its irregularity and a large number of collected data points. In order to find boundary points in a quick and accurate way from a huge number of data points, a parallel algorithm for building extraction is proposed. In this paper, every processor reads the LiDAR data points and builds a quadtree respectively. Thus, a quadtree is built and shared through a network file system. Later, a breadth first search (BFS) is applied on every processor. The position of a node in the quadtree, in which has the number of children smaller than a predefined grain size, is stored in an array called as BFSArray. Process farming paradigm has been applied to process each nodes using MPI. In our primary experiment, results show significant speedup for multiple processors compared to a sequential algorithm.

Keywords: Boundary point, process farming, MPI, LiDAR

1 Introduction

Recently Light Detection And Ranging (LiDAR) has become a reliable technique for 3D data collection [1]. The LiDAR system consists of three data collection tools: a laser scanner, Global Position System (GPS) for sensor position information, and an Inertial Measuring Unit (IMU) for orientation information. The laser scanner transmits and detects infrared signals to measure ranges.

LiDAR offers many advantages over traditional photogrammetric methods for collecting elevation data. These include high vertical accuracy, fast data collection and processing, robust data sets with many possible products, and the ability to collect data in a wide range of conditions [2]. Therefore, LiDAR system has been introduced as a new tool that efficiently provides accurate data collection from extensive areas.

In LiDAR processing there are two completely different possible approaches for processing images. One is using 2-D images. The advantages of 2-D images are less time required for processing and less cost. However, images in 2-D format are inadequate for accurate modeling and for defining road boundaries and other building boundaries due to their lack of high resolutions [3]. Accordingly, recent demand and utilization of 3-D terrain information has been increased in various field and researchers have already investigated 3-D extraction methods of geographical features, buildings, and road networks [4].

Obviously more time is required for processing 3-D images. In order to find boundary point in a 3-D images format in a quick and accurate pattern, parallel computing algorithm is one option. In our paper, parallel computing algorithm is applied. Firstly, all LiDAR points are built in a quadtree structure in every processor. Hence, a shared quadtree is accomplished. Secondly, BFS search is used on every processor for finding all nodes, in which the number of children is smaller than a predefined grain size. Experiment results show that building a quadtree and BFS searching take only a small amount of time compared to finding boundary point. At this point, all preparation work for our parallel computing algorithm is finished. Afterwards processor farming is performed for distributing tasks to processors. Messages Passing Interface (MPI) is employed as a communication tool during the whole process. Details description will be presented later.

The rest of the paper is organized as follows. Section II introduces quadtree representation of LiDAR data and the concepts of processor farming model. In Section III we discuss the detail operation of our algorithm. Experiment results and discussion are described in Section IV. Conclusion is drawn in the last section.

2 Quadtrees and processor farming model

2.1 Quadtree and its properties

In the field of image processing, computer graphics, and remote sensing two dimensional point and ranging data are often indexed using quadtrees[5]. A quadtree is a tree data structure in which each internal node has up to four children. All data information are stored only in the leaf nodes. Data in a quadtree is often collected in point format, for example, a height measurement is made at that location, and represented by an \((x,y,z)\) triplet with \(x\) and \(y\) representing coordinates and \(z\) representing a measurement at that location. Nevertheless, a parent node only has the information of the number of children and the area range of its all leaf nodes locate in.
Quadtrees are most often used as a representation of a regular partitioning of space where regions are split recursively into quadrants until there is only one point in each quadrant. In other words, each quadtree block (also referred to as a cell, or node) only contains one particle. In Fig. 1, the region is recursively divided into four parts if more than one particle exists inside a single block.

Figure 1. Quadtree structure

Quadtree-based spatial domain decomposition algorithm is designed for general use, and it can produce scalable geographical workload [6]. As we mentioned previously, LiDAR points are in a dense pattern, it is necessary to reduce the size of quadtree first before extracting boundary point. A quadtree with either a threshold or a maximum-depth limit allows us to reduce the size of tree at the cost of prediction accuracy, as it is no longer an exact copy of the original data. Setting the accurate threshold helps smooth the experimental data [7]. In this paper, a threshold value of height was set to compress a quadtree. A threshold means a criterion to merge any adjacent blocks as a single block if the height difference for the adjacent blocks is smaller than the predefined threshold.

Figure 2. Reduced quadtree structure

Assuming the different height values in block 120, 121, 122, 123 in the Fig. 1 fall into the threshold, we can consider them as a single block and merge them. Equally, we assume the difference of height in block 10 and 11 is smaller than the threshold and the different height values in remaining blocks are bigger than the threshold. Therefore, block 10 and block 11 can be taken as a single block. After two blocks are merged, we use the average height as a new value to represent the height information in such block. Fig. 2 illustrates the reduced quadtree structure from Fig. 1 based on our assumption.

In this paper, the parallel computing algorithm is mainly focused. Therefore, finding the optimized threshold value for merging is out of scope of our problem. Adjusting the different threshold values will produce a more precise image.

2.2 Processor farming

The processor farming model works well even in a condition where pipeline structure cannot balance tasks or there are not enough buffers. Cok [8] also stated that efficient parallel algorithms could be implemented with processor farming.

Processor farming is defined as a group of independent tasks. The processors consist of a master processor and a number of worker processors [9]. A master processor controls the whole parallel processing. Our quadtree is not well balanced, that is, some nodes contain a little number of children, while other nodes contain a large number of children in BFSArray. It degrades the efficiency of algorithm if a master only sends one index of BFSArray to a worker processor at each time.

To solve the problem, we define a lower bound as grain size - $\alpha$, and a upper bound as grain size + $\alpha$ at first. The $\alpha$ has been chosen to be equal to approximately 10% of the predefined grain size. When parallel computing algorithm begins, a master processor starts from the first index of the array. We call it a start index. If the number of children of a target node is smaller than a lower bound, then we move the pointer to the next index of array, until the total number of children of all nodes reaches lower bound or exceeds upper bound. Now, we mark the position of current pointer as an end index. If the second situation happens, we prefer to move the pointer back by one index. Surely, if the total number of children for all rest of nodes in BFSArray is smaller than lower bound, then master sends the start index and the last index of BFSArray to a worker processor and broadcasts a “finish” signal to all worker processors. Worker processors stop receiving tasks as long as a “finish” signal received from master processor.

Likewise, a “finish” signal is sent back by a worker processor to the master processor as soon as it finished the assigned task, then the master processor repeats the previous procedure and sends start and end index to that worker processor until no more tasks are available in BFSArray.

Fig. 3 illustrates the first 13 elements of a hypothetical BFSArray when grain size is defined as 40,000. The smaller number denotes the index of array. The number in the array box presents the number of children the node has. Actually, BFSArray is a pointer array and stores the address of those nodes and each node contains the information of number of children it has.
When a grain size is 40,000, the lower bound and upper bound of grain size is 36,000 and 44,000, respectively. Consequently, a master processor stops moving a pointer to next index value if total number of children either falls between a lower bound and a upper bound or is beyond the upper bound scope.

The first element of BFSArray only has 42 children, and then master processor moves the index pointer to next one and calculates the summation of number of children. The index pointer stops at the index 10. Because the accumulated of children of the first eleven elements is 36916 ∈ \([\text{lowerbound}; \text{upperbound}]\). After that, a master processor sends \text{start index} 0 and \text{end index} 10 to a worker processor. A master processor repeats this procedure until all tasks are assigned to each worker.

In next section, a detail operation for how the boundary points is generated by a proposal parallel computing algorithm will be discussed and our parallel computing algorithm will be presented.

3 Parallel Monotone Chain algorithm

Andrew’s Monotone Chain algorithm [10] is implemented for extracting boundary points in this paper. The “Monotone Chain” algorithm computes upper and lower hulls of a monotone chain of points. It runs in O(\(n \log n\)) time due to the sort time. After that, it only takes O(\(n\)) time to compute the hull, a hull has the same meaning of boundary point in our paper.

As we have introduced the LiDAR data previously, an object, such as a roof or a road, is represented by a set of dense data points. In order to extract the boundary points for an building, a reduction on quadtree size is preferred at first. Finding a group of neighbor points is a priori for extracting boundary points of an building. Without reducing the size of quadtree, it increases the computation time of finding groups of neighbor points. A group of neighbor points could be an object or multiple adjacent objects and the computation time for determining a group of neighbor points is O(\(n^2\)). Because every point has to look for all other points for one time and the final neighbor points are determined. In our experiment, we find out reducing a quadtree does not take too much time. In contrast, looking for a group of neighbor points at the same height consumes most of computation time. Thus, reducing quadtree size is essential for speeding up the whole process.

Each index in BFSArray points to a node, in which has less number of children than a grain size. When a worker processor receives a \text{start} and \text{end} index of BFSArray, reducing each sub-quadtree according received index information. It starts to reduce the sub-quadtree, which is pointed by \text{start index}, and finishes until the sub-quadtree that is pointed by \text{end index} is compressed. After that, processor collects all the remaining data points in each sub-quadtree and sorts them by their height values.

Andrew’s Monotone Chain is designed for 2D-image. In order to apply Andrew’s Monotone Chain in 3D-image, we have to extract 3D-image points at different height level. Therefore, a reasonable threshold of 0.1 meter is used. When height threshold is used, all points with the height difference is smaller than the height threshold are considered on the same plane. With all the data points that a worker processor collects from reduced sub-quadtrees, a worker processor classifies all these data points. It starts from the data with the lowest height value and finds all other data points which are on the same plane. Afterwards, processors look for group of neighbor points on this plane. Finally, boundary points of each group of neighbor points can be found at this height level. After that, worker processor moves to next height level and find boundary point until all data points from reduced sub-quadtrees are searched.

Suppose all the points in Fig. 4 are the remaining points after a processor performed a sub-quadtree reduction. Since all data points are sorted by height value, we know the data point whose height is the lowest. In Fig. 4, all height values have subtracted the lowest height value. Hence, the value of height starts from 0 in our example. Basically, we use the divide and conquer method to speed up the whole process. Instead of allowing one processor to find all groups of neighbor data points and then extract boundary point from each group of neighbor data points, we first store all LiDAR data points in a quadtree tree and master sends the index of BFSArray to processors, which can be considered as sending a small number of data in a small area each time to processors. With small amount of data points, worker processors finish the task much faster, since a huge amount of time on finding neighbor data points is saved. A master processor sends another task to a worker processor as soon as the worker processor finished previous assigned task.

Worker processors store the boundary point into a buffer and send the list of boundary points back to a master processor if buffer is full. We have not proved the optimal buffer size; however, the performance is acceptable in the
experiment when the buffer size is chosen as 4 or 5 times of grain size. When a master processor receives a boundary point list from a worker processor, then it resorts the all the boundary points and extracts a new list of boundary point as the same way as worker processor does.

Processor farming plays a key role in our parallel computing algorithm for extracting boundary point list. Fig. 5 presents the processor farming model of our parallel computing algorithm and the parallel computing algorithm for extracting boundary point is presented in Fig. 6.

**Algorithm 1: Parallel Extraction of Boundary Points**

**Master part:**
Build a quadtree and creates a BFSArray;
Distribute a new task as soon as a worker finished an assigned task;

if the buffer is full in a worker then
    Receives the list of boundary point in the buffer and resort all boundary points;
end

Collect all boundary point list from workers when no available task in BFSArray;
Produce the final boundary point list;

**Worker part:**
Build a quadtree and creates a BFSArray;
Receive a task and compresses sub-quadtrees;
Extract boundary point of buildings;

if buffer is full then
    Sends a list of boundary points back to master;
end
Send the list of boundary points when all assigned tasks were finished;

4 Experiment results

The implemented parallel computing algorithm for abstracting boundary point was experimented on a clustering system, which has a master processor and 17 worker nodes. the master processor is Intel Xeon 2.8GHZ and each worker processor is Pentium 4 630, 3GHZ. The parallel library is MPI 2.0 which was released by the MPI Forum [11]. The total number of data set is over four millions, which were collected by LiDAR from urban area.

Fig. 7 shows a sample LiDAR data and corresponding buildings extracted. Since LiDAR data are a set of coordinates with its height, the height values have been visualized as shown in Fig. 7 (a) using a rainbow bar. The extracted buildings were outlined connecting points from the Andrew’s Monotone Chain algorithm. We confirmed visually that every building has been extracted correctly with an aerial photograph.
takes more time for computation boundary point list from receiving worker processors and communication overheads were increased sharply.

From Fig. 8, it also indicates that scalability fails earlier when a very small grain size is applied. In our case, the scalability fails only after 3 processors involve, when grain size is chosen as 4,000, which is only 1/1000 of our test data. This issue should be taken care of in further study.

![Figure 8. Execution time for extracting boundary points](image)

## 5 Conclusions

Extraction of buildings from 3-D LiDAR data requires heavy computational processing. A parallel computing algorithm with processor farming model is proposed in this paper to reduce the processing time. Our algorithm is starting with building a quadtree on every processor and performs a BFS search. A BFSArray stores all the position of nodes, in which has less number of children than a predefined grain size. Processor farming distributes the indices of BFSArray to worker processors. The worker that completes a task first always asks for another task from the master. Boundary points only are sent among workers and master when the buffer is full in the workers or workers have finished all tasks in order to reduce the communication overheads. Grain size affects the performance of our parallel computing algorithm. With a small grain size, the scalability fails earlier and load balancing becomes an issue, since a master is busy at most time with extracting boundary points. The experiment results show our parallel computing algorithm improves the processing time six times faster than a sequential approach, which only one processor works.

In this paper, we do not focus on the accuracy of boundary points. However, with adjusting threshold or applying more sophisticated building extraction algorithms will generate more precise buildings. During the experiments, it proves that our parallel computing algorithm with a processor farming model works well for a large number of data set. Therefore, it is suitable for a vast range of parallel computing applications.

## 6 References


