Parallel Processing of Large LIDAR Datasets

Matthew McMahon[1], Michael Webb[2], Fredric Bernal[1], Scott Hamilton[3], Eric LaMar[1]

[1]*The Institute for Scientific Research, Inc., Fairmont, WV*
[2]*Lockheed Martin Technology Services, Fairmont, WV*
[3]*Galaxy Global Corp., Fairmont, WV*
Email address of corresponding author: mmcmahon@isr.us (Matthew McMahon)

Abstract

This paper describes our experience implementing parallel image processing algorithms for processing large, raw LIDAR (Light Detection and Ranging) point-cloud datasets on a Linux cluster. We describe the design of parallel algorithms for image rasterization and triangulation, two algorithms for manipulating LIDAR data. As with many fields of science, the size of image datasets in the Geographic Information Systems (GIS) realm has grown with the resolution and precision of the measuring equipment. The results of our work indicate that the power of a Linux computational cluster can facilitate the processing of this data, allowing the viewer to view and manipulate results in a timely manner. This work sets a foundation to improve on some of the basic assumptions made about the data we used, and also motivates the future parallelization of additional image processing algorithms for large LIDAR datasets.

1. Introduction

LIDAR (Light Detection and Ranging) technology is the use of laser light to gather position and absorption information about objects. As compared with radio waves in RADAR, the use of reflected laser light enables high resolution and accuracy in such varied applications as vehicle speed measurement, atmospheric research, and forestry. A LIDAR dataset is a unstructured point cloud, with elevation and signal intensity recorded for each point. Typical LIDAR datasets also include basic statistics about the point cloud (e.g,: spatial extent) Due to its unstructured nature, it is typically converted to other formats (e.g., pixel images, triangle meshes) for display and analysis.

The work described in this paper concentrates on LIDAR datasets obtained from airborne LIDAR systems used to gather elevation and intensity information from terrestrial objects. As the performance of LIDAR systems has become better in terms of sampling rate and resolution, LIDAR is increasingly being used for remote sensing applications. Higher performance yields larger image sizes, however, and there has been an associated increase in demand for full-resolution interactive processing and visualization of this data. The requirements of this processing create a computational bottleneck, often forcing Geographical Information Systems (GIS) professionals to work with reduced resolution or reduced spatial extents. This shortcoming suggests the application of parallel computing resources to improve performance.

The objective of the work described here is to design a system capitalizing on parallel computing resources to allow users to process and analyze LIDAR data at its fullest resolution. We first discuss two common display and analysis formats and the processing required to transform LIDAR data into these formats. We performed

extensive scalability tests (see Section 3) on a range of datasets and found that our parallel algorithms scale quite well with the number of processors used. We observe that while this field is still developing, our results are promising (see Section C); we also discuss a number of possible improvements and propose a few new research directions.

2. Background and Discussion of Application

2.1 Background

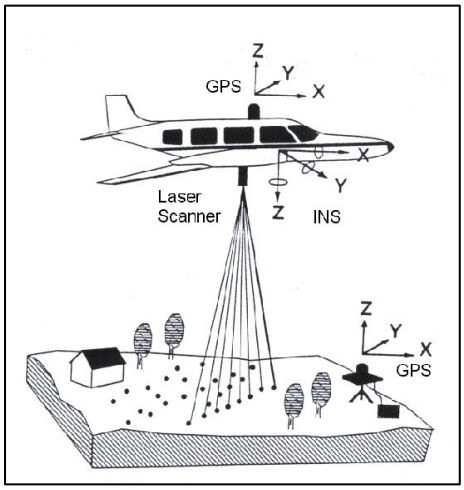

Figure 1. Airborne LIDAR system. Modified from illustration in [6]

LIDAR data gathered from airborne LIDAR systems is generated in the form of a point cloud containing information about surface elevation and signal intensity. The point cloud is in the form of scattered, unstructured data points. Understanding a large point cloud is difficult, so LIDAR data is typically converted from its raw, unstructured format into one of two structured formats: digital images and triangle meshes. Both of these format can be analyzed and understood by human users much more readily than raw LIDAR data.

The conversion of large, unstructured point clouds to more manageable formats is computationally expensive. For purposes of viewing and interacting with the data, it is useful to apply some conventional digital image processing techniques. Mapping the data to a structured grid makes image manipulation more tenable, and lays the groundwork for application of additional image processing algorithms, e.g., for feature extraction and classification. For this work, two algorithms were applied:

- image rasterization, which maps the spatially dispersed point cloud data onto a structured grid and locally averages the elevation information to achieve an interpolating effect to fill spaces, and
- triangulation, which creates a triangulated network using the point cloud points as vertices, yielding an mesh that can be viewed and manipulated in three dimensions.

In describing the implementation of these algorithms in parallel, the terminology described in [6] is used. Buyya's approach describes the parallel development process in terms of problem partitioning, inter-process communication, agglomeration of results, and mapping of computations.

2.2 Image Rasterization

The raster model takes as input unstructured LIDAR data and converts it into a high-resolution structured grid (i.e. a color image). There are a variety of approaches for accomplishing this [9]. For this application, the 2D Shepard algorithm [1,2] was parallelized. The parallel Shepard algorithm includes the following steps:

- Partitioning—in Shepard's algorithm, each point in the structured grid comprises a distance-weighted average of the 15 "nearest neighbors" to that point from the raw LIDAR data. The computation may be partitioned by computing each point in parallel. Partitioning is a simple division of the data into contiguous subregions, with some overlap between adjacent subregions to ensure the finding of nearest neighbors on edges between regions.
- Communication—due to the coarse granularity of this computation, interprocess communication and control are minimal. Initially each processor must read in the entire data set using file I/O and each processor autonomously determines for which part of the structured grid it must compute points.
- Agglomeration—there is minimal message passing, and the final results of each computation can be agglomerated and transmitted en masse at the end of the processing step. (no intermediate message passing). This ensures the largest possible message sizes, and streamlines the communications time.
- Mapping—because this is a data-parallel algorithm, and because the chosen approach ensures no data dependencies between individual computations of the distance-weighted mean values, mapping of tasks to processors involves dividing amongst the available processing elements the total number of structured raster points to be computed.

To summarize, the 2D Shepard algorithm for computing the rasterized data from raw data points takes the form of a balanced, data-parallel mapping to the available processors of the raster points to be computed. This yields a maximally efficient division of work and communication. Figure 2 depicts graphically the results of a rasterization on the Gauley River dataset. Note angular artifacts around the edge of the image. These occur in areas of the image containing no LIDAR data points, and are a manifestation of the rasterization algorithm itself.

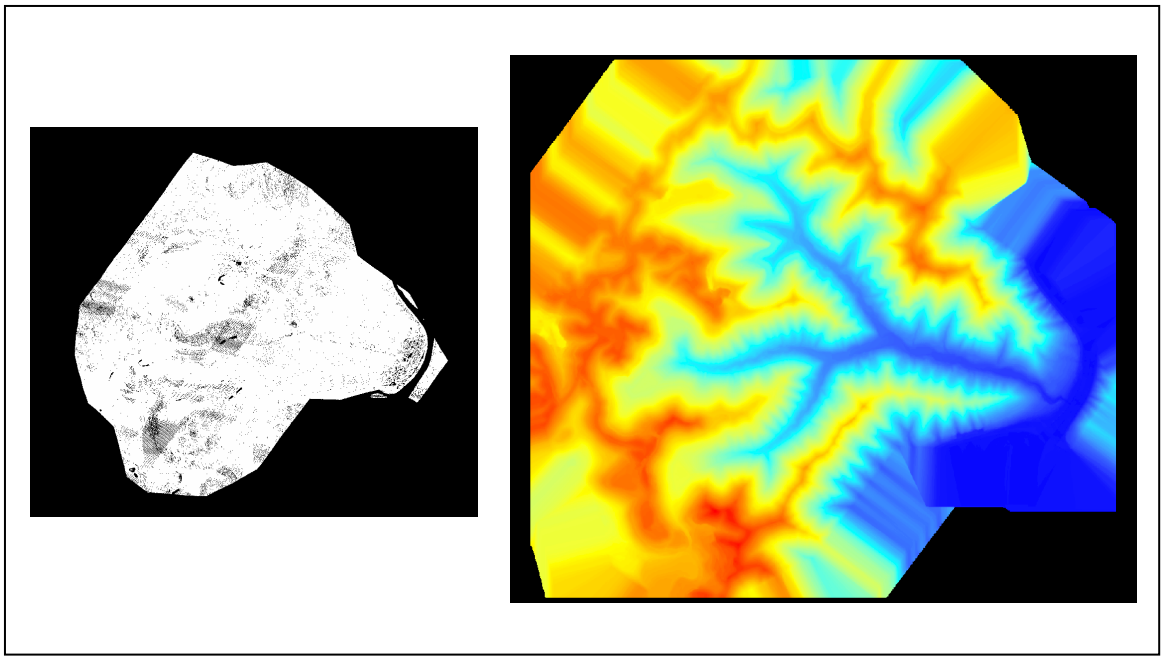

Figure 2.: rasterization of the Gauley River dataset. The left image depicts the raw point cloud, while on the right is the rasterized image, with elevation data mapped to colors.

2.3 Triangulation

The Delaunay Algorithm[3,4] is widely used in computational geometry to compute the simplicial—or unstructured—mesh from a set of points, in this case the raw LIDAR input data. This creates a topological structure viewable as a 3 dimensional image. Triangulation is an integral part of simulation in many applications, such as finite element methods, visualization, gaming, crystallography, and GIS. Thus, there is a broad set of literature exploring parallelization. Blelloch's approach to parallel implementation of the Delaunay triangulation [5] was implemented for this, using the serial Delaunay triangulation found in VTK (visualization toolkit)[8]. Whereas the parallel Shepard's algorithm is a straightforward (coarse-grained) decomposition of the problem, parallel Delaunay triangulation is complicated by data dependencies between partitioned processes. This requires code to be written to ensure proper handling of border information between partitioned processes, and introduces communications overhead. The Blelloch algorithm was chosen for three reasons. First, the algorithm uses a "divide-and-conquer" approach, which is well suited to implementation on a commodity cluster. Second, unlike many other treatments in the literature, the algorithm is detailed in the paper (making it reproducible), and has been implemented and tested with practical problems (millions of points). Finally, the authors provide extensive performance figures and analyses for several systems.
The Blelloch parallelization of the Delaunay algorithm in terms of parallel implementation distills to the following:

- Partitioning—the Delaunay algorithm is medium- to fine-grained parallel: subregions of the dataset of interest have a serial Delaunay triangulation performed upon them.

Partitioning of the problem can be accomplished in many ways, two ways of which were explored:

1. using a kd-tree partitioning approach to evenly subdivide the dataset into balanced subregions
2. using simple division of the data into spatially homogenous rectangular stripes

The simple division was chosen due to the relative regular distribution of LIDAR points in GIS datasets. Thus, load is passively balanced. The Blelloch algorithm identifies the edges between parallel regions by means of the lower convex hull of a series of parabolic projections of the points to be triangulated (detailed algorithmically in [5]). This contrasts with the partitioning step for rasterization in that the Blelloch algorithm is necessary to identify the irregular adjacent edges between parallel subregions. Figure 3 depicts the salient differences between the rasterization and triangulation algorithms in terms of partitioning of data.

- Agglomeration—as with the Shepard algorithm, interprocess communication is minimized during computation, and subimages are agglomerated at completion of processing. Whereas the serial stitching of the Shepard subimages is straightforward (along the linear edges of the subimages), the Delaunay serial stitching is more complex, requiring insertion of the triangles from each subimage into the final mesh.
- Communication—initial communication is very similar to that used in the rasterization algorithm, in that each processor initially needs access to all available data. The edges of adjoining regions are computed according to the Blelloch algorithm, to ensure that borders between distributed subimages utilize the same triangulated vertices (depicted in figure 4).

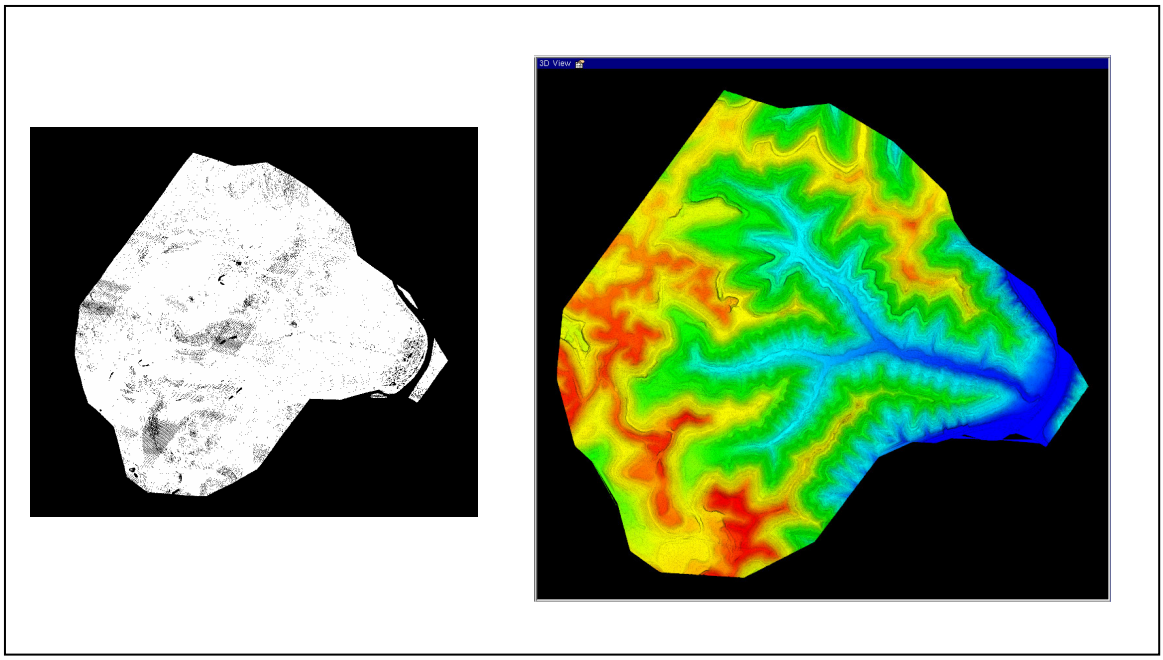

Figure 3.: triangulation of the Gauley River dataset.

3. Discussion of Algorithms

The parallelization described in Section 2 is coarse-grained; data is divided into contiguous subregions and serial algorithms are applied to the subregions. Figure 4 depicts schematically the difference between partitioning of data for the two algorithms. Rasterization uses a simple coarse-grained partitioning whereby adjacent subregions of data are divided into homogenous rectangles. Triangulation is partitioned similarly, but requires the additional step of determining which points fall on triangle edges between adjoining regions.

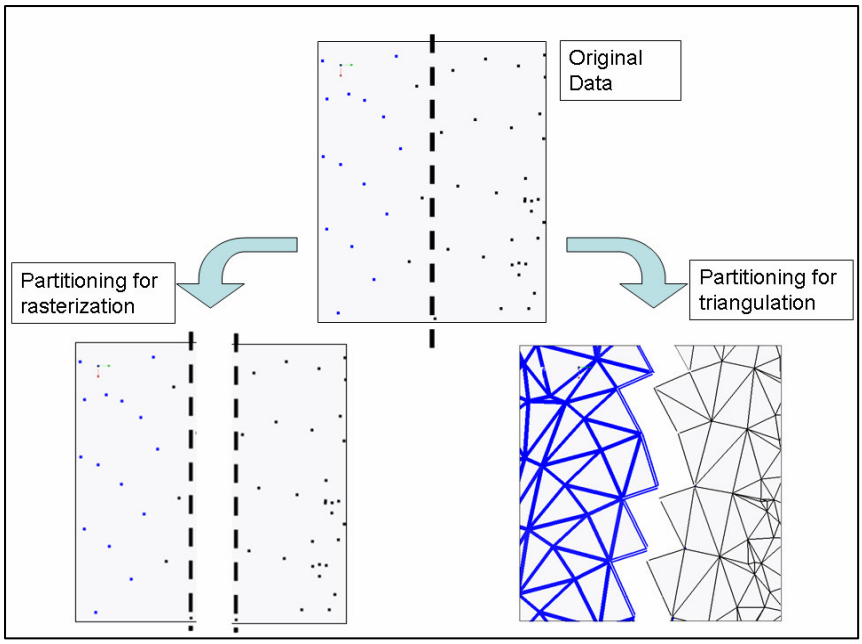

Figure 4. Partitioning of data between processors for rasterization and triangulation, depicted for two processors.

Table 1 summarizes the algorithm partitioning, computing, and agglomerating steps for each of the two algorithms. As described in section 4, the processing component scales well with the number of processors. The nature of LIDAR data presents a complication in terms of partitioning, however. While subregions require spatially contiguous regions of data, the original raw LAS data is not ordered. This is an artifact of the collection process for this data (see Figure 1), in that 1. the routes taken by aircraft are not linear (a criss-crossing flight pattern is required to cover a swath of ground), and 2. the data points are collected by the LIDAR apparatus in a sweeping pattern relative to the aircraft's position.

Due to the unordered LAS format data, the simple partitioning algorithm used in our work requires each processor to read and process the entire data file in order to determine which points lie in that processor's partition. This is required in order to ensure spatially contiguous blocks of data points for application of the serial

computations. Reading of the file by each processor introduces a serialization into the computation, described in Section 4. Due to time constraints, this global reading of the data file was used in the described algorithms. As discussed in future work, a more scalable approach is required for datasets significantly larger than the sample files used in our work, or for significantly more processors involved in the computation.

Table 1. Parallel Algorithms for rasterization and triangulation.

Rasterization	Triangulation
Partition data: 1. Read File across NFS 2. Store points residing in that processor's partition	
Process Points: 1. Apply serial Shepard rasterization 2. Write processed partition	**Process Points:** 1. Determine which triangle vertices lie on edges of partitions (Blelloch algorithm) 2. Apply serial (vtk) Delaunay triangulation 3. Write vertices for partition
Agglomerate data 1. Accumulate processed partitions 2. Assemble final output file output by concatenation of partitions into a VTK structured grid	**Agglomerate data** 1. Accumulate processed partitions 2. Assemble final output file by insertion of points into a VTK unstructured grid.

4. Performance Results

This section describes the datasets used in testing the software, raw performance results, speedup, and efficiency for the parallel algorithms. Also, data is given describing the contribution of file reads to the overall computation time. The rasterization and triangulation algorithms were developed on a Linux cluster of 88 AMD Athlon 1800 processors (44 dual-processor nodes), with Myrinet network interconnect. Access to data files was via NFS (Network File System) on a Gigabit Ethernet interconnect. Each of the 44 nodes has 2GB RAM and a 40 GB hard drive for storage. Development was performed using Linux, standard Linux c++ development tools, VTK 4.0, and MPICH-GM 1.2.5.9 programming libraries.

Table 1 describes the datasets used in development and testing. The datasets range in size from 0.75 - 4.0 million points, representing various locations in West Virginia. The datasets were provided in LAS 1.0 format[10] by West Virginia Department of Environmental Protection, and each comprises information about the spatial extents of the data, the number of data points, position and elevation information for each point. The rasterization and triangulation algorithms were run for each dataset on a range of processors. The execution times were recorded in seconds, reported in Tables 2 and 3.

In order to examine the contribution of the serial data file reads to the observed total execution time for each set of runs (described in Section 3), test runs were conducted for the Gauley River dataset. For the range of processors used in the speedup and efficiency computations, the read time for each processor was measured and reported as an average. In between runs, the NFS cache and RAM were cleared by reading a 1 GB file on each processor (equal to the RAM available per processor). The fraction of the total computation time required for the file read step is reported in Table 4.

Table 2. Dataset Statistics

Dataset	Smithfield	Brisco	Clearfork	Gauley	Coal River
Mbytes	80.5	78.4	29.3	14.9	18.0
Points	4,024,706	3,921,654	1,465,915	747,449	901,100
Triangles	7,980,898	7,815,415	2,930,756	1,493,976	1,801,130

Table 3. Execution Times for Rasterization on the Cluster, in seconds

# Processors	Smithfield	Brisco	Clearfork	Gauley	Coal River
1	84,057.40	20,106.88	32,341.29	7,650.71	21,202.93
2	49,514.85	10,950.66	12,611.43	3,925.67	12,512.45
4	29,080.98	6,245.29	7,980.58	2,959.32	6,886.66
8	15,653.34	2,844.49	4,310.49	1,433.22	2,263.89
16	5,333.35	832.70	1,707.52	520.78	598.17
32	1,774.23	170.03	480.14	142.86	170.95
64	543.79	92.46	181.74	73.08	92.53

Table 3. Execution Times for Triangulation on the Cluster, in seconds

# Processors	Smithfield	Brisco	Clearfork	Gauley	Coal River
1	20,679.00	12,096.40	5,414.38	2,083.26	4,225.21
2	12,122.17	6,209.70	3,230.59	1,102.20	2,252.20
4	7,352.22	3,178.61	1,607.16	687.62	689.96
8	4,831.39	1,695.28	850.40	345.27	340.54
16	2,398.71	1,126.23	687.07	177.94	205.41
32	1,257.61	537.14	220.77	122.57	131.10
64	666.71	352.40	167.22	74.02	121.87

Table 4. Serial reads as a fraction of total wall clock time, for Gauley dataset.

# Processors	Execution Time, in sec.	Mean File Read Time / Processor, in sec.	File Read Time/ Execution Time
1	7,650.71	0.757747	9.90E-05
2	3,925.67	0.7303	1.86E-04
4	2,959.32	0.60914	2.06E-04
8	1,433.22	0.318996	2.23E-04
16	520.78	0.23704	4.55E-04
32	142.86	0.210337	1.47E-03
64	73.08	0.251829	3.45E-03

Speedup plots for each algorithm are shown in Figures 5 and 6. Both algorithms exhibit near-linear speedup using the test datasets. The apparent super-linear speedup is attributable to decreasing subregion size and memory footprint, for fixed data size on an increasing number of processors. On a small number of processors, significant swapping to disk occurs, while this effect is diminished as the number of processors is increased. This effect is reflected in the efficiency plots in Figures 7 and 8, as well, especially with the rasterization algorithm, in which there is apparent efficiency > 100% as the number of processors increases. The lack of smoothness in the speedup curves is related to the assumption that data points are uniformly spaced in the datasets. As an example, the Coal River dataset is irregularly shaped. For this dataset, the partitioning approach used in our algorithms yields inconsistent distribution of work to processors as compared the distribution of work for a uniformly spaced set of points.

Figure 9 depicts a plot of the contribution of the average file read times to the total computation time as a fraction, for the Gauley dataset. As described in Section 3, each processor reads the entire data file to identify which points belong to that processor's partition for computation. While this read time is quite small for the dataset sizes and number of processors used in our work, it is clear that the fractional contribution of file read times increases substantially with the number of processors. The read times introduce a bottleneck into the computation which further explains the tendency of the efficiency plots to taper asymptotically. As the size of our datasets increases in the future, the read time will become a substantial contribution. Also, extrapolating forward from these measurements, the file read time becomes significant at between 2048 and 4096 processors (approaching 0.10, or 10%). This issue is addressed in section 5, future work.

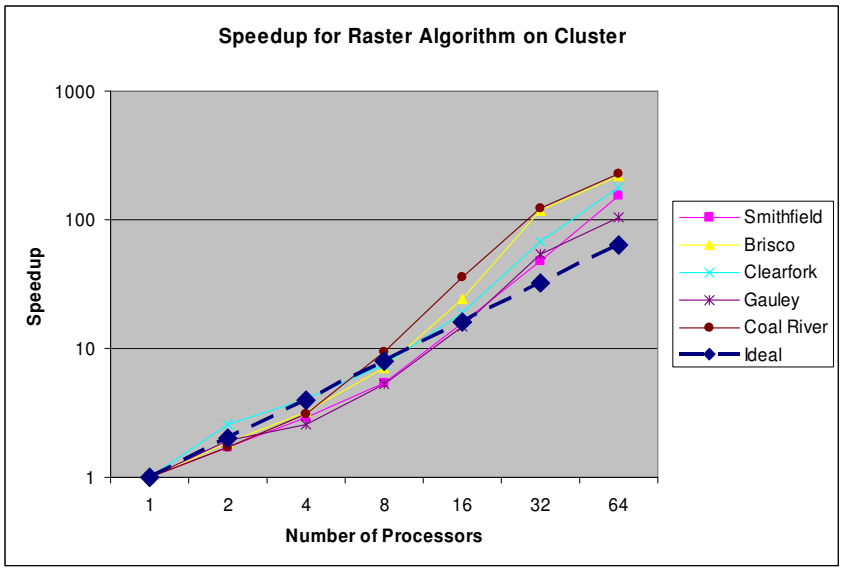

Figure 5. Speedup for parallel Raster Algorithm.

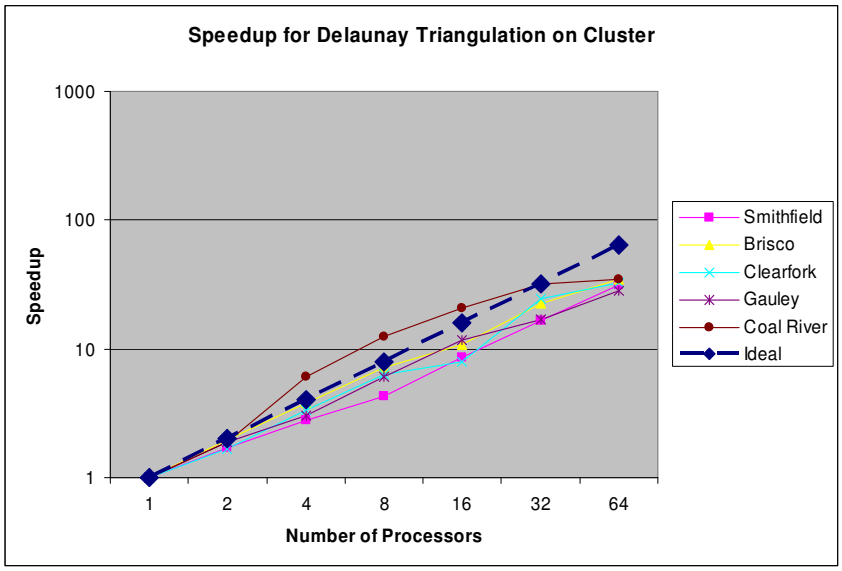

Figure 6. Speedup for Delaunay Triangulation on Cluster

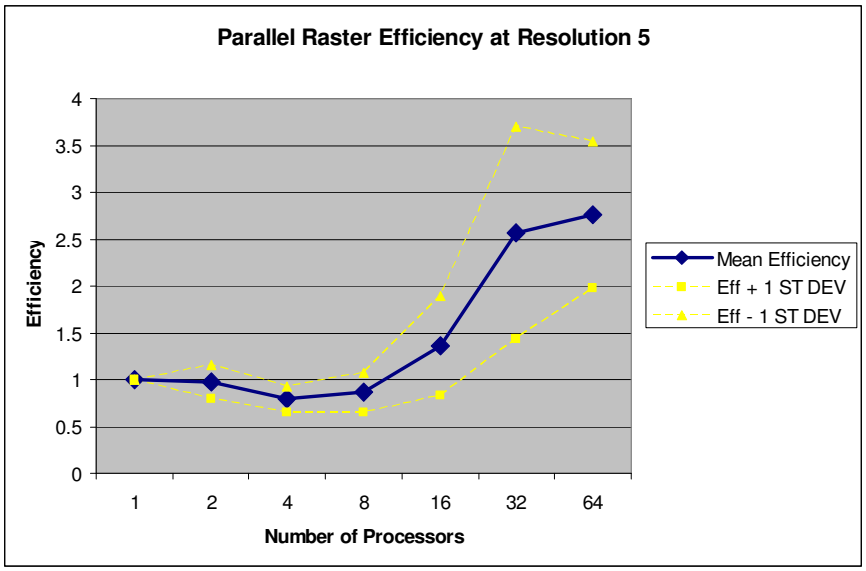

Figure 7. Efficiency for Rasterization on Cluster

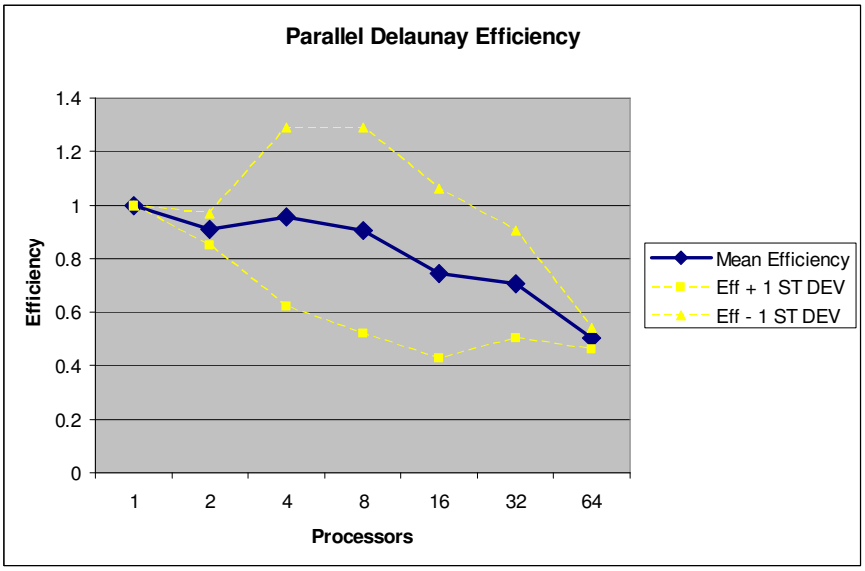

Figure 8. Efficiency for Delaunay Triangulation on Cluster

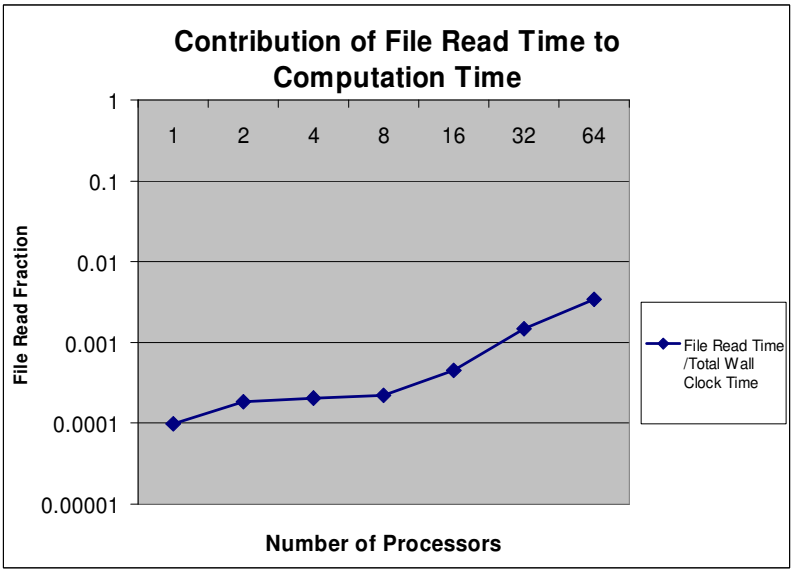

Figure 9. Contribution of file read times to observed computation time.

5. Conclusions and future work

We presented the results of processing large LIDAR datasets on a Linux cluster. Two conventional image processing algorithms were studied, parallelized and run with a range of datasets. Speedup measurements indicate near-linear performance across the datasets for both algorithms. As LIDAR datasets continue to grow in size, and as commodity supercomputing clusters become more available, these are promising results for the GIS community. Whereas rasterization of our largest dataset required almost 24 hours on a single processor, rasterization of the same dataset on 64 nodes required approximately 10 minutes. Similarly, triangulation of the same dataset required almost 6 hours in a single processor versus about 10 minutes on 64 processors. While these figures certainly make processing more tractable from the user's perspective, future work is necessary to improve computational performance. Our results point out several directions for future work:

First, there is clearly a serial contribution to the computation resulting from reading of the data file in its entirety by each processor. While pre-processing of the file is required due to the unordered nature of LIDAR data files, there are more sophisticated approaches to partitioning. A pre-processing step which partitions the data in parallel is one candidate, and would obviate the need for each processor to read the entire file to locate its own partitioned data.

Second, the datasets used in this application were not scaled in size with the number of processors. In order to truly portray the speedup and scalability for this implementation, appropriate datasets need to be located or created.

Third, the points in the datasets used in this study was relatively uniformly spaced. Non-uniformly spaced data requires a more sophisticated partitioning approach. Future work will concentrate on load balancing by dividing the image into regions by data distribution rather than by regular rectangular intervals. This should yield more consistent results and smoother curves, independent of the distribution of points in the LIDAR datasets.

Finally, the work described here is part of a larger project, in which a variety of feature extraction and imaging algorithms have been developed for LIDAR data (e.g. building identification). Future work on this front will involve parallelizing and performance-tuning these additional algorithms. Also, the coarse-grained approach we have used shows promise for deployment across clusters on a grid.

References

[1] W.J. Gordon and J.A. Wixom. "Shepard's method of metric interpolation to bivariate and multivariate interpolation," *Mathematics of Computation*, 32(141):253--264, 1978

[2] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," *Proc. 23rd National Conference ACM*, ACM, 517-524, 1968

[3] J. O'Rourke "Computational Geometry In C (Second Edition)", Cambridge University Press, 1998.

[4] B. Delaunay, 1934. "Sur la sphère vide," *Bulletin of the Academy of Sciences of the U.S.S.R.* Classe des Sciences Mathématiques et Naturelle, Series 7 (6), 793-800.

[5] G.E. Blelloch, et al., "Design and Implementation of a Practical Parallel Delaunay Algorithm," *Algorithmica*, 24:243-269, 1999

[6] R. Buyya, "High Performance Cluster Computing, Volume 2, Programming and Applications", Prentice Hall, NJ, 1999.

[7] M. Flood, B. Gutelius, and M. Orr, 1997, "Airborne: Terrain Mapping," *Earth Observation Magazine*, Feb 1997.

[8] http://public.kitware.com/VTK/

[9] G.M. Nielson, T.A. Foley, Bernd Hamann, and D.A. Lane , "Visualizing and modeling scattered multivariate data", *IEEE Computer Graphics and Applications*, 1991, 11(3): 47-55.

[10] http://www.lasformat.org/