Parallelized Interpolation: A Quantitative Assessment

Scott Blaha
Swarthmore College

Mustafa Paksoy
Swarthmore College

Abstract

The conversion of raw point-cloud elevation data to grid DEMs is done by interpolation. Current interpolation methods produce either high quality results or take an acceptable amount of time, but not both. This paper seeks to reconcile these two objectives through parallelizing a high-quality interpolation method, nearest-neighbor averaging. We explore the speed-up obtained by parallelization and compare run-time with the lower quality binning method.

1 Introduction

LIDAR, one of the primary forms of elevation data collection, yields a cloud of elevation data points. However, Geographic Information Systems (GIS) like GRASS often require data in the form of a digital elevation model (DEM). One of the simplest methods to perform this conversion is called *binning*. Binning simply averages the points in each grid cell of the DEM to yield an elevation for the grid cell. However, because of the non-uniform nature of the point cloud, some of the grid cells of the DEM might not contain any points. Thus, it is possible to have holes in the DEM after binning.

The solution to this is to use one of a set of methods known as *interpolation*. A typical simple linear interpolation might take an average of points close to an empty cell, weighted by distance from the cell. Unfortunately, if there are n points, then there are potentially $O(n^2)$ interpolation calculations to be performed. This fact, paired with the typically extremely large data sets of interest in GIS, makes interpolation a highly non-trivial task. In fact, in a recent I/O-efficient point cloud to DEM algorithm (0), from 52% to 86% of running time was spent interpolating depending on the data set. Clearly, a faster method of interpolation is needed.

The basic trade-off in interpolation is quality (e.g. representativeness) of the resulting DEM versus the computational complexity of the interpolation. Rather than deal with reduced DEM quality in our quest for better interpolation run-times, we will parallelize the interpolation of point cloud elevation data. Because of the locality of reference of the interpolation task, parallelization can provide an exponential reduction of the time to interpolate a set of points, based on the number of computers.

2 Methods

2.1 Serial Binning

Our first method is a simple implementation of serial binning. A grid cell's value is the average of all points from the point cloud in that cell. A grid cell containing no points is assigned a "no value" constant; in our case, this was -99999. Serial binning will be our base-line for comparison of competing interpolation methods, both in terms of interpolation quality and run-time efficiency. We hope that parallelization will speed up creating a DEM, and that interpolation will improve the quality of the DEM. Since it takes constant time to place a point into a bin, the run time complexity of binning is $O(n)$,

where n is the number of input points. However, the constant hidden in this notation was experimentally shown to be quite small.

2.2 Parallelized Binning

Next, we implemented a parallelized binner. This simply splits the task of binning and averaging points between several computers. It will produce the same interpolation as a serial binner, but hopefully with a slight efficiency boost. We do not expect to see much improvement by using this method.

Our parallel applications use the Message Passing Interface (MPI) standard for distributing and collecting data. MPI provides various facilities for passing data around and synchronizing computation. In our case, we just needed to send data back and forth. We use the Local Area Multi-computer (LAM) implementation of MPI, it lets us set up virtual clusters using an arbitrary number of nodes in the Computer Science Department network. These machines are all on the same subnet, so we expect network bandwith to be high and latencies to be low.

Initially, we split data into equal size y-intervals. This led to different hosts creating overlapping grids, which greatly complicates the process of merging results. To ameliorate this, we increase the y-interval so it becomes a multiple of the bin height of the grid (see Figure 1). So, the splits in the data align with the grid, which prevents different hosts from creating overlapping grids.

2.3 Nearest-Neighbor Interpolation

We have implemented a simple type of interpolation, *nearest-neighbor* interpolation. In this method, we first calculate the *centroid* of each grid cell, the center point of the cell. Then, we sort points by their Euclidean distance from the centroid. Finally, we set the elevation of the grid cell to be the average of the k points nearest to the centroid. Experimentation showed that 10 is an acceptable value for k.

Because sorting is $O(n \log n)$, if we have g grid cells and n points, then this algorithm has a run-time complexity of $O(g \cdot n \log n)$. This is because we must sort all the points based on distance from each cell's centroid. As we note below, this method is unacceptably slow. However, it does not suffer from the "holes" in the grid that binning does.

2.4 Parallelized Nearest-Neighbor Interpolation

Parallelization of our interpolation algorithm proceeds in much the same way as with simple binning: we break the grid into a number of approximately equal sections, one for each host, and send each host only the points which fall in that bin. Each host then performs the nearest-neighbor interpolation described above.

The theoretical speed-up we should see is more than quadratic in the number of hosts. If we have h hosts, and the points are approximately evenly distributed over the grid, then each host will get about $\frac{g}{h}$ of the grid and $\frac{n}{h}$ of the points. So, each host will have a runtime complexity of $O(\frac{g}{h} \cdot \frac{n}{h} \log \frac{n}{h}) = O(\frac{gn}{h^2} \log \frac{n}{h})$. Our tests have supported this analysis: we do in fact get super-quadratic speedups from adding hosts (see Section 3).

2.5 Smoothing Parallelized Interpolation

Unfortunately, parallelization can result in edge effects in interpolation results. Along the edge of a sub-grid sent to a host for interpolation, the closest elevation data points to a centroid might be in a different sub-grid, and thus are not considered during the interpolation. This can result in noticeable lines or bands in the resultant DEM. We have called our solution to this *smoothing*. We pass the points belonging to the immediately surrounding sub-grids along with the points in the sub-grid we send to each host. This approximately triples the number of points sent to each host, but results in less noticeable edge effects.

3 Results and Discussion

We have fully implemented the algorithms described above. We ran tests on a 100,000 point subset of a LIDAR-generated 100 foot resolution point cloud. This subset was picked by sorting the 2,000,000 points by y-value, and picking every twentieth. See Figure 2 for visualizations of the interpolated data set. Interpolation is the clear winner in the quality department - there are no holes, and the resultant DEM looks like a "filled-in" version of the binned DEM (Figure 2(a)). It is hard to notice the difference between smoothed and non-smoothed DEMs with the naked eye, but Figure 2(d) shows the result

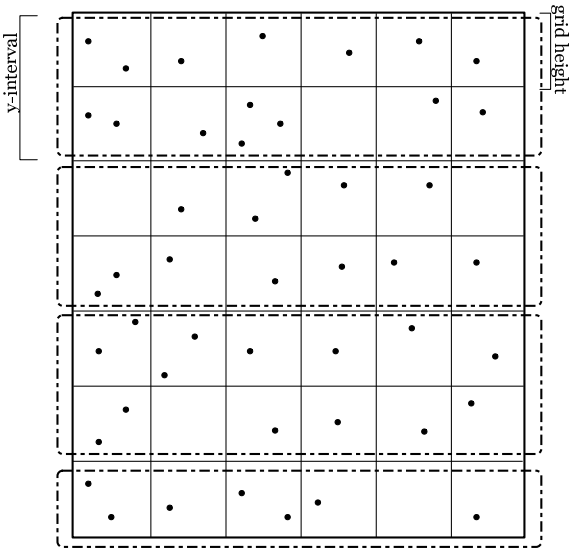


Figure 1: Splitting the grid into sub-grids for parallelization

of subtracting the two DEMs from each other. Notice that the difference lies along the cuts between the sub-grids sent to different hosts. This shows that smoothing actually does what it is supposed to, that is, it smooths out the edge effects between sub-grids.

The actual running time of each algorithm was calculated several times to test the possible speed-ups obtainable by parallelization. Unexpectedly, parallel binning is actually slower than serial binning (see Figure 3). We believe this in due to the overhead of passing data over the network compared with the blazing speed of the simple binning algorithm. In the case of interpolation, we observed the extreme parallelization speed-ups predicted above. See Figure 4 for a chart of running time versus number of hosts - note that the time axis is logarithmic. So the predicted super-quadratic speed-up does occur. As a simple comparison between binning and interpolation, with 100,000 points, 20 hosts, and a 100 foot resolution, binning takes 1.4 seconds, non-smoothed interpolation takes 24.8 seconds, and smoothed interpolation takes 73.3 seconds. Refer to Figure 5 for a comparison between the three methods

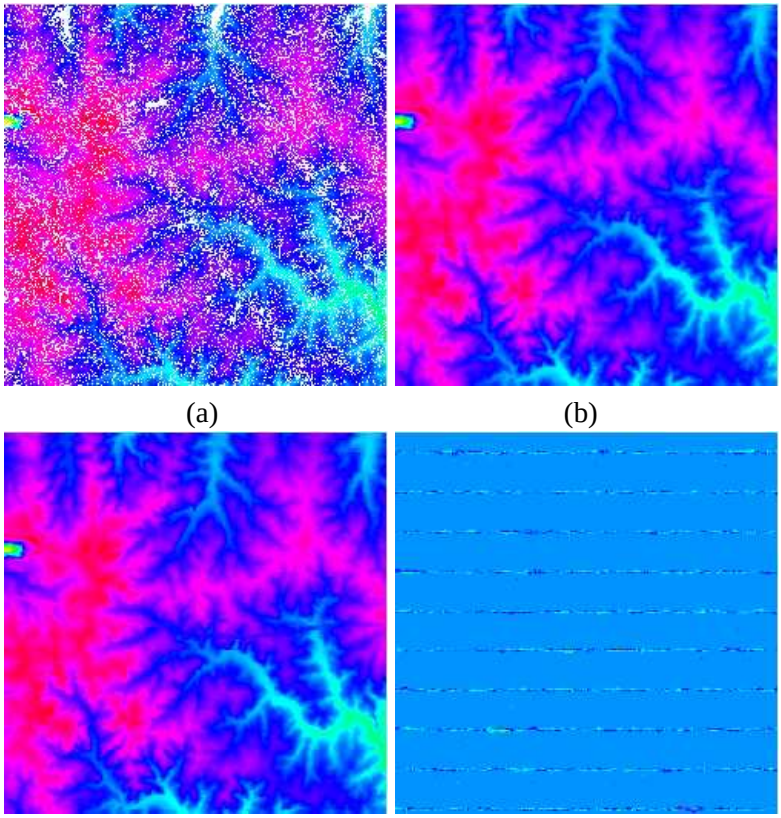
4 Conclusion

Parallelization provides an excellent speed-up for interpolation methods, but does not decrease the running time of binning. Before parallelization, our interpolation method was intolerably slow, but with 20 hosts, its runtime is very reasonable. Smoothing is also a desirable option when using parallelized nearest-neighbor interpolation, however it approximately triples the run time of the interpolation. For casual use, parallelized non-smoothing interpolation suffices. However, if more accuracy is required, then smoothing provides that with only a three-fold slowdown.

Future work in this area could include parallelizing the extremely popular quad-tree interpolation algorithm. Also, we can optimize the nearest neighbor interpolation by using a scan-line approach and only sorting a section of the whole data set a time.

References

P. K. Agarwal, L. Arge, and A. Danner. From point cloud to grid DEM: A scalable approach. In Andreas Riedl, Wolfgang Kainz, and Gregory Elmes, editors, *Progress in Spatial Data Handling. 12th International Symposium on Spatial Data Handling*, pages 771–788. Springer-Verlag, 2006.



(a) (b)

(c) (d)

Figure 2: Results of interpolating sparse data at 100 feet. (a) is the result of binning - note the cells with no value. (b) is our parallel interpolation algorithm without smoothing, and (c) is with smoothing. (d) is the difference between (b) and (c).

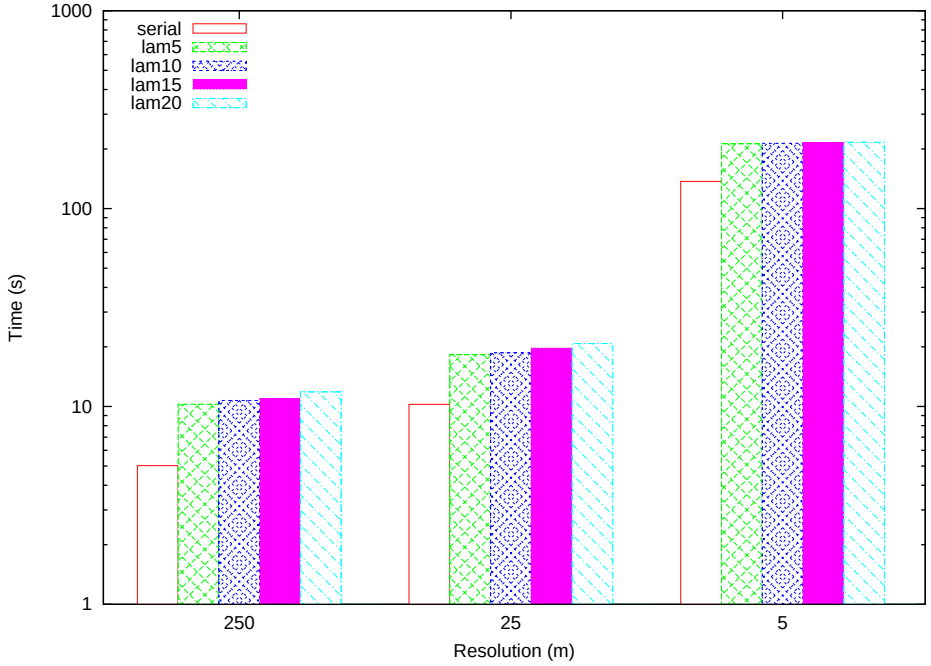


Figure 3: Comparison of serial and parallel binning running times.

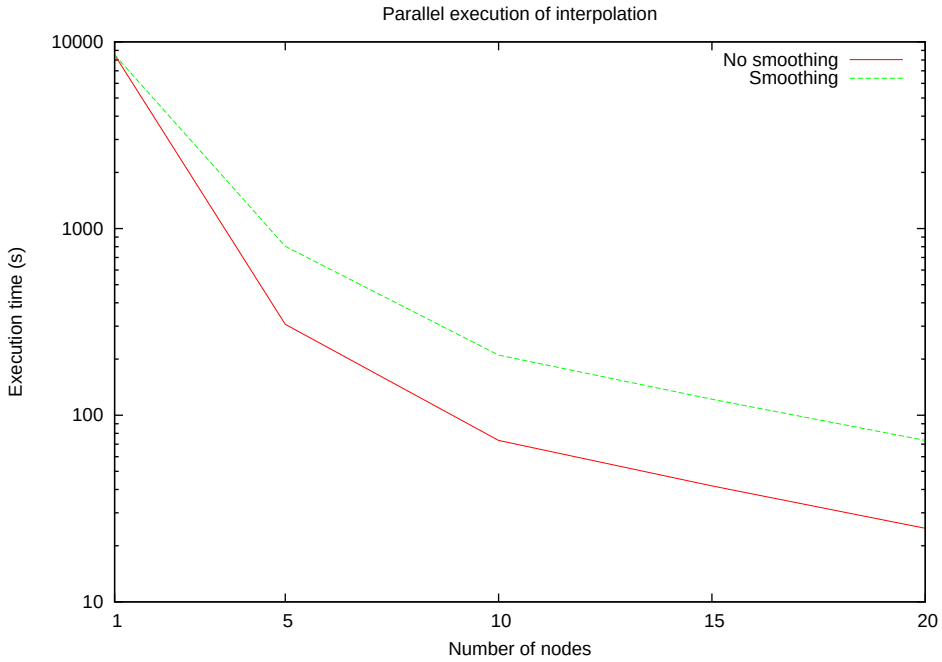


Figure 4: Comparison of non-smoothed and smoothed interpolation running times, showing super-quadratic speed-up for adding hosts.

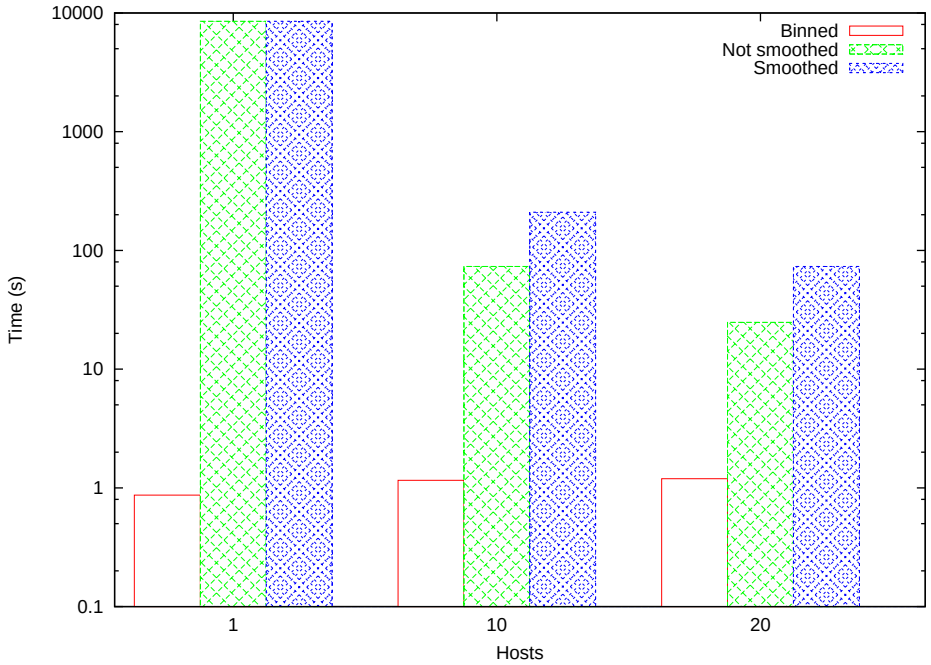


Figure 5: Comparison of parallel binning, non-smoothed interpolation, and smoothed interpolation running times.