LAS File Processing with HPC

- Problem: Make high resolution DEMs from large lidar datasets.
- Approach: Modify open source software to run in parallel and test on XSEDE clusters.

Make high resolution DEMs from large lidar datasets.

- DEM resolution of 1 meter.
- Lidar datasets with coverage over about a 15x15 minute footprint.
- This implies a raster size of about 27000 rows X 27000 columns, around 750 million cells.

Make high resolution DEMs from large lidar datasets.

- Obtained two datasets in LAS 1.2 format
- A 16 GB file with 500 million points in the Smoky Mountains over a 40000 X 20000 meter area.
- A 117 GB file with 4GB points over the Grand Canyon over a 25000 X 30000 meter area.
- Both files are somewhat sparse in that some “tiles” within the coverage are missing.

Modify open source software to run in parallel and test on XSEDE clusters.

- Las2las from the lastools suite to filter all but ground points.
- Points2grid to make the DEM from the filtered LAS file.
- Test on Stampede, an XSEDE cluster at the Texas Advanced Computing Center. https://portal.xsede.org/tacc-stampede

XSEDE – Extreme Science and Engineering Development Environment

- A five year 127M dollar project funded by NSF
- Supports 16 supercomputers and high-end visualization and data analysis resources.
- One of those is “Stampede” at TACC
- Accessed through “Campus Champion” allocations
- USGS and University of Illinois are Champions

What is a “Supercomputer” Anyway?

- Collection of nodes (Xeon X5 based computers)
- Networked together (Infiniband)
- Running a common OS (Linux)
- Shared parallel filesystem (Lustre).
- Running a scheduler. (SLURM)
- An IPC mechanism. (MPI... MPI_Send, MPI_Recv, MPI_File_write_at ...)
- A good example is Stampede.

p_las2las Implementation

- las2las application and supporting LASlib library were extended with the MPI API to allow the application to be run in parallel on a cluster.
- Goal was an application that would scale to arbitrarily large input, limited only by the amount of disk space needed to store the input and output file.
- No intermediate files are generated and individual process memory needs are not determined by the size of the input or output.

p_las2las Implementation

Native las2las algorithm

- For all points:
- Read the point
- Apply a filter and/or transformation
- Write the possibly transformed point if it passes the filter.

p_las2las Implementation

p_las2las algorithm

- Processes determine point range and set input offsets
- For all points in a process's point range:
  Read point and apply filter.
  Keep count of points that pass filter.
- Processes gather filtered point counts from other processes.
- Processes can then set write offsets.
- Processes set read offsets back to beginning point, begin second read for all points:
  Read point and apply filter and transformation.
  Write the possibly transformed point if it passes the filter.
- Gather and reduce point counts, return counts, min and max x values
- Update the header with rank 0 process.

p_las2las Implementation

Detailed Explanation of the p_las2las Implementation

Each process opens the LAS input file and reads the file header to determine the number of points in the input file, point size, and size of the header. Based on the point count, process rank and process count, each process calculates the range of LAS points for which it will be responsible. Since point and the header size are known, each process can calculate and set its file pointer to its beginning point.

Each process then reads each point in its range and applies any filter passed to the program, keeping a count of points that pass the filter. After reading and filtering the last point, all processes gather from one another the number of points that have passed the filter, and thus will be writing. Each process uses the results from this gather and its rank order to calculate and set its output file pointer.

Each process then sets its read pointer back to the beginning of its range of points. A second read and filtering of its point range begins, but this time the points that pass the filter are written to the output file. It is this second read pass that allows the program to scale to arbitrary input and output size without allocating extra memory or writing temporary files.

The process with rank 0 is charged with writing the output file header with data gathered from the input header, and gathering and reducing process dependent data such as minx, maxx, miny, maxy, minz, max from the other processes. To minimize the number of calls to MPI_File_write, each process allocates a buffer of configurable size and only calls MPI_File_write when its buffer is full, along with a final flush to disk after the last point is processed.

p_las2las Results

Smoky Mountains 16 GB File:

Processes	Filter / Transformation	Output Size	Elapsed (seconds)
Native	None	16 GB	138
Native	Keep Class 2	2 GB	73
Native	Reproject	16 GB	502
64	None	16 GB	20
64	Keep Class 2	2 GB	6
64	Reproject	16 GB	26
256	None	16 GB	8
256	Keep Class 2	2 GB	4
256	Reproject	16 GB	9
1024	None	16 GB	8
1024	Keep Class 2	2 GB	5
1024	Reproject	16 GB	8

p_las2las Results

Grand Canyon 117 GB File:

Processes	Filter / Transformation	Output Size	Elapsed
Native	None	117 GB	1211
Native	Keep Class 2	25 GB	623
Native	Reproject	117 GB	6969
64	None	117 GB	128
64	Keep Class 2	25 GB	59
64	Reproject	117 GB	150
256	None	117 GB	33
256	Keep Class 2	25 GB	18
256	Reproject	117 GB	42
1024	None	117 GB	18
1024	Keep Class 2	25 GB	9
1024	Reproject	117 GB	24

p_points2grid Implementation

- points2grid application was extended with the MPI API to allow the application to be run in parallel on a cluster.
- Goal was an application that would scale to arbitrarily large input, limited only by the amount of disk space needed to store the input and output file and the number of processes available.
- No intermediate files are generated and individual process memory needs are not determined by the size of the input or output.

p_points2grid Implementation

Native points2grid algorithm

For each point:

- Update output raster cells when the point falls within a circle defined by the cell corner and a given radius.
- Optionally fill null cells with adjacent cell values.
- Write the output raster.

p_points2grid Implementation

p_points2grid algorithm

- Processes are designated as reader or writer.
- Reader processes are assigned a range of points.
- Writer processes are assigned a range of rows.
- Reader processes read LAS points from the input file and sends them to the appropriate writer processes based on whether the point falls within a circle defined by the cell corner and a given radius.
- Writer processes receive LAS points from reader processes and update cell contents with elevation values.
- When all points have been sent and received, Writer processes apply an optional window filling parameter to fill null values. (This involves writer to writer communication when the window size overlaps two writers.)
- Writer processes determine and set write offsets and write their range of rows to the output file. The first writer rank is responsible for writing the output file header and in the case of tiff output, the tiff directory contents.

p_points2grid Implementation

Detailed Explanation of the p_points2grid Implementation

Initialization
Each reader process allocates a LAS point buffer for each writer process. These buffers are necessary to keep process-to-process communication at reasonable levels, since without them an MPI send/receive would occur with every LAS point. The size of these buffers is dependent on the writer count and is calculated and capped at run time so as not to exceed available memory.

Each writer process allocates memory to hold the grid cell values for the rows for which it is responsible. The row count is determined by the number of rows in the grid divided by the writer process count. This introduces a memory dependency that our current implementation does not address. As a practical matter, the number of writer processes can be increased to address this limitation. Each writer process also allocates write buffers of configurable size to limit the number of calls to MPI_write. When a window filling parameter is specified, writer processes allocate and fill two dimensional raster cell buffers of up to three rows before and after their range. This is necessary to keep process-to-process communication at reasonable levels.

Reading and Communication:
Each reader process calculates and sets its input file pointer to the the beginning of its range of points. It then reads each point and determines which raster cells have overlap with the point and a circle defined by the cell corner and a radius given either by default or as a program input parameter. For each overlap, the point is added to the appropriate LAS point buffer. That is, the buffer corresponding to the writer responsible for processing that cell. When a buffer fills, it is sent with a MPI_Send to the appropriate writer. The writer receives the buffer with MPI_Recv and updates its raster cell values with the point buffer data. When a reader process completes, it flushes its point buffers to all writers one last time.

Writer Processing:
Once all points have been received by the readers, Each writer iterates over their raster cells and calculates mean and standard deviation values. If a window size parameter has been passed, each writer iterates over its cells and attempts to fill null cell values with weighted averages of values from adjacent cells up to three cells away. When cells fall near a writer's beginning or ending row, these values are retrieved from the rows of adjacent writer processes.

Writing:
Each writer first determines the total number of bytes it will write for each DEM output type and each raster cell type, Since the output types supported are ASCII text, each writer must iterate over its cell values and sum their output length. Once all writer's have determined their output length counts, each gathers these counts from all other writer processes and uses these counts along with rank order and header size to set output file pointer positions. Each process then iterates over its values again, but this time writes the ASCII format of the value to a buffer. When the buffer fills it is written to disk with MPI_File_write. The first writer process is responsible for writing the file header.

p_points2grid Results

Smoky Mountains 16 GB Input File, (12, 1 meter resolution DEMs totaling 70 GB of output for p_points2grid runs. 12, 6 meter resolution DEMs totaling 2 GB of output for native run.)

Processes	Readers	Writers	Reading, Communication	Writing	Elapsed
Native	1	1	NA	NA	328
128	32	96	33	56	105
128	64	64	26	84	125
512	32	480	20	13	40
512	64	448	10	17	33
512	128	384	8	23	36
512	256	256	7	26	40
512	384	128	11	44	68
1024	64	940	10	11	32
1024	384	640	2	14	29
1024	768	256	6	28	46

p_points2grid Results

Grand Canyon1 117 GB Input File, (12, 1 meter resolution DEMs totaling 71 GB of output for p_points2grid runs.
12, 6 meter resolution DEMs totaling 2 GB of output for native run.)

Processes	Readers	Writers	Reading, Communication	Writing	Elapsed
Native	1	1	NA	NA	1548
512	64	448	104	15	135
512	128	384	55	21	90
512	256	256	60	30	110
1024	64	960	80	7	101
1024	128	896	39	11	62
1024	256	768	27	8	51
1024	384	640	24	15	53
1024	512	512	26	19	56
1024	768	256	47	24	90
1024	896	128	89	44	167
4096	256	3840	17	10	63
4096	512	3584	10	11	53
4096	1024	3072	8	8	46
4096	2048	2048	8	18	76