Evaluation of MapReduce for Gridding LIDAR Data

Sriram Krishnan, Ph.D.,
Senior Distributed Systems Researcher,
San Diego Supercomputer Center
sriram@sdsc.edu

Co-authors: Chaitanya Baru, Ph.D., Christopher Crosby

IEEE CloudCom, Nov 30, 2010
Talk Outline

• Overview of OpenTopography & LIDAR
  • Introduction to Digital Elevation Models (DEM)
• DEM Algorithm Overview
• C++ and Hadoop Implementation Details
• Performance Evaluation
• Discussion & Conclusions
The OpenTopography Facility

- High resolution topographic data: [http://www.opentopography.org/](http://www.opentopography.org/)
  - Airborne
  - Terrestrial*

* Terrestrial data is currently limited to lidar datasets.
Introduction to LIDAR

- **LIDAR**: Light, Detection, and Ranging
- Massive amounts of data
- Real challenge to manage and process these datasets

*D. Harding, NASA*
Digital Elevation Models (DEM)

- Digital continuous representation of the landscape
  - Each (X, Y) is represented by a single elevation value (Z)
  - Also known as Digital Terrain Models (DTM)
- Useful for a range of science and engineering applications
  - Hydrological Modeling, Terrain Analysis, Infrastructure Design

Mapping of non-regular LIDAR returns into a regularized grid*
DEM Generation: Local Binning

Full-featured DEM  Bare earth DEM
C++ In-Core Implementation

- Initialize – allocate memory for grid cells
- For all elements in the point cloud dataset
  - Update \{min, max, mean, idw, count\} values for all neighboring grid cells, based on search radius
  - Generate DEM’s in requested formats from the in-memory grid

\( O(N) \)

\( O(G) \)
C++ Out-of-Core Implementation

• Sorted
  • Input read: O(N)
  • Each block is processed once: (O(\(\frac{G}{M}\)•M))
  • Sum: O(N+G)

• Unsorted
  • Input read: O(N)
  • Block processing: (O(\(\frac{G}{M}\)•M))
  • Swapping overhead:
    • O(f•(\(\frac{G}{M}\)•(Cwrite_M+Cread_M)))
  • Sum: O(N+\(\frac{G}{M}\)•M+f•(\(\frac{G}{M}\)•(Cwrite_M+Cread_M)))

Initialize:
• Split the grid cells into multiple blocks, with overlaps
• Load first block into memory

For all elements in the point cloud dataset

Neighbor grid cell in memory?
Yes

Add point to the queue for the \(i^{th}\) block
No

Update \{min, max, mean, idw, count\} values for all neighboring grid cells

Queue length > threshold?
No

Yes

Swap \(i^{th}\) block into memory, and update grid cells from its queue

Flush all queues and generate DEM’s in requested format
Hadoop Implementation

Assignment to local bin: $O(N/M)$

Z value generation: $O(G/R)$

Output Generation: $O(G \cdot \log G + G)$
Experimental Evaluation: Goals

• Evaluation of C++ and Hadoop performance for different data sizes and input parameters
  • Size of input point cloud versus grid resolution
  • Similarities and differences in the performance behavior
• Performance (and Price/Performance) on commodity and HPC resources
• Implementation effort for C++ and Hadoop
Experimental Evaluation: Resources

- **HPC Resource**
  - 28 Sun x4600M2, eight-processor quad-core nodes
    - AMD 8380 Shanghai 4- core processors running at 2.5 GHz
  - 256GB-512GB of memory
  - Cost per node around $30K-$70K USD each

- **Commodity Resource**
  - 8-node cluster from off-the-shelf components
  - Quad-core AMD PhenomTM II X4 940 Processor at 800MHz
  - 8GB of memory
  - Cost per node around $1K USD each
Experiment Evaluation: Parameters

• Four input data sets – 1.5 to 150 million points
  • From 74MB to 7GB in size
  • Overall point density – 6 to 8 per sq meter

• Three different grid resolutions (g)
  • 0.25m, 0.5m, 1m
  • Modifying the resolution from a 1x1m to 0.5x0.5m quadruples the size of the grid
Discussion - Bottlenecks

- **Hadoop**
  - Output generation (which is serial) accounts for around 50% of total execution time for our largest jobs
    - If there is more than one Reducer, the outputs have to be merged and sorted to aid in output generation
  - If not for the output generation phase, the implementation scales quite well

- **C++**
  - Memory availability – or lack thereof is the key factor
  - Size of the grid is a bigger factor than the size of the input point cloud
  - If jobs can be run in-core, then the performance is significantly better
Discussion - Performance

- Raw Performance
  - Hadoop implementation on commodity resource is significantly faster than the C++ version for large jobs on the same resource
  - However, it is still slower than the C++ version on the HPC resource
    - If the C++ jobs can be run in-core, it is faster than the Hadoop version

- Price/Performance
  - Performance on commodity resource is the same order of magnitude of the HPC resource
  - But a 4-node commodity cluster costs an order of magnitude less
Discussion – Programmer Effort

• Hadoop version more compact
  • 700 lines of Hadoop (Java) code versus 2900 lines of C++ code
• Only have to program Map and Reduce methods in Hadoop
  • The framework takes care of everything else
• C++ code needs to account for memory management by hand
  • For in-core and out-of-core capability
Ongoing & Future Work

• Hadoop performance tuning
  • Implementation of a custom range partitioner to obviate the sorting requirement for reduced outputs

• myHadoop - Personal Hadoop clusters on HPC resources
  • Accessible via PBS or SGE

• Implementation Techniques
  • MPI-based implementation for HPC resources
  • User Defined Functions (UDF) for relational databases
Conclusions

• A MapReduce implementation may be a viable alternative for DEM generation
  • Easier to implement
  • Better price/performance than a C++ implementation on an HPC resource
• May also be applicable for other types of LIDAR analysis
  • Vegetation Structural Analysis: Biomass Estimation
  • Local Geomorphic Metric Calculations: Profile Curvature, Slope
• Current MapReduce implementation doesn’t beat the in-memory HPC implementation
  • But memory limits may be reached in the near future for larger grid jobs, or for multiple concurrent jobs
• Serial bottlenecks may be the limiting factor for large parallel jobs
Acknowledgements

• This work is funded by the National Science Foundation’s Cluster Exploratory (CluE) program under award number 0844530, and the Earth Sciences Instrumentation and Facilities (EAR/IF) program & the Office of Cyberinfrastructure (OCI), under award numbers 0930731 & 0930643.
• Han Kim and Ramon Arrowsmith for designing the original C++ implementation
• OpenTopography and SDSC HPC teams
Questions?

• Feel free to get in touch with Sriram Krishnan at sriram@sdsc.edu