BERTUD: BUILDING FOOTPRINT EXTRACTION AND REGULARIZATION IN LIDAR DATASETS THROUGH UTILIZATION OF A DISTRIBUTED SYSTEM

Ivan Marc H. Escamos[1], Allen Roy C. Roberto[1], Joseph Anthony C. Hermocilla[1], Miyah D. Queliste[1], Edwin R. Abucay[1], Gillian Katherine L. Inciong[1]

[1]University of the Philippines Los Baños, Los Baños, Laguna, Philippines
Email: ivanhescamos@gmail.com

KEY WORDS: Distributed Computing, Multi-core Processing, Disaster Risk Management

ABSTRACT: The Philippines is situated inside the pacific ring of fire, which makes it vulnerable to natural disasters such as typhoons, tsunamis, volcanic eruptions and earthquakes. Disaster risk management plans must be developed and updated to keep up with the effects of climate change. Currently, such plans in the Philippines often lack updated data on the location of infrastructure and residential areas. Thus, building footprint extraction is an important task.

One of the tasks of our project, UPLB Phil-LiDAR 1, is to extract building footprints in the Laguna and MIMAROPA areas of the Philippines. Manual digitization of building footprints using GIS is tedious. With limited personnel and an extensive area to cover, devising an automated workflow is important.

Through the use of remote sensing techniques, building footprint extraction can be automated. Light Detection and Ranging (LiDAR) is one of the most powerful remote sensing technologies nowadays. We have developed an automated workflow for building footprint extraction and regularization from LiDAR datasets. One of the problems encountered with the created workflow is its running time. A 1km X 1km LiDAR tile of an urban area would take hours to finish.

In order to address this problem, we developed BERTUD, a distributed system to enable full utilization of the available computing resources of our project. Distributed systems are systems that make networked computers finish a big task.

The system was able to fully utilize the computing resources of our project by providing two layers of maximization: macro-level and micro-level. On the macro-level, BERTUD divides large areas for building extraction to multiple computers effectively speeding up the process. On the micro-level, the slave program maximizes the available resources of its host computer by utilizing multiple CPU cores, without interrupting its current user.

1. INTRODUCTION

The Philippines is located inside the Pacific Ring of Fire, which makes it vulnerable to natural disasters such as typhoons, tsunamis, volcanic eruptions, and earthquakes. Every year, an average of 20 typhoons enters the Philippine area of responsibility. In 2013, the country was ravaged by one of the strongest typhoons in history: Haiyan (locally known as Yolanda). Around 16 million people were affected and about 12.6 Billion USD worth of damage was caused by Yolanda (GFDRR, 2015).

The effects of climate change clearly cannot be ignored. Thus, disaster risk management plans must be kept up to date. In the Philippine context, disaster risk management plans are rife with the lack of updated spatial data on infrastructure and residential areas. In the wake of advancing technology, methods of hazard exposure and disaster mitigation are continuously developing. Technologies such as Geographic Information Systems (GIS) and Remote Sensing (RS) are useful for this endeavor. One problem in hazard exposure is to identify man-made structures susceptible to hazards. Thus, building footprint extraction is an important task.

UPLB Phil-LIDAR 1 (http://phil-lidar.uplb.edu.ph) is a product of the concluded UP Disaster Risk and Exposure Assessment for Mitigation (DREAM) Program (http://dream.upd.edu.ph). Its main objective is to develop 3D hazard maps for the Philippine river systems in the Laguna and MIMAROPA areas of the Philippines. Another objective of our project is to extract building footprints in the said areas. Building footprint extraction is usually done through manual

digitization using GIS, and this task is tedious. With limited personnel and an extensive area to cover, devising an automated workflow is of importance.

Light Detection and Ranging (LiDAR) is one of the most powerful remote sensing technologies nowadays. The LiDAR is an instrument similar to radar, instead of radio waves, it uses laser pulses (Campbell, 2011). These laser pulses measure variable distances to the surface, and when combined with GPS data, three-dimensional information about the surface can be gathered.

We have developed an automated workflow for building footprint extraction and regularization in LiDAR datasets. One of the problems encountered with the created workflow is its running time. A 1km X 1km LiDAR tile of an urban area would take approximately X hours to finish on a single high-end workstation.

In order to address this problem, we developed BERTUD, which stands for Building Footprint Extraction and Regularization Through Utilization of a Distributed System. According to Tanenbaum, a distributed system is "a collection of independent computers that appears to its users as a single coherent system (Tanenbaum, 2002)." We have observed that some CPU cores of the workstations used in our project are underutilized and can thus be used as part of a distributed system for building footprint extraction and regularization.

The general objective of this study is to develop BERTUD, a distributed system to allow the full utilization of the available computing resources of our project. The specific objectives are as follows:

a. To adapt our automated workflow for building footprint extraction into a distributed system
b. To fully utilize our project's computer laboratory by maximizing the resources of every computer and unifying these computers through a distributed system
c. To evaluate the performance of the distributed system

2. REVIEW OF RELATED LITERATURE

Through the years, several studies on the application of distributed computing and distributed systems in remote sensing have been done. In 2002, Petrie, et al., explored the "Beowulf Class Cluster Computing" strategy, where COTS (Commercial Off-the-Shelf) computers were clustered together to work as a computational team. They explored the software PiCEIS (Parallel Computational Environment for Imaging Science) and its possible application in remote sensing. The researchers ran a land cover classification experiment on a synthetic IKONOS image. Using a distributed system, they were able to achieve improvements in the running time of the experiment (Petrie, 2002).

Lee, et al. conducted a review on different distributed computing infrastructures (DCIs) for remote sensing. The Open Cloud Consortium (OCC) has an ongoing project called the Matsu Project which aims to provide cloud-based disaster assessments using comparisons of satellite images. Their infrastructure uses a Eucalyptus-based cloud with 300 cores, 80 TB of storage, and 10 Gbps network connections. It was initially used in flood prediction for Namibia. The European Space Agency started the Grid Processing on Demand (G-POD) project which aims to provide on-demand processing of Earth observation data. It was initially developed using a grid structure but it now uses the cloud approach. Users of this system can use various tools and algorithms to process datasets from ERS-1 and ERS-2 satellites, and the Envisat ASAR and MERIS sensors. Another DCI is the GEOSS, or the Global Earth Observation System of Systems, and it is managed by the Group on Earth Observations (GEO). It aims to launch an international, organized infrastructure for sharing Earth observation data products worldwide, with the intent to benefit disaster management, health, energy, climate, water, weather, ecosystems, agriculture, and biodiversity. In their 2009-2011 work plan, they targeted the development of the GEOSS Common Infrastructure (GCI). The GCI will enable organizations from different countries to register their datasets and services (Lee, 2011).

The applications of distributed systems in remote sensing are quite extensive. Distributed systems are really helpful in improving and optimizing remote sensing tasks. It is one of the logical steps to take in improving our automated building footprint extraction workflow.

3. METHODOLOGY

3.1 Building Footprint Extraction Workflow

3.1.1 Software and Technologies Used

ESRI's ArcMap was primarily used for the visualization of input and output datasets. LiDAR derivatives were generated using the LAStools software suite by rapidlasso (Isenburg, 2015). The main programming language used was Python, specifically Python version 2.7.3. Several third party libraries for Python were used in this study. The SciPy stack was used for additional scientific and mathematical functionalities (Jones, 2001). Digital image processing was done using the Scikit-image library (van der Walt, 2014).

3.1.2. Input preparation

The inputs for the building footprint extraction workflow are LiDAR point cloud data in the LAS (.las) open file format. These LAS files are from the DREAM Phil-LiDAR 1 program. Various derivatives are generated from the LAS data using the LAStools software package, specifically *laszip*, *lasground*, *lasclassify*, *lasgrid*, and *blast2dem*. These derivatives were further processed via digital image processing to generate the other inputs needed.

The raw LAS file was first compressed to the LAZ (.laz) open file format using *laszip*. The LAZ file format is a compressed version of the LAS file format. The compression greatly reduces the file size of the point cloud data which makes it faster to transfer data within the network. UTM projection was also done using *laszip*. To identify ground points in the data, *lasground* was used. After ground point classification, *lasclassify* was used to classify the point cloud into vegetation and building points. *lasgrid* was used on the classified point cloud to rasterize the number of returns and classification. The digital surface model (DSM) and the digital terrain model (DTM) rasters were generated using *blast2dem* on the classified point cloud. These rasters were generated as Tagged Image Files (TIFF) with a cell size of 0.5m. A Normalized Digital Surface Model (NDSM) was then generated by subtracting the DSM and the DTM. From the NDSM, the slope raster was calculated using the average maximum technique (Burrough, 1998).

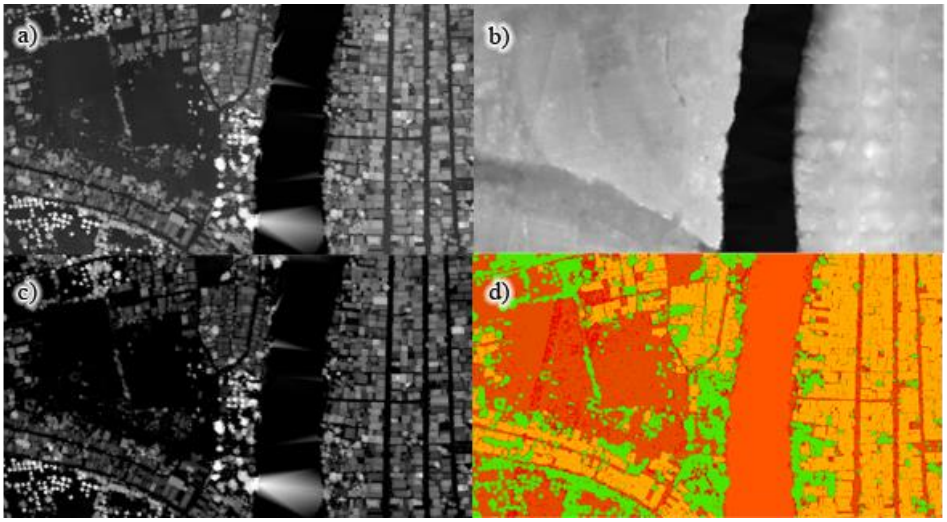

Figure 1. Inputs for Building Footprint Extraction Workflow. (a) DSM. (b) DTM. (c) NDSM. (d) Classified Raster

3.1.3 Initial Building Footprint Generation

The main basis for our building footprint generation workflow is the Watershed algorithm. This algorithm provides a segmentation of an image by treating it as a "topographic surface". This simulated surface is "flooded" from user-defined marker regions, with each marker region having a uniquely "colored" water. The water levels "rise", and these colored floods will inundate in the surface in a uniform rate. Dams are built to prevent different colored waters to merge, and they serve as the lines between the segmented regions (Audigier, 2004). There are two key parameters in this algorithm: the image where the watershed algorithm is applied, and the user-defined markers.

To generate the watershed base, vegetation pixels were removed from the NDSM by cross-referencing it with the classified raster. Morphological opening was performed to smoothen the base to allow the growth of regions. Markers were generated by using the slope raster to remove the areas on the NDSM with a slope greater than 30°. Once again, the vegetation pixels were removed. The watershed algorithm was then applied on the generated base using the generated markers. On some cases, there were artifacts on areas with low point density; this phenomenon is usually seen in river areas. These were removed by cross referencing the result of watershed to the number of returns raster. (Figure 2)

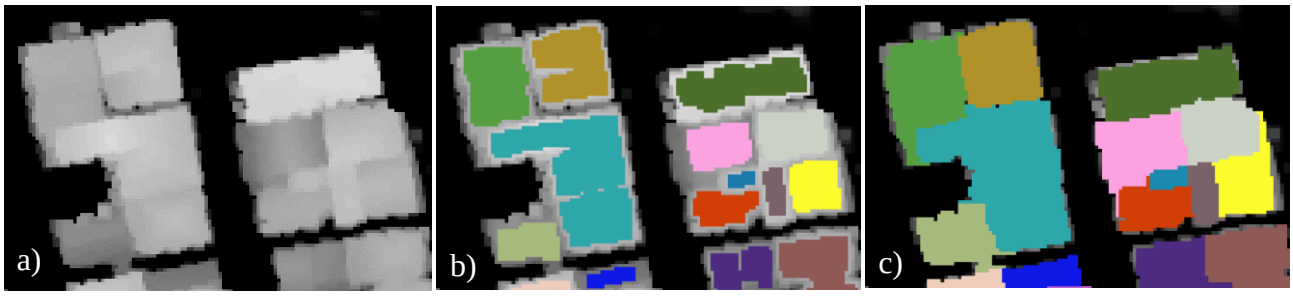

Figure 2. Initial Building Footprint Generation. (a) Watershed Base. (b) Markers. (c) Segmented Result

3.1.4 Building footprint regularization

The generated building footprints from the previous step are quite rough, thus the building footprints were regularized. Regularization refers to generating simpler polygons from complicated polygons while retaining its shape, effectively straightening edges and defining building corners. The method used for regularization was based on Wang's "Bayesian Approach to Building Footprint Extraction from Aerial LiDAR Data" (Wang, 2007). From the initial footprint (Figure 3a), building boundary points were first identified. These points were ordered using depth-first search neighbour tracing. Critical points were then identified by approximating line segments on the sets of ordered points (Figure 3b). These critical points will likely be the corners of our building footprint. The most probable building footprint was determined by maximizing its fitness using Bayesian probability optimization and simulated annealing techniques. This step encourages straight lines (180° angles), 90° and to a lesser extent 45° and 135° angles for the building corners (Figure 3c).

Further refinements were done by adjusting the footprint's area and orientation. These steps are important because the randomness aspect brought by the optimization tends to oversimplify the footprint (Figure 3c). The footprint's area was adjusted by either growing or shrinking the sides of the building polygon while the orientation was fixed by rotating the footprint at its center on either directions (Figure 3d). The algorithm then decides whether to accept or reject the newly adjusted footprint by using a simple fitness function whereas the number of the points of the original footprint that lies within the adjusted footprint were counted. If there are more points that lie inside the adjusted footprint, the solution is accepted. These steps were repeated until there can be no more changes on the footprint.

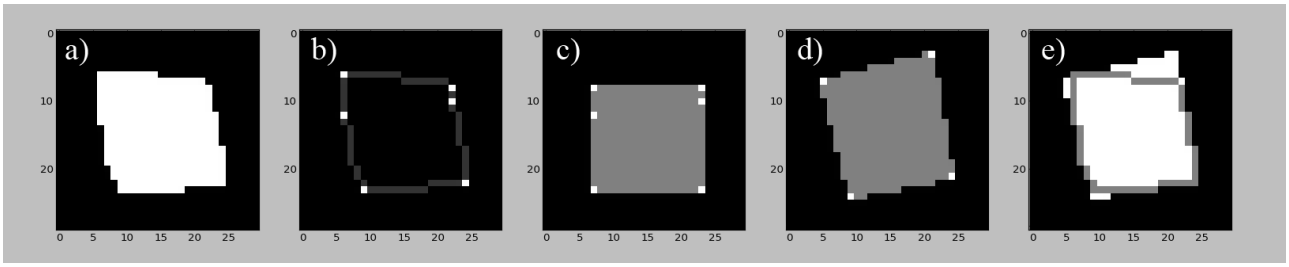

Figure 3. Example for building footprint regularization. (a) Initial rough footprint of the building. (b) Building boundary points shown in gray and critical points in white. (c) Regularized building footprint using Bayesian probability optimization and simulated annealing. (d) Adjusted building footprint by growing / shrinking its sides and rotating it at its center to find the maximum fitness. (e) Final building footprint colored in white along with the initial footprint in gray.

Our method for building footprint regularization is computationally expensive. This process takes a long time to finish. The running time is directly proportional to the area of the building footprint being regularized, the larger the building footprint, the longer it takes to be regularized. We addressed this problem in section 3.2.3

3.2 Development of the Distributed System

3.2.1 System Architecture

We treated the problem of optimizing our building footprint extraction workflow in our project laboratory as an "embarrassingly parallel" problem. Problems are "embarrassingly parallel" when they can be easily separated into a number of parallel tasks (Herlihy, 2012). The idea was to make multiple computers run the same building footprint extraction workflow at the same time.

The distributed system was patterned after the *master/slave paradigm*. It is a fundamental and a commonly used approach in developing distributed systems. In this paradigm, a single *maste*r process distributes work to a group of identical slave *processes* (Shao, 2000).

In our case, the master allows a user to send LAZ files to other computers, the slaves, for processing. We created slave processes that wait for the master process to send LAZ files. As LAZ files are sent, the slave processes fetch a single LAZ file and apply the building footprint extraction workflow.

3.2.2 Software and Technologies Used

Python was also used as the main programming language. Python Remote Objects or Pyro was the library used for the implementation of the distributed system. It is an advanced and powerful Distributed Object Technology system (de Jong, 2016). Flask was used to develop the backend of the web application that served as the user interface of the distributed system (Ronacher, 2016). HTML, CSS, and Javascript were used for the development of the frontend part.

3.2.3 The Master Program

A program's data and functionalities become exposed to other programs within the network when registered to a unique Pyro name server. For our case, we registered the master program, allowing the master program to wait and handle incoming requests from the slave program/s.

Users are able to and only have to interact with the master program through a web application developed using Flask, HTML, CSS, and Javascript. In this web application, users are able to add or remove units of work/LAZ files in the queue of the master program, which in turn will be accessed by the slave program/s. It displays the list of work items that were successfully and unsuccessfully processed. Work items are labelled successful if they finished processing with no errors. It also shows the updates coming from the slaves. These updates show the current status (Working, Non-Working, Busy), and CPU & RAM usage of the slave computer/s.

3.2.4 The Slave Program

In the case of the slave program, what it only needs to find is the unique Pyro Name Server where the master program is registered. After successful connection, the slave program automatically checks for work items inside the master's queue. If the slave program finds pending work, the corresponding input file will then be sent remotely through the network from the computer running the master program to the slave computer that requests it.

After receiving the input file, the slave program will first compute the number of recommended CPU cores that can be used. This is because the slave program provides another layer of optimization to the building footprint extraction workflow by allowing the use of multiple CPU cores.

As mentioned earlier, building footprint regularization takes a long time to finish. Once again, we treated this problem as *embarrassingly parallel*. We solved this by distributing the processing of these building footprints to different CPU cores. This allowed the regularization of all the building footprints in the LAZ file finish faster since there are more CPU cores that work together (See section 4.2.1).

The number of recommended CPU cores depends on the usage of the other programs running in the background. This recommended number of cores will be used in running the algorithm for building footprint extraction and regularization on the input file.

If the process was successful, the output files will be sent back to the master program and it will be categorized as *finished work*. However, if the process fails, the work will be labelled *unsuccessful*. Another feature of the slave program is that it can update its status to the master program. *Non-working* status indicates that the slave is currently looking for work while *working* status shows that the slave is processing. Lastly, the *busy* status signifies that the computer is currently preoccupied with other tasks (above 90% CPU usage). If the slave is not running, the master defaults into the *disconnected* status. Other than its status, the slave also periodically updates its CPU and RAM usage for it to reflect in the master's interface.

4. RESULTS AND DISCUSSION

4.1 The Developed System: BERTUD

Figure 4 shows the master web application. On the left side is the "Queue Management Panel." It has the "Add Files" tab, where users can specify the input folder of the LAZ files, and the output folder where the outputs will be stored. It also has the queue panel, where users can see the current queue, and what LAZ file each slave computer processes. On the bottom of the queue panel, we can see the "View Logs" button where users can see the processing logs of the current session, whether a LAZ file was labeled *successful* or *unsuccessful*. On the right side, we can see the "Slaves" panel where we can see the status of the slave, whether it's *working, non-working, busy, or disconnected.* We can also see the slave's CPU and RAM usage. Currently, the BERTUD system's web application can only handle six slaves.

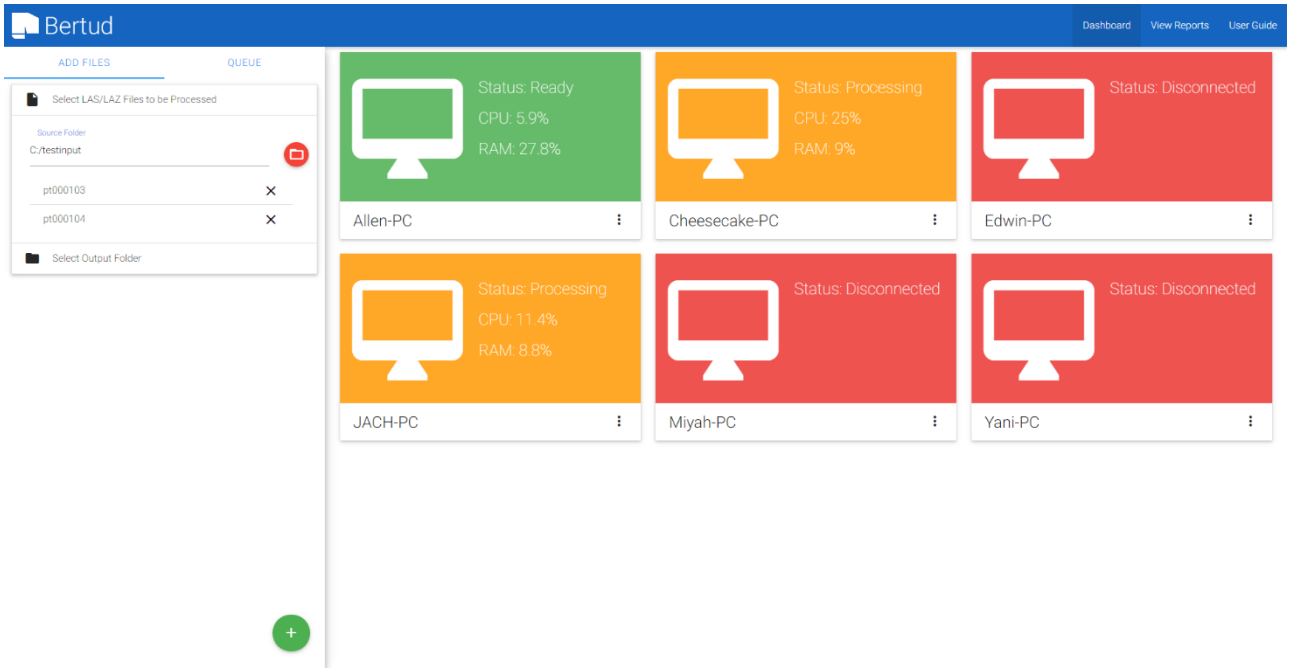

Figure 4. The BERTUD Master Web Application

Unlike the master program that has a web application that serves as a user interface, the slave program is a console application. Figure 5 shows the slave program running in the command line. It displays status messages on what it is currently doing. The slave program also runs quietly in the background due to its calculation of usable CPU cores, effectively maximizing the power of the computer without interrupting its current user. It defaults to six CPU cores on an eight-core system to provide leeway for the slave computer's users, but it can be set to eight when the slave computer is not used by anyone.

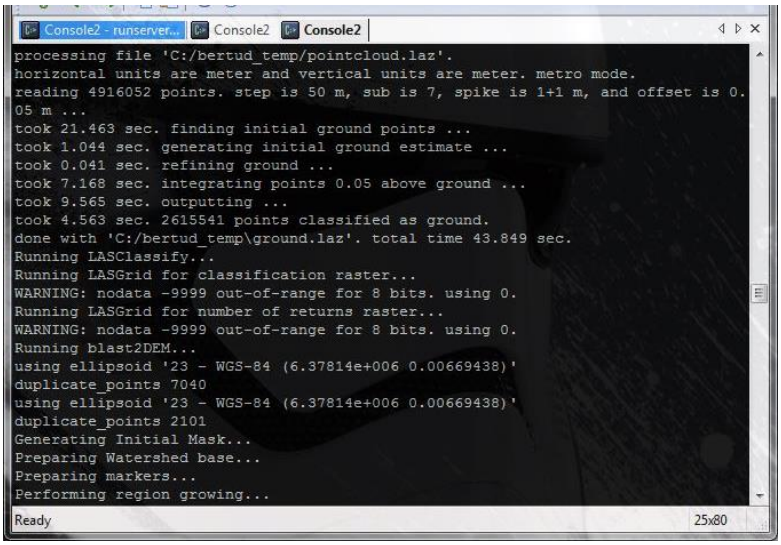

Figure 5. The BERTUD Slave Console Application

4.2 Performance

We aimed for two layers of laboratory optimization: *micro-level* and *macro-leve*l. Micro-level optimization is the full CPU utilization of each computer, since we observed that our computers were underutilized. Macro-level optimization is the simultaneous usage of multiple computers in our laboratory which also includes the efficient distribution of work. We performed a series of experiments to evaluate the said layers of optimization. For these experiments, we used HP® Z230 tower workstations. These workstations are equipped with Intel® Xeon CPUs with eight 3.60GHz cores, 32 GB of RAM, 256GB solid state drive storage, and 1TB of hard drive storage.

4.2.1 Evaluation of Micro-level Optimization

In this experiment, we made the slave program process three 1km X 1km LAZ files: a non-urban, semi-urban, and urban area. The non-urban area consists of approximately 200 buildings. The semi-urban area comprises approximately 300 buildings with large warehouse type buildings. The urban area consists of approximately 2200 buildings. Each LAZ file was processed using one, two, four, and eight CPU cores. Figures 6, 7, and 8 show the running times for these configurations.

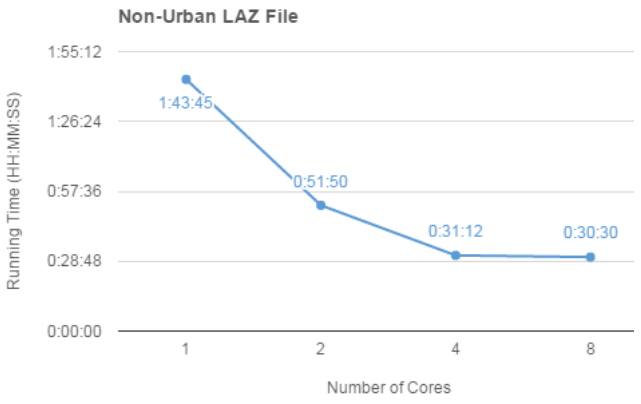

Figure 6. Non-Urban LAZ File

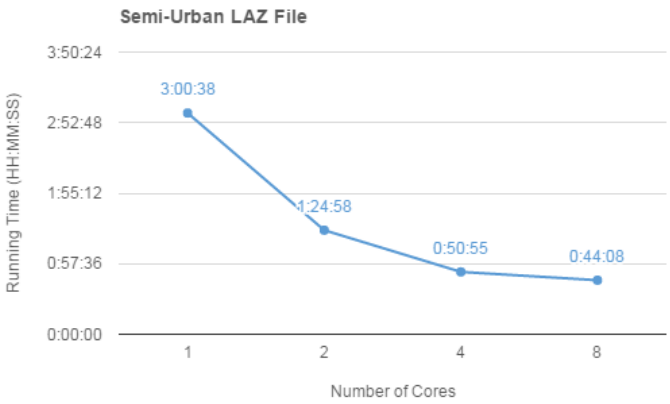

Figure 7. Semi-Urban LAZ File

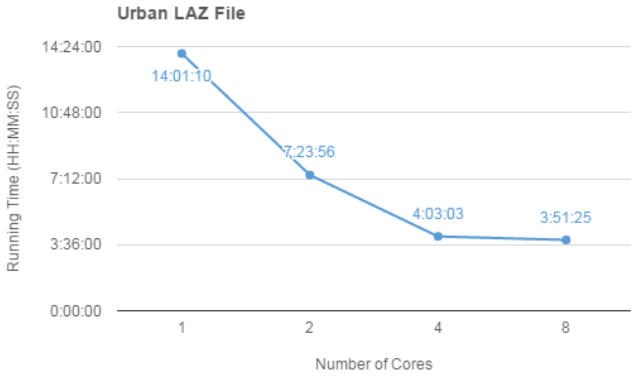

Figure 8. Urban LAZ File

The graphs for all three types of LAZ files (non-urban, semi-urban and urban areas) show a similar trend. The improvement in running time as the number of cores goes up diminishes as it reaches 4 cores. This is possibly caused by the inefficient distribution of work in the CPU cores. The number of building footprints to be regularized are evenly distributed among the CPU cores without considering the size and shape of the building footprints. This leads to idling of some CPU cores since they finish their workload first before the other CPU cores. It is also apparent that the more urban the area is, the more evident the speedup in running time.

4.2.2 Evaluation of Macro-level Optimization

We wanted to explore the effect of the number of slaves in finishing a set of LAZ files while exploring the effect of different degrees of urbanness in the set of LAZ files. Thus we created two sets of LAZ files, the *easy set* and the *mixed set*. The *easy set* is composed of ten copies of the non-urban LAZ file used in the previous experiment. The *mixed set* is comprised of ten unique LAZ files with varying degrees of urbanness. We also wanted to study two use cases for BERTUD: where slave computers use the default maximum of six CPU cores with the usual workload of its users, and use all of their eight CPU cores.

Experiments for Macro-level optimization are shown below. Experiments A (Figure 9) and B (Figure 10) fully utilize the processing power of the slave computer by using all eight available cores while experiments C (Figure 11) and D (Figure 12) only use six cores providing leeway for other user processes. On the other hand, experiments A and C both use the *easy set* of input LAZ files while the *mixed set* was used in B and D. Also, these sets of input LAZ files for all experiments were processed separately using a different number of slaves.

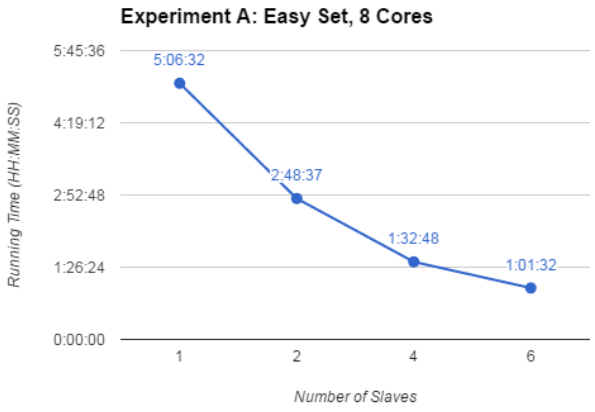

Figure 9. Experiment A: Easy Set, 8 Cores

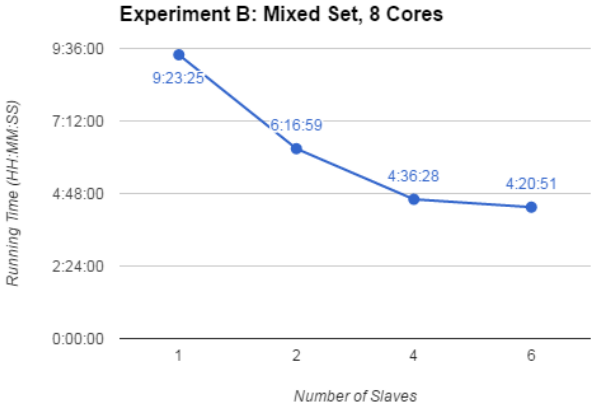

Figure 10. Experiment B: Mixed Set, 8 Cores

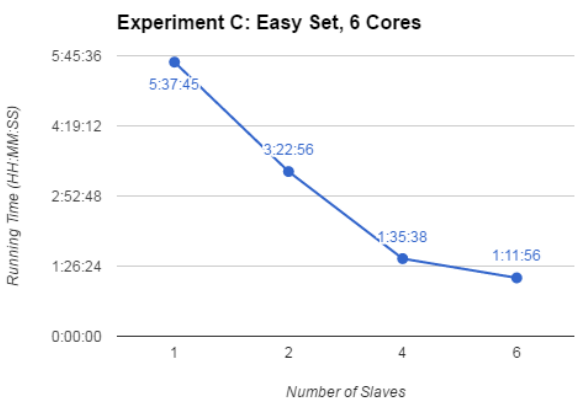

Figure 11. Experiment C: Easy Set, 6 Cores

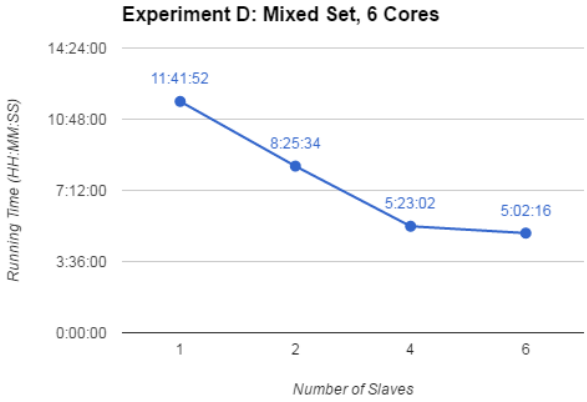

Figure 12. Experiment D: Mixed Set, 6 Cores

Based on the results above, all experiments show speedups in running time as the number of slave computers increase. However, these speedups also slowly diminishes but not as evident compared to micro-level optimization. These diminishing results were caused by the scenario where there is no more pending work available for other computers while some slaves are still processing their respective LAZ files. This causes the idling of some computers.

Comparing the experiments (A and C) that both use the *easy set* of LAZ files, we observed that the difference in their running time is not that much significant even though we used different numbers of cores for the two. However, this is not the case for experiments B and D. Even though they both used the same *mixed set* of LAZ files, the difference in their running times were more evident. From these observations, we deduced that the main contributing factor when it comes to the running time is the degree of urbanness of the input LAZ file.

5. CONCLUSION AND FUTURE WORK

In this study, we developed the BERTUD system, which is a distributed system that generates building footprints from LiDAR datasets. We adapted our automated building footprint extraction workflow to a distributed system since it took a long time to run on just a single computer. We designed BERTUD after the master/slave paradigm and we created our master and slave programs. The master program allows users to send LAZ files to the slave programs through the network. As LAZ files become available, slaves fetch a single LAZ file and apply the workflow. BERTUD provided two layers of optimization: micro-level, and macro-level. Micro-level refers to the full utilization of the slave computer by utilizing multi-core processing. Macro-level refers to the usage of multiple slave computers and the efficient distribution of work.

We ran experiments to evaluate the micro-level and macro-level optimizations. In the micro-level, we found out increasing the number of cores improves the running time, but there are diminishing returns as we use more cores. This is due to inefficient load balancing between the CPU cores. In the macro-level, we observed that increasing the number of slaves significantly improves the running time to finish a set of LAZ files. However, we observed that there are diminishing returns as we increased the number of slaves due to slaves being idle as the list of files to be processed runs out. We also found out that the problem of diminishing returns in the micro-level greatly affects the macro-level. Overall, we observed that the running time to finish a set of LAZ files is greatly affected by the degrees of urbanness of these files.

Currently, the BERTUD system is used in our project, UPLB Phil-LiDAR 1. We use it to extract building footprints for the different river basins in the MIMAROPA and Laguna regions of the Philippines. We have generated building footprints for at least nine river basins using BERTUD. Outputs from our building footprint extraction workflow are still not perfect, there are misclassifications and building outlines to be adjusted. Manual editing is still done to further refine the results of BERTUD. Despite the need for manual editing, BERTUD is still helpful in the building footprint extraction process of our project.

In the future, we intend to improve the algorithm for building footprint extraction to lessen the need for manual editing. We also intend to improve the load balancing in micro-level optimization which will yield even better running times. We could explore the realm of GPU and high-performance computing using NVIDIA's CUDA API to further improve the running time of the workflow. We also aim to improve the developed master web application to handle more slave computers. Assessment of the impact on the productivity of the personnel when executing their regular workflows while running BERTUD on their workstations can also be done. Lastly, the BERTUD system can be tailored to be used in other research areas and workflows, even outside the geospatial and remote sensing realm, as long as the problem is *embarrassingly parallel.*

Acknowledgement

The authors would like to acknowledge the Department of Science and Technology-Philippine Council for Industry, Energy and Emerging Technology Research and Development (DOST-PCIEERD) for financial support of the UPLB Phil-LiDAR 1 project. We also acknowledge DREAM/Phil-LiDAR 1 UP Diliman for the data used in the study.

References

Audigier, R., Lotufo, R., Falcao, A., 2004. On On integrating iterative segmentation by watershed with tridimensional visualization of MRIs. Brazilian Symposium of Computer Graphic and Image Processing. p. 130-137

Burrough, P., and McDonell, R., 1998. Principles of Geographical Information Systems. Oxford University Press, New York, p. 190

Campbell, J.B., Wynne, R.H., 2011. Introduction to Remote Sensing. The Guilford Press, New York, p. 243
de Jong, I., 2016. Pyro - Python Remote Objects - 4.47, Retrieved September 9, 2016, from http://pythonhosted.org/Pyro4/

GFDRR, UNDP, EU, 2015. Typhoon Yolanda Ongoing Recovery. Retrieved September 8, 2016, from https://www.gfdrr.org/sites/gfdrr/files/New%20Folder/Philippines%20August%202014.pdf

Herlihy, M., Shavit, N., 2012. The Art of Multiprocessor Programming, Revised Reprint. Elsevier. p. 14

Lee, C., Gasster, S., Plaza, A., et al., 2011. Recent Developments in High Performance Computing for Remote Sensing: A Review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. 4 (3), pp. 508-527.

NOAA, 2015. What is LiDAR?, Retrieved September 8, 2016, from http://oceanservice.noaa.gov/facts/lidar.html

Petrie, G., Dippold, C., Fann, G., et al., 2002. Distributed computing approach for remote sensing data. Proc. 34th Symp. Interface, pp. 477–489

Ronacher. A., 2016. Flask (A Python Microframework), Retrieved September 9, 2016, from http://flask.pocoo.org/

Shao, G., Berman, F., Wolski, R., 2000. Master/slave computing on the Grid. Heterogeneous Computing Workshop, 2000 Proceedings.

Tanenbaum, A., 2002. Distributed Systems. Prentice Hall, Inc., New Jersey, pp.2

Wang, O., Lodha, S., Helmbold, D., 2007. A Bayesian approach to building footprint extraction from aerial LIDAR data. Proceedings - Third International Symposium on 3D Data Processing, Visualization, and Transmission, p. 192-199