PARALLELIZING THE EXECUTION OF ARCGIS GEOPROCESSING TOOLS TO IMPROVE THE PERFORMANCE OF COMPUTING AND PROCESSING MASSIVE GEOGRAPHIC DATASETS: A HEURISTIC RESEARCH ON BIG DATA PROCESSING IN THE PLANNING FIELD

By

CHANGJIE CHEN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ARTS IN URBAN AND REGIONAL PLANNING

UNIVERSITY OF FLORIDA

2014

To my Dad and Mom

ACKNOWLEDGMENTS

I thank the faculty in the Department of Urban and Regional Planning for their continuous guidance and support during the studies of my master program. I would like to express my most sincere gratitude to my committee chair Prof. Paul Zwick who introduces me to the area of spatial analysis in the planning field and leads me to research this particular topic of parallel processing in ArcGIS environment. I thank my committee co-chair Prof. Ilir Bejleri for initiating me in planning thoughts and theories when I was beginning my master program, and his guidance in strengthening this thesis.

I need to especially thank my parents for all kinds of supports they give to me, without which I cannot go this far. I want to thank my friends for their company during both tough and happy times.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CEM	Conflict Evaluation Matrix
CPU	Central Processing Unit
COM	Component Object Model
CSV	Comma Separated Variable
CTPP	Component Object Model
DOT	Department of Transportation
ESRI	Environmental Systems Research Institute
FGDL	Florida Geographic Data Library
FLOPS	Floating Point Operations Per Second
FOSS	Free and Open Source Software
GIS	Geographic Information System
GIL	Global Interpreter Lock
HDD	Hard Disk Drive
I/O	Input/Output
OD	Origin-Destination
OOP	Object-oriented Programming
LED	Local Employment Dynamics
LEHD	Longitudinal Employer-Household Dynamics
LODES	LEHD Original-Destination Employment Statistics
LUCIS	Land-Use Conflicts Identification Strategy
PSF	Python Software Foundation
RAC	Residence Area Characteristic
RAM	Random-Access Memory

SSD	Solid State Drive
VBA	Visual Basic for Application
WAC	Workplace Area Characteristic

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master in Arts of Urban and Regional Planning

PARALLELIZING THE EXECUTION OF ARCGIS GEOPROCESSING TOOLS TO IMPROVE THE PERFORMANCE OF COMPUTING AND PROCESSING MASSIVE GEOGRAPHIC DATASETS: A HEURISTIC RESEARCH ON BIG DATA PROCESSING IN THE PLANNING FIELD

By

Changjie Chen

December 2014

Chair: Paul. D. Zwick
Cochair: Ilir Bejleri
Major: Urban and Regional Planning

The term big data has become pervasive in recent years. Successful applications can be seen in every field from science to technology, and from public domains to private sectors. In the planning field, this arriving era is moving the data that we used towards a more real-time and particularized direction. Institutions such as the United States Census Bureau and DOT (Department of Transportation), they are releasing new data in a shorter cycle and with richer detail. Meanwhile, a growing number of devices are installed around the world, which are automatically and ceaselessly generating new geographic data.

In face of the opportunities coming with the explosion of data, planners should find out their way to catch them. As a widely-used tool in the planning realm, ArcGIS possesses the potential to accomplish this task with its powerful functions in analyzing and processing geographic data. Through parallel programming ArcGIS Geoprocessing Tools, this research serves as an icebreaking effort in leveraging big data processing in the planning field. The positive results indicated a huge potential in performing the

parallelism on a supercomputer, such as HiPerGator[1], which is the next level of the research. Being able to process and analyze the big data could induce a shift from a traditional long-term strategic planning to a more flexible and adjustable short-term tactical planning, which would definitely change the way we do planning today.

[1] HiPerGator: A supercomputer owns by University of Florida, which has 16,384 cores and 65,536 GB memory, with the peak speed of 119.3 TFLOPS (Tera- floating point operations per second; 1 TFLOPS = 10^{12} FLOPS). (Strohmaier, 2013)

CHAPTER 1
INTRODUCTION

Research Purpose

The first decade of 21st century has witnessed a great leap in information technology advances. By the meantime, the explosion of billions of records of data lead us to an era of the so-called Big Data. There is no such a limit of size in defining what big data is, however, a commonly accepted statement is any datasets whose size beyond the processing ability of typical database software. With the panoramic perspective provided by big data, people are able to look at the whole spectrum of a problem and can be as precise as possible. Successful applications of big data can be found in almost every domain, such as business, industrial production, social science and even criminal prevention.

In fact, what really lighted the fuse of big data blast is the collection of data pertaining to human activities. Another notable fact is that, today, most human activities happen within cities. Therefore, the successfulness of being able to analyze big data is undoubtedly meaningful to manage and regulate human activities in urban areas in return, which is also the ultimate goal of urban planning.

In planning area, the term of big data seems to be not as fresh as it to other fields. Planners are familiar with working with and making decisions (suggestions in reality) based on a large amount of data and information. Rather than the scale of data, the more attractive aspect to planning professionals about big data is the temporal connotation it contains. Since human activities are the results of people's judgment of their surrounding environment and adaptive responses to its change, human activities can be totally varied in time. In face of this great variation, traditional planning, which

usually has a range of 5 to10 or 10 to 20 years, seems to be strengthless. An emerging trend that has been spurred by big data in planning field is the shifting of emphasis from longer term strategic planning to short-term thinking about how cities function and can be managed (Batty, 2013).

In 2010, more than 30 million networked sensor nodes are present in the transportation, automotive, industrial, utilities, and retail sectors around the world. The number of these sensors is increasing at a rate of more than 30 percent a year (Manyika, et al., 2011). These devices are generating thousands of millions of records of highly detailed data day after day. It should be noted that, in order to depict the real world, there is no “too detailed” data in big data realm. Seemingly irrelevant data can be linked together to generate meaningful information. With the datasets advancing in a direction towards real time and particularization, in terms of making a better plan, both researchers and practitioners must be able to take advantage of the opportunities generated by this increasing scale of data availability.

Modern computer hardware technology makes the computation speed keep increasing. However, the traditional way of data processing in the planning field has no way to leverage that capability. Although ArcGIS software, as a commonly used tool in the planning field, has powerful functions in processing geographic data, it cannot satisfy planners’ needs in quickly processing large amount of data. Since ArcGIS functions can only employ a single processor at any point of time, planners are prohibited to utilize the power of multiple processors.

Through combing Python’s multiprocessing package and the ArcPy site package, the research aims to find out the possible way to improve the performance of

processing geographic data Chapter 2 reviews the relating literatures. In chapter 3, the author discusses the methods to realize the parallelization in the ArcGIS environment. The result are explained and discussed in chapter 4 and 5. Conclusions are made in chapter 6 which also pointed out the imperfections of this research. With chapter 7, the direction of future research is discussed.

This research is all about seeking the possibilities of utilizing big data in the planning field. The chapters included in this paper concentrate on articulating the theories and approaches which supported those possibilities. Although the research included a huge amount of Python coding to realize parallelism, this paper is not a "Python Programming for ArcGIS" tutorial. However, since parallel programming is difficult, especially when working with ArcPy module to process geographic data, it has significant meaning to discuss about it. Therefore, the author included four parallel programing source code in the appendix. Interested readers are welcome to contact the author to explore more possibilities together.

Research Questions

In order to get better performance in processing geographic datasets, this research seeks the possibilities of incorporating ArcPy functions into Python multiprocessing package to leverage the computational power of multiple processors. The research will answer the following questions:

1) Is it possible to utilize multiple processors to run more than one ArcGIS tasks at the same time?
2) How to take the full capability of multiple processors of CPU to improve the performance of geographic data processing?

3) How much faster it could be comparing parallelized processing with the conventional single-core processing? and

4) In what scenarios, should we do parallel processing to increase the speed of processing geographic data?

Contributions of This Research

This research explores the possibility of combining Python's capability in parallel processing and the capability of ArcGIS in processing geographic dataset, so that, to improve the performance of processing large amount of datasets. Planners can directly employ the methodology developed in the research to process planning related data in a much faster speed than the way it used to be. As the result claims that it is possible to take advantage of the computational power offered by multiple cores in executing functions in the ArcPy site package, the research create a way for servers with hundreds of cores to be able to perform geographic data processing. Therefore, the research could also be seen as an icebreaking effort in leveraging big data analysis in the planning field.

CHAPTER 2
LITERATURE REVIEW

Python and Scripting for ArcGIS

Python is a popular open source programming language used for both standalone programs and scripting applications in a wide variety of domains. Several notable features of Python contribute to its prevalence and make it an ideal programming language for working with ArcGIS.

It's simple and easy to learn. Python is well known as a relatively easy programming language, especially for starters. Thus, ArcGIS users could focus more on tackling the real GIS (Geographic Information System) problems, instead of dealing with the language difficulties.

It's free and open source. Python is free and open source software (FOSS). This feature makes Python accessible and free to distribute. Python published very detailed documentations of every syntax of the language. Also, there are a great amount of sources to learn about it on the internet. The free distribution feature makes a third-party, like Esri (Environmental Systems Research Institution), could use it, build their own site package, and distribute the ArcPy to everyone.

It's object-oriented. Object-oriented Programming (OOP) allows users to write programming scripts to model objects and how they interacts with other data. Each individual ArcGIS's geoprocessing task, by its nature, is an OOP code. Essentially, an object is set as input; run through a particular method; and an object is generated as an output. Thus, Python qualifies a perfect programming language that could work with ArcGIS in this respect.

It’s interpreted. Many programming languages require that a program be converted from the source language, such as C++ or Visual Basic, into binary code that the computer can understand. This requires a compiler with various options. Python is an interpreted language, which means it does not need compilation to binary code before it can be run. You simply run the program directly from the source code, which makes Python easier to work with and much more portable than other programming language. (Zandbergen, 2013)

Python’s 2.X/3.X story is worth to tell to any user who wants to learn Python. PSF (Python Software Foundation) released version 2.6 in October 2008. However, only two months later, they released an entire new generation of Python 3.0 in December. Python is now dual-version world, with many users running both 2.X and 3.X according to their software goals and dependencies (Lutz, 2013). Different from most other software, Python is not backward compatible, which means 3.X cannot substitute or cover the whole range of capability provided by 2.X. Python 3.X is seen as the future. Although the latest version of 2.X, Python 2.7, is still supported by Python developer, it would be the last 2.X. Nevertheless, at present, the new generation cannot simply replace its predecessor, with the fact that 2.X is still downloaded more often than 3.X.

Python is regarded as both a programming language and scripting language. A programming language has the ability to develop advanced and sophisticated software and applications. It’s also capable of dealing with raw resources of computer to build an operating system. On the other hand, scripting language is the “gear” that links each individual parts and makes them work together. In other words, scripting languages use

higher-level built-in functions to perform a related new application. Therefore, scripting always works as a part of a bigger programming task.

ArcGIS IS COM (Component Object Model) compliant, which is the most widely used software architecture. This makes scripting languages can access to all the tools available in ArcGIS to automate tasks and workflows. Although this automation can be achieved by programming, scripting always requires less effort on coding itself, but allowing programmers focus on the real problem.

With the attractive feature of both a programming and scripting language, Python quickly displaced the former widely used language – VBA (Visual Basic for Application) – in ArcGIS software. The reason is largely because Python has the advantage of ease of use of a scripting language, as well as the programming capability of a highly-structured developer language. ArcGIS 10 has seen further integration of Python within the ArcGIS Interface, and Esri has officially embraced Python as the preferred scripting tool for working with ArcGIS (Zandbergen, 2013).

Like Microsoft built the “Windows” operating system based on C++, an advanced OOP language, Esri also relies on C++ to develop their software with the key components named ArcObjects. Programmers could use C++ to call these objects as well as create their own objects to realize a particular goal of a project. However, it could be even easier and efficient to utilize scripting language to connect the existing functions in a new way to fulfill the same task as well. To most GIS professionals who are not in the field of computer science, planners as an example, the latter way is more realizable and it is much more pervasive indeed. Esri makes scripting much more productive in ArcGIS environment by introducing ArcPy.

What Is ArcPy

ArcPy (often referred to as the ArcPy site package) provides Python access for all geoprocessing tools, including extensions, as well as a wide variety of useful functions and classes for working with and interrogating GIS data (Essential ArcPy vocabulary, 2014). It is included with a typical installation of ArcGIS software. Besides the fundamental tools stored in ArcPy module, the site package also includes four modules to offer the accessibility to all the functions that you have with the conventional ArcMap application. They are:

- Data Access Module (arcpy.da),
- Mapping Module (arcpy.mapping),
- ArcGIS Spatial Analyst Extension Module (arcpy.sa),
- ArcGIS Network Analyst Extension Module (arcpy.na).

ArcGIS 10.2.2, the latest version of the software, relies upon Python 2.7.5, which means the successfulness of working with ArcPy site package depends on a proper installation of a corresponding version of Python. Since the 2.X version of Python would be unsupported in a foreseeable future, ArcGIS would eventually adopt the 3.X as an alternative. By then, users may encounter a considerable changes, however, the new capability of 3.X, on the beneficial side, could also create new possibilities and be a great experience to users working with ArcGIS.

Generally speaking, ArcPy is organized in tools, functions, classes and modules. The most absorbing benefit of using ArcPy is being able to access and work with the comprehensive toolset that developed by GIS programmers for the sake of automating the workflow of geographic data analysis.

ArcPy is a great invention that makes professionals with less knowledge of programming programmers. It lifts users up to a certain level, on which you do not need to know how a particular function coded, but only need to know what result it can provide. By calling different tools in ArcPy and combine them in different ways, users are able to build very sophisticated and customized scripts to reach their objectives.

Parallel Computing

Parallel Computing is a terminology used in the field of Computer Science, which defines a computing approach evolved from the traditional *Serial Computing*. Serial computing utilizes one CPU (Central Processing Unit) of one computer; breaks a problem into a discrete series of instructions; executes only one instruction at a given point in time. Parallel computation, on the other hand, takes advantage of multiple CPUs to improve the computing performance.

While the progress in hardware technology has significantly increased the capability that we have with computer, it triggered higher expectations to hardware itself. However, as the computational requirements are continually raised up, we face some limitations, for example overheating, which could not be handled. As the silicon-based processor chip is reaching its physical limits in processing speed, chip makers launched a major shift from inventing faster single-processor chip to build chips with connected multiple processors working in coordination with each other.

Modern chips consist of multiple microprocessors (also called cores), buses, and cache memory on the same chip. Server processors such as *Intel*'s *Single Chip Cloud Computer* demonstrate that it's possible to integrate 48 general-purpose processors on just 567 mm^2 (Pankratius, Schutle, & Keutzer, 2011). As of this writing, oct-core (eight

multiprocessors) chips are widely used on desktop, and this number is likely to increase.

With the ability offered by multi-processor CPU, a big task could be divided into smaller subtasks which can be executed simultaneously, or many tasks can be executed at the same time by individual cores. The course change of hardware development also causes the according adaptions to software. Programming and scripting languages, such as Java and C++, are able to take advantage of multiple cores through parallelism. Python is one of them.

Key Concepts in Parallel Processing

Thread vs. Process

Generally speaking, there are two ways of doing computation simultaneously. Multithreading, on the one hand, run multiple tasks by using a single processor. When one thread is utilizing the processor, the other threads are waiting. The threads quickly switch from one another to keep the single processor being used. It works like an internet browser, with which you can have multiple web pages open, but only one of them is currently viewed. Multithreading is a perfect solution in dealing with I/O bound tasks. However, it cannot really use multiple processors of CPU.

On the other hand, multiprocessing utilizes multiple processors of CPU at the same time. It is a new package coming with the release of Python 2.6. It uses the same API (Application Programming Interface) as multithreading. But, it work around with the GIL by copying multiple interpreters to multiple processors. Multiprocessing allows programmers to really take advantage of the computational power of multiple processors on CPU, and it work very well to solve CPU bound tasks.

Other Terminologies

To fully understand the theory of parallel processing, some other concepts have to be clarified.

- **Concurrency**, when applied to application/program logic, is the simultaneous execution of tasks (Noller, 2009)

- **Compiler vs. Interpreter.** Programming languages can be generalized categorized by using compiler or using interpreter. They are both for translating human-readable programming language to computer-understandable binary language. The difference between them is compiler translate the whole script first before running the code. So that, if there was a grammar error, the code would not run. The example of compiled language is VBA (Visual Basic for Application). Interpreter, on the other hand, translating the language while the code is running. As a result, even if there was a grammar error in coding, the script would still run until it meet the error line. Python is an interpreted programming language, the most commonly used interpreter of Python is CPython which is written in the C language. Examples of some other interpreters for python are Jython, IronPython, RubyPython, and PyPy.

- **GIL** (Global Interpreter Lock) is a mutex that prevents multiple native threads from executing Python bytecodes at once. This lock is necessary mainly because CPython's memory management is not thread-safe. (Athanasias, 2014)

- **I/O bound** is short for Input/Output bound, which described a situation that the computing speed is restricted by the reading and writing speed from and to a hard drive.

- **CPU bound** is the opposite of I/O bound, which which described a situation that the computing time is restricted by the computational power of CPU.

- **Process and Pool** are both objects in the multiprocessing package of Python. They have similar functionalities in utilizing multiple processors. One little difference might be that Process is able to work with Queue object. Combining Process with Queue objects can make parallel programming safer.

CHAPTER 3
RESEARCH METHODOLOGY

Theoretical Basis

The application of ArcGIS software in the planning field has been a long story. However, with the scale of data availability continually increasing, the conventional way of using ArcGIS appears to be less powerful in satisfying the need of processing big data. This situation is going to get worse, as planning data is moving towards a direction of real-time and particularized. In order to be productive, being able to process big data is a desire to professionals in planning field. A promising solution is parallel processing.

PSF (Python Software Foundation) introduced the new *multiprocessing* package, when they released the Python 2.6 version. It provides the objects and methods that support users to utilize multiple processors on one computer. Different from the previous *thread* which mainly focused on solve I/O bound tasks, the new package work around the GIL and allow multiple processes to run simultaneously, so that machine with multiple cores can really do computation much faster in dealing with CPU bound tasks.

The *ArcPy* site package provided by Esri offers the programming accessibility to each individual tool that can be found in the ArcMap application. Since ArcPy was written in Python language, incorporating the ArcPy objects and functions into Python programming is possible. Programming Python together with the ArcPy module can create more possibilities than only use either one of them. With Arcpy's powerful functions in geographic data processing and the ability possessed by multiprocessing package, parallelism in the ArcGIS environment can be realized.

Data Usage

To see the performance of parallel processing with an individual huge dataset, major roads of Florida was employed. Since Euclidean Distance function was applied, a processing extent covered the whole region of Florida is qualified, in terms of the purpose of the test. Figure 3-1 shows the major roads of the state of Florida.

To test parallel processing with large numbers of datasets, the LODES (LEHD[1] Origin-Destination Employment Statistics) datasets are employed. Figure 3-2 shows the downloading interface of LODES data on Census website. LODES datasets consists of a large number of CSV (Comma Separated Variable) files, which have three categories: OD (Origin-Destination) data, RAC (Residence Area Characteristic) data, and WAC (Workplace Area Characteristic) data. This research used the Florida 2010 WAC dataset. The specific file name is “fl_wac_S000_JT00_2010.csv”. The LODES data offers a high degree of geographic detail, with information available at the Census block level. However, to serve the test purpose, the data used in this research was aggregated to the Census block group level. Furthermore, the number of fields used is reduced to forty from the original fifty-three. Table 3-1 shows the field names and explanations that are being used.

Conceptual Model

Theoretically, there are two general situations in which parallelism can improve the performance of data processing. One is processing one dataset with large numbers

[1] LEHD: Longitudinal Employment-Household Dynamics program was launched by U.S. Census Bureau in 1999. It is part of the Center for Economic Studies at the U.S. Census Bureau. The LEHD program produces new, cost effective, public-use information combining federal, state and Census Bureau data on employers and employees under the Local Employment Dynamics (LED) Partnership. (LEHD Origin-Destination Employment Statistics (LODES) Dataset Structure Format Version 7.0, 2013)

of features, the other is processing large numbers of individual datasets. Figure 3-3 and Figure 3-4 described these two cases respectively.

Parallel Processing One Dataset with Large Numbers of Features

To be able to process one big dataset with multiple processors, a large dataset will be divided into several smaller datasets, so that individual processor can be employed to do the separate work. Euclidean Distance is used to evaluate this type of parallel processing. It is a function of ArcPy in the ArcGIS spatial analyst extension module (arcpy.sa). Generally speaking, this function calculates the Euclidean distance to the closest source for each cell and returns an output raster with the value of the calculations. Figure 3-5 shows the illustration of this function. The following formula explains how the Euclidean distance is calculated in the two-dimensional space.

$$d(a,b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

Where,

d(a, b) is the Euclidean distance from point a to point b;

x_a and x_b are the X-axis coordinate values;

y_a and y_b are the Y-axis coordinate values.

As discussed above, to parallelize the task of performing a Euclidean distance analysis for major roads in the whole state of Florida, the big task should be divided into small pieces, so that, multiple cores can compute simultaneously. A logic and proper division is to divide the processing extent by counties. Through setting the Extent parameter in ArcPy's environment class, sixty-seven counties in Florida become sixty-seven small tasks which are processed by multiple processors by turns. To eliminate the boarder effect, a 2-mile buffer is applied to each processing extent.

The research employed Mosaic dataset which allows users to store, manage, view, and query a collection of raster data. These capabilities make the Mosaic dataset a perfect solution to merge the individual results of parallel back together. Because of the 2-mile buffer, there would be overlaps on the boarder of each county. Mosaic dataset also allows users to set the mosaic operation to calculate the cell value in overlapping areas. These operations include: first, last, min, max, mean, blend, and sum. In this case, since Euclidean distance is calculating the distance to the closest source, the min (minimum) operation will be applied. This part of research will create two outputs, one of which is generated by executing in ArcMap application without parallelization, and the other is by mosaicked individual results processed by parallelizing. To test the similarity of the two outputs, the ArcGIS Cell Statistics Tool is utilized. Figure 3-6 shows the illustration of this function. The “Statistic Type” applied in this case is “Range” which calculates the difference between the largest and smallest value of cells on the same geographic location.

Parallel Processing Large Numbers of Individual Datasets

To test the performance of parallelizing large numbers of individual datasets, the Polygon to raster function was applied. This function allow users to convert a polygon shapefile to a raster dataset. The idea comes from the LUCIS[1] model developed by Dr. Paul Zwick. The first step of building the model is to create a CEM (Conflict Evaluation Matrix). Essentially, a CEM is a raster dataset that contains multiple fields. Each of

[1] LUCIS stands for Land-Use conflict identification strategy. It is a goal-driven GIS model that produces a spatial representation of probable patterns of future land use. (Carr & Zwick, 2007) A newer model called LUCIS^{PLUS}, of which PLUS stands for planning land-use scenario. The new model focuses on the allocation of future population and employment. It provides different scenarios of future land-use plan, whose product can be directly utilized by planning organizations.

these fields represent an attribute associated with the particular cell. The creation of CEM needs to combine more than ten raster datasets (the maximum value is twenty). Also, in the reality, the CEM is going to be generated multiple times to find a best one that matches a certain analytical purpose. Therefore, it is efficient to prepare all raster datasets which would possibly be used before doing the analysis. That's where parallel processing fits in this case.

As stated in the previous section of this chapter, forty fields of Florida 2010 wac LODES dataset is used to accomplish the purpose of the research. Each field is joined to a GIS shapefile of the state of Florida with a geographic unit of census block group. So that, forty shapefiles with field values represented different columns in LODES dataset will be converted to forty raster datasets. This scenario fulfills the purpose of evaluating parallel processing with large numbers of individual datasets.

Hardware Description

The performance of Parallelism, similar to the conventional single-processor computation, also highly depends on the capability of hardware. This research is being completed on a laptop whose major components' parameters are shown in Table 3-2. The CPU contains four physical cores. Because of Intel's Hyper-threading technology, each physical consists two logical processors. (Hyper-threading, 2014)

Table 3-1. LODES Workplace Area Characteristics (WAC) File Structure (LEHD Origin-Destination Employment Statistics (LODES) Dataset Structure Format Version 7.0, 2013)

Pos	Variable	Type	Explanation
1	w_geocode	Char15	Workplace Census Block Code
2	C000	Num	Total number of jobs
3	CA01	Num	Number of jobs for workers age 29 or younger
4	CA02	Num	Number of jobs for workers age 30 to 54
5	CA03	Num	Number of jobs for workers age 55 or older
6	CE01	Num	Number of jobs with earnings $1250/month or less
7	CE02	Num	Number of jobs with earnings $1251/month to $3333/month
8	CE03	Num	Number of jobs with earnings greater than $3333/month
9	CNS01	Num	Number of jobs in NAICS sector 11 (Agriculture, Forestry, Fishing and Hunting)
10	CNS02	Num	Number of jobs in NAICS sector 21 (Mining, Quarrying, and Oil and Gas Extraction)
11	CNS03	Num	Number of jobs in NAICS sector 22 (Utilities)
12	CNS04	Num	Number of jobs in NAICS sector 23 (Construction)
13	CNS05	Num	Number of jobs in NAICS sector 31-33 (Manufacturing)
14	CNS06	Num	Number of jobs in NAICS sector 42 (Wholesale Trade)
15	CNS07	Num	Number of jobs in NAICS sector 44-45 (Retail Trade)
16	CNS08	Num	Number of jobs in NAICS sector 48-49 (Transportation and Warehousing)
17	CNS09	Num	Number of jobs in NAICS sector 51 (Information)
18	CNS10	Num	Number of jobs in NAICS sector 52 (Finance and Insurance)
19	CNS11	Num	Number of jobs in NAICS sector 53 (Real Estate and Rental and Leasing)
20	CNS12	Num	Number of jobs in NAICS sector 54 (Professional, Scientific, and Technical Services)
21	CNS13	Num	Number of jobs in NAICS sector 55 (Management of Companies and Enterprises)
22	CNS14	Num	Number of jobs in NAICS sector 56 (Administrative and Support and Waste Management and Remediation Services)
23	CNS15	Num	Number of jobs in NAICS sector 61 (Educational Services)
24	CNS16	Num	Number of jobs in NAICS sector 62 (Health Care and Social Assistance)
25	CNS17	Num	Number of jobs in NAICS sector 71 (Arts, Entertainment, and Recreation)
26	CNS18	Num	Number of jobs in NAICS sector 72 (Accommodation and Food Services)

Table 3-1. Continued

Pos	Variable	Type	Explanation
27	CNS19	Num	Number of jobs in NAICS sector 81 (Other Services [except Public Administration])
28	CNS20	Num	Number of jobs in NAICS sector 92 (Public Administration)
29	CR01	Num	Number of jobs for workers with Race: White, Alone
30	CR02	Num	Number of jobs for workers with Race: Black or African American Alone
31	CR03	Num	Number of jobs for workers with Race: American Indian or Alaska Native Alone
32	CR04	Num	Number of jobs for workers with Race: Asian Alone
33	CR05	Num	Number of jobs for workers with Race: Native Hawaiian or Other Pacific Islander Alone
34	CR07	Num	Number of jobs for workers with Race: Two or More Race Groups
35	CT01	Num	Number of jobs for workers with Ethnicity: Not Hispanic or Latino
36	CT02	Num	Number of jobs for workers with Ethnicity: Hispanic or Latino
37	CD01	Num	Number of jobs for workers with Educational Attainment: Less than high school
38	CD02	Num	Number of jobs for workers with Educational Attainment: High school or equivalent, no college
39	CD03	Num	Number of jobs for workers with Educational Attainment: Some college or Associate degree
40	CD04	Num	Number of jobs for workers with Educational Attainment: Bachelor's degree or advanced degree

Table 3-2. List of Hardware's the Main Parameters

Component	Producer	Model	Main parameters
Motherboard	*Lenovo*	W530	2436 CTO
CPU	*Intel*	i7-3610 QM	2.30 GHz
Memory bank 1	*G skill*	F3-1600C9D-16GRSL	Capacity: 8 GB Speed: PC3 12800 Voltage: 1.35V
Memory bank 2	*G skill*	F3-1600C9D-16GRSL	Capacity: 8 GB Speed: PC3 12800 Voltage: 1.35V
Memory bank 3	*Hynix*	HMT351S6CFR8C	Capacity: 4 GB Speed: PC3 12800 Voltage: 1.5 V
Hard drive	*Samsung*	MZ-7TE1T0BW	Device Type: Solid state drive Capacity: 1 TB Read speed: 540 MB/s Write speed: 520 MB/s

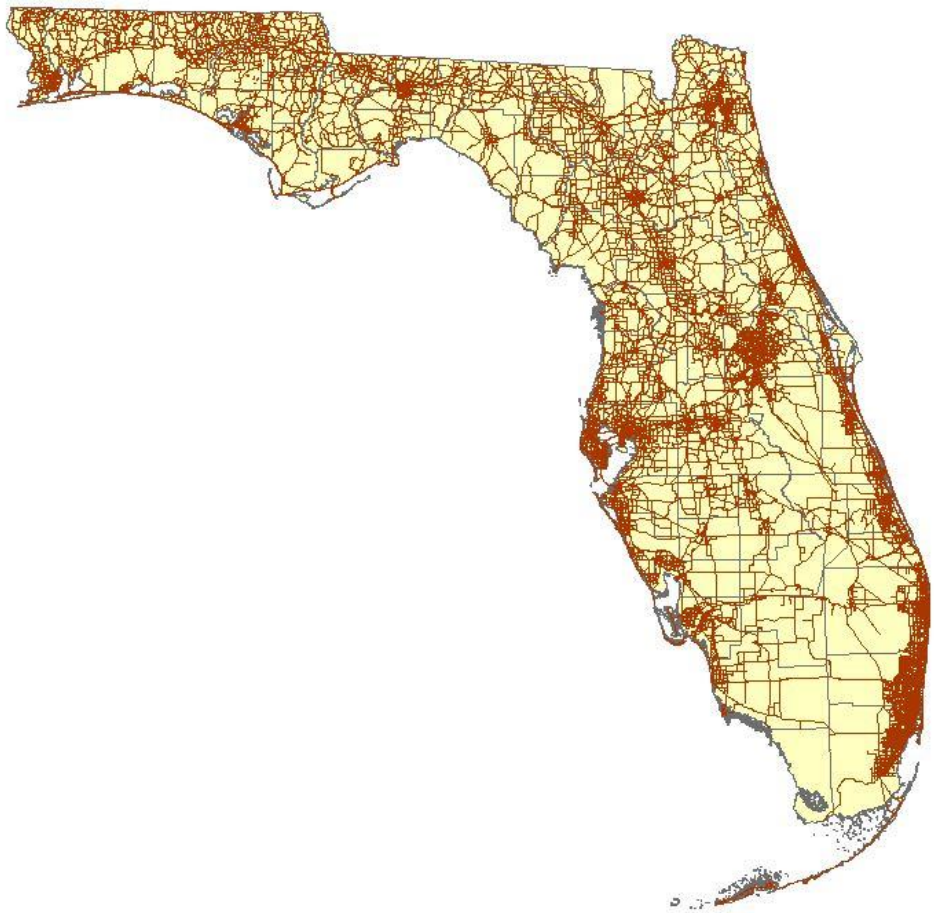

Figure 3-1. Major Roads in the State of Florida

Download LODES data*:

Version: LODES7 ▾ State/Territory: Florida ▾ Type: Workplace Area Characteristics (WAC) ▾

View Files Close

Metadata for FL | Geography crosswalk for FL | FL md5sum file

fl_wac_S000_JT00_2002.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT00_2003.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT00_2004.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT00_2005.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT00_2006.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT00_2007.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT00_2008.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT00_2009.csv.gz	14 Nov 2013 12:51	3 MB
fl_wac_S000_JT00_2010.csv.gz	14 Nov 2013 12:50	3 MB
fl_wac_S000_JT00_2011.csv.gz	14 Nov 2013 12:51	3 MB
fl_wac_S000_JT01_2002.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT01_2003.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT01_2004.csv.gz	14 Nov 2013 12:51	2 MB
fl_wac_S000_JT01_2005.csv.gz	14 Nov 2013 12:51	2 MB

Figure 3-2. Interface for Downloading LODES Dataset

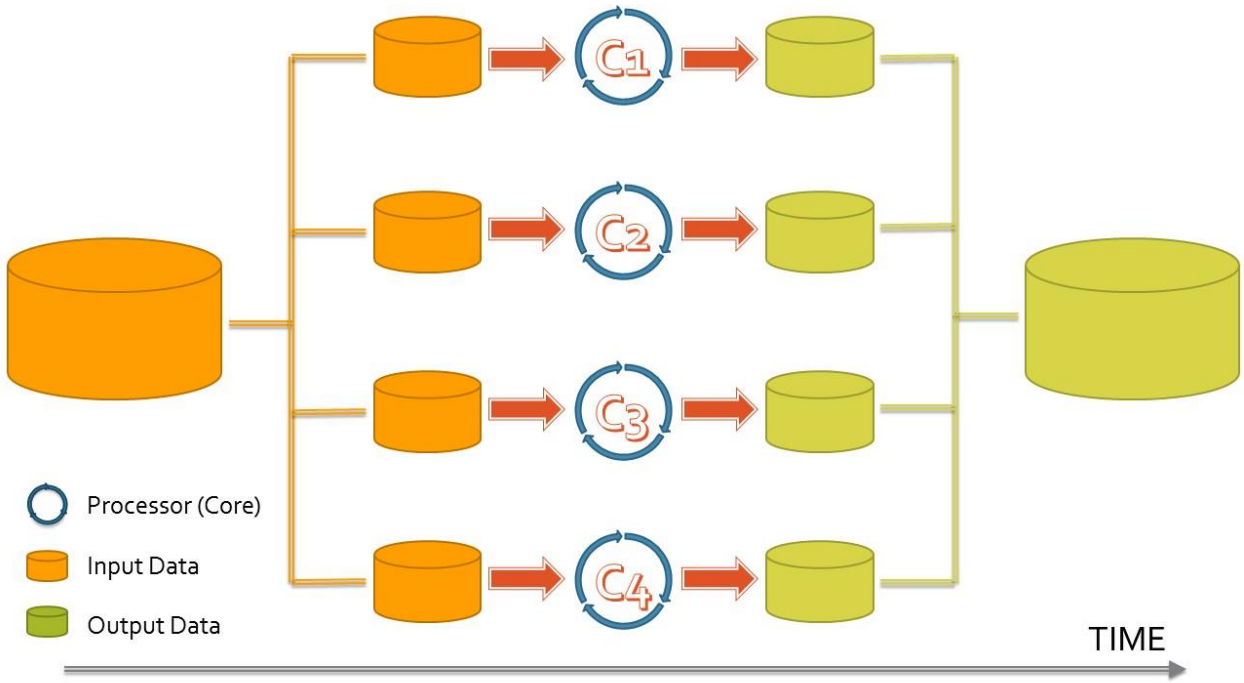

Figure 3-3. Conceptual Model for Parallel Processing One Dataset with Large Numbers of Features

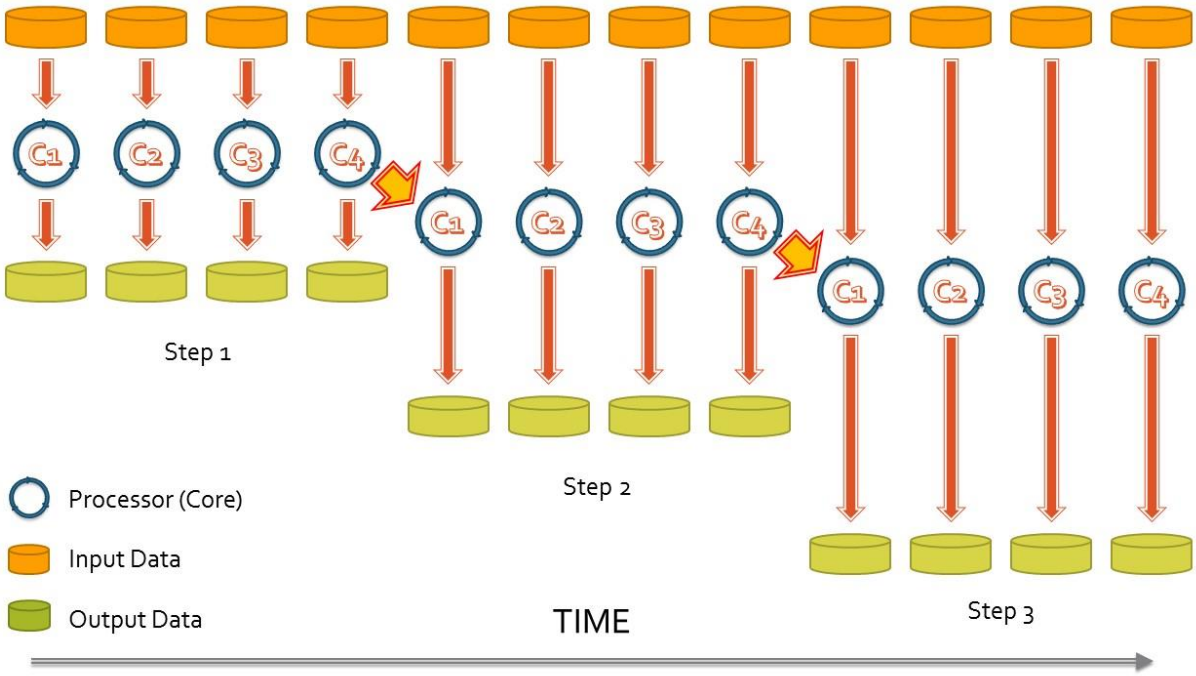

Figure 3-4. Conceptual Model for Parallel Processing Large Numbers of Individual Datasets

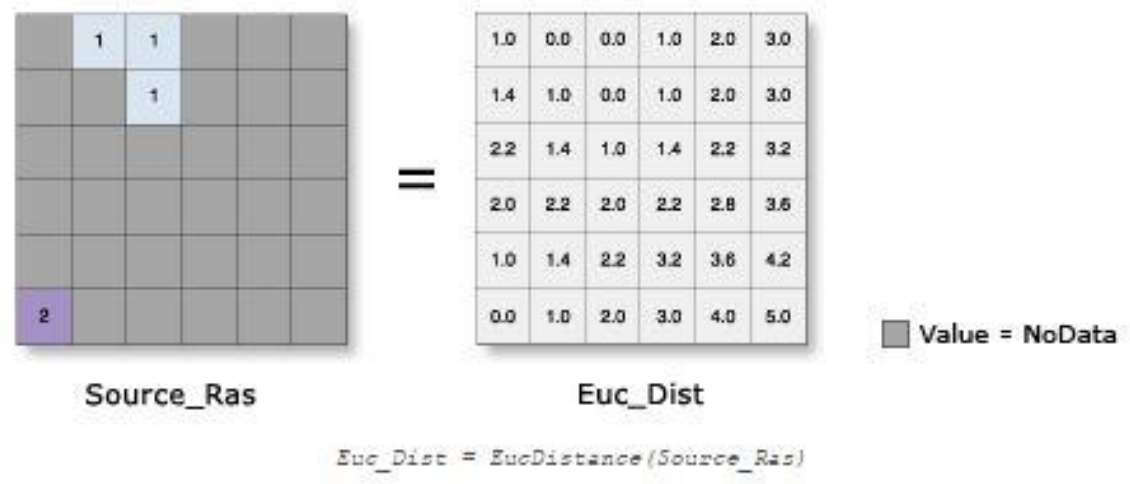

Figure 3-5. Illustration of Euclidean Distance (Euclidean Distance (Spatial Analyst), 2014)

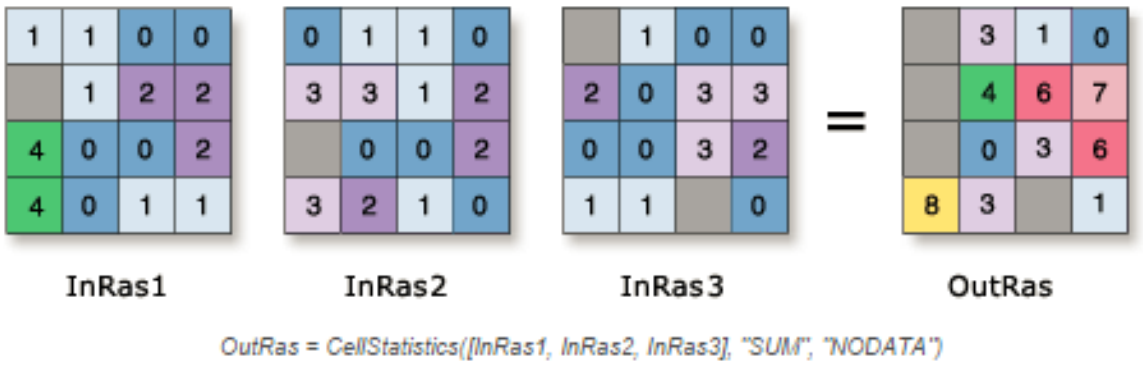

Figure 3-6. Illustration of Cell Statistics (Cell Statistics (Spatial Analyst), 2014)

CHAPTER 4
RESULTS

Performance Comparison In Terms of Time Consumption

Parallel Processing One Dataset with Large Numbers of Features

Figure 4-1 shows the CPU usage when eight processors are being used in a parallelization. The average time consumption for performing a typical Euclidean distance analysis within the ArcMap application on researcher's laptop is about 18.5 minutes. Using four processors to parallel Processing sixty-seven counties individually took 4.06 minutes. It also took about 1.25 minutes to merge the individual raster datasets back through *Add Raster to Mosaic Dataset* function. Therefore, using 4 cores is 3.48 times faster than only employ single processor.

Parallel Processing Large Numbers of Individual Datasets

Figure 4-2 shows the general time consumption of performing Euclidean distance function with different cell size by employing different numbers of processors. Figure 4-3 describes the speed differences among using two processors, four processors and eight logical processors comparing with single processor. Figure 4-4 explains how much time could save by using two processors, four processors and eight logical processors comparing with only applying a single processor. Generally speaking, in terms of time consumption, employing two processors is 1.85 times faster than using single processor. The numbers for four processors and eight processors (logically) are 3.16 and 4.32 respectively.

Performance Comparison In Terms of Output

Regarding parallel processing with large numbers of datasets, because each individual dataset is treated separately by multiple processes, the outputs will be exactly

the same with using single processor. Therefore, this type of parallel processing is unnecessary and will not be included in this comparison. Figure 4-5 is the images of the output raster dataset of the non-parallelized processing. And, its counterpart Figure 4-6 shows the output of the parallelized process. Figure 4-7 shows the result of the analysis of Cell Statistic function whose statistical values are listed in Table 4-1. The minimum difference is 0. However, the maximum difference is 27409.83.

Table 4-1.Statistical Results of Cell Statistics Function

Statistics	Value
Count	1470549742
Minimum	0
Maximum	27,409.82813
Sum	230,823,154,700
Mean	156.9638538
Standard Deviation	1,024.2925

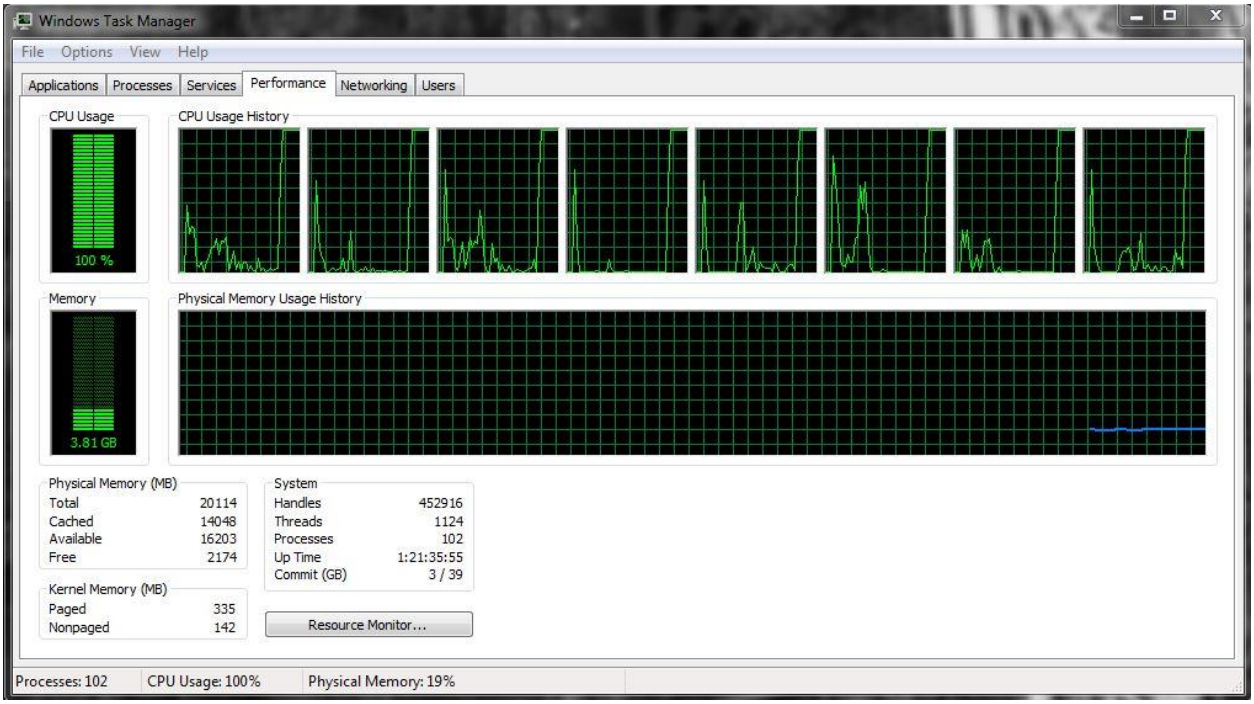

Figure 4-1. CPU Usage While Performing an Eight-Processor Parallelization

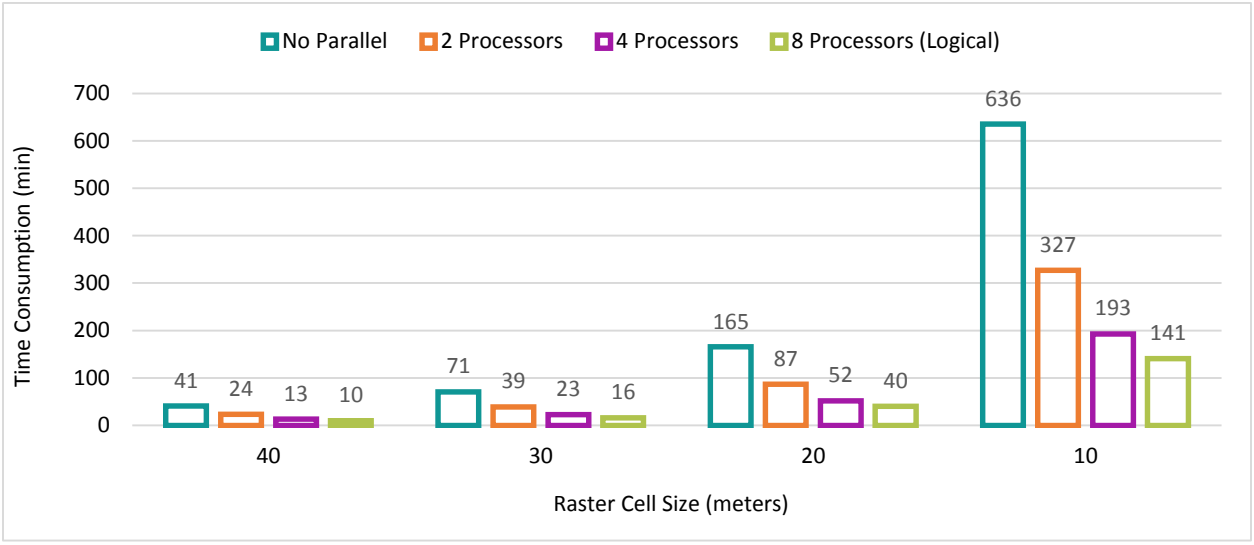

Figure 4-2. Time Consumption by Employing Different Number of Processors

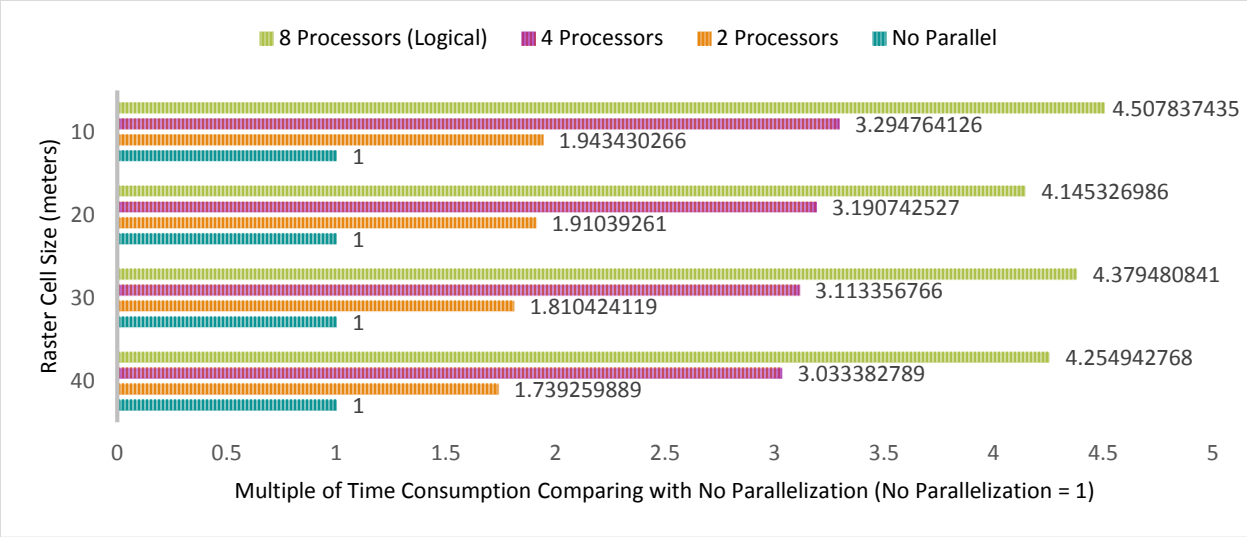

Figure 4-3. Multiple of Time Consumption Comparing with Non-parallelization

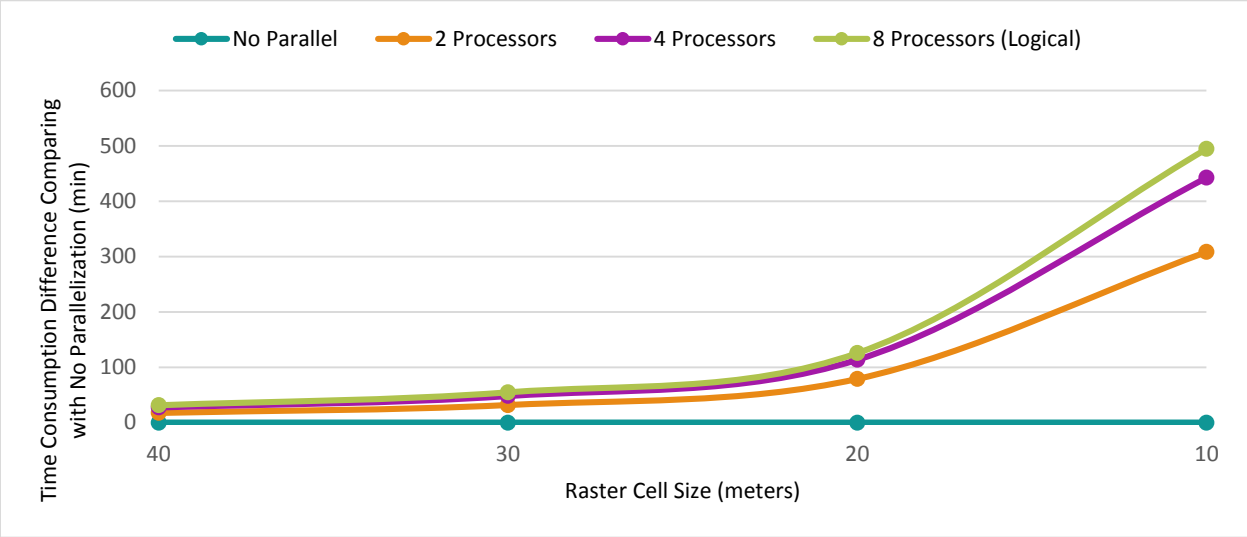

Figure 4-4. Time Saved by Performing Euclidean Distance of Different Cell Size

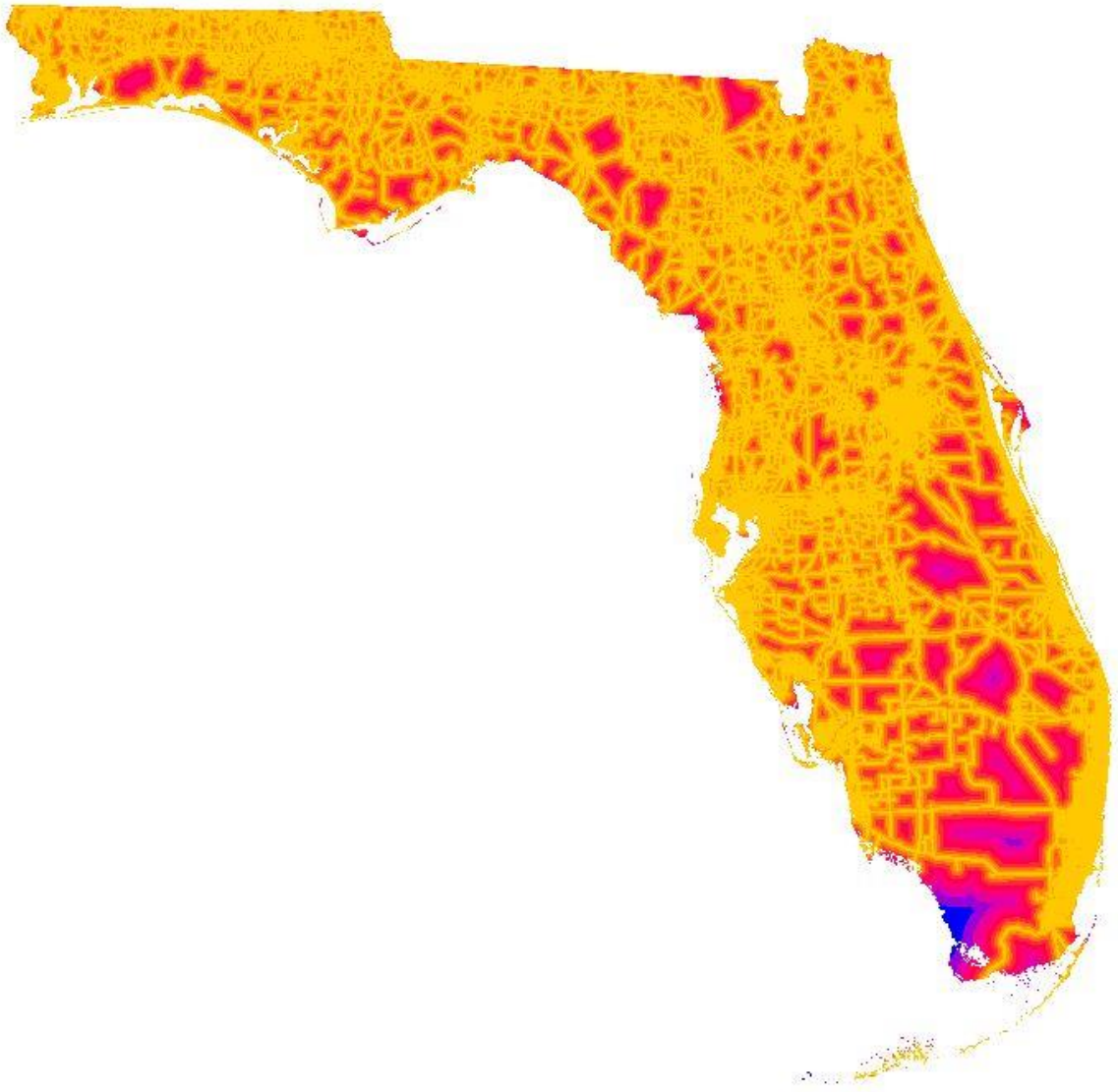

Figure 4-5. Non-Parallelized Result (Parallel Processing One Dataset with Large Numbers of Features)

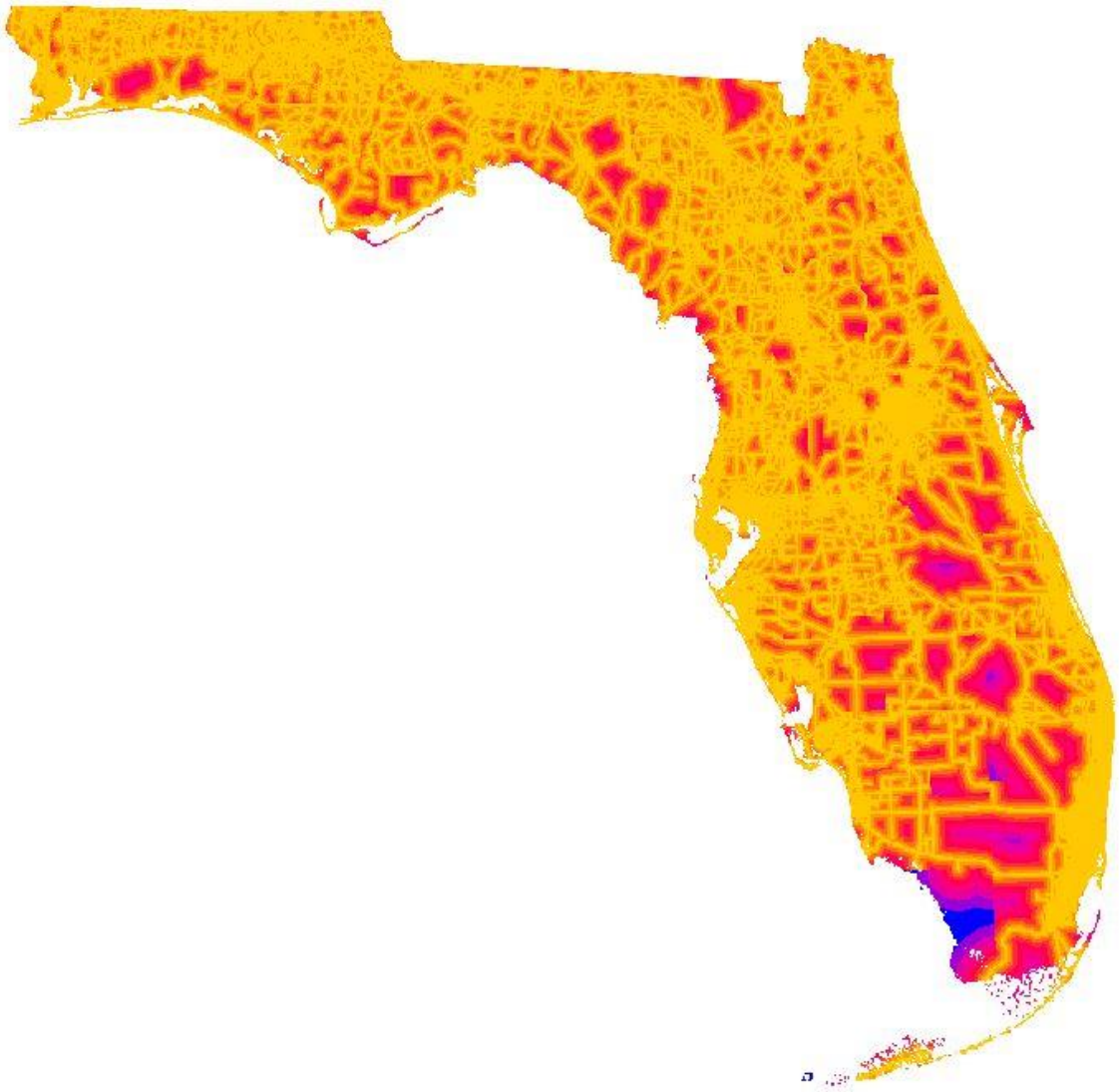

Figure 4-6. Parallelized Result (Parallel Processing One Dataset with Large Numbers of Features)

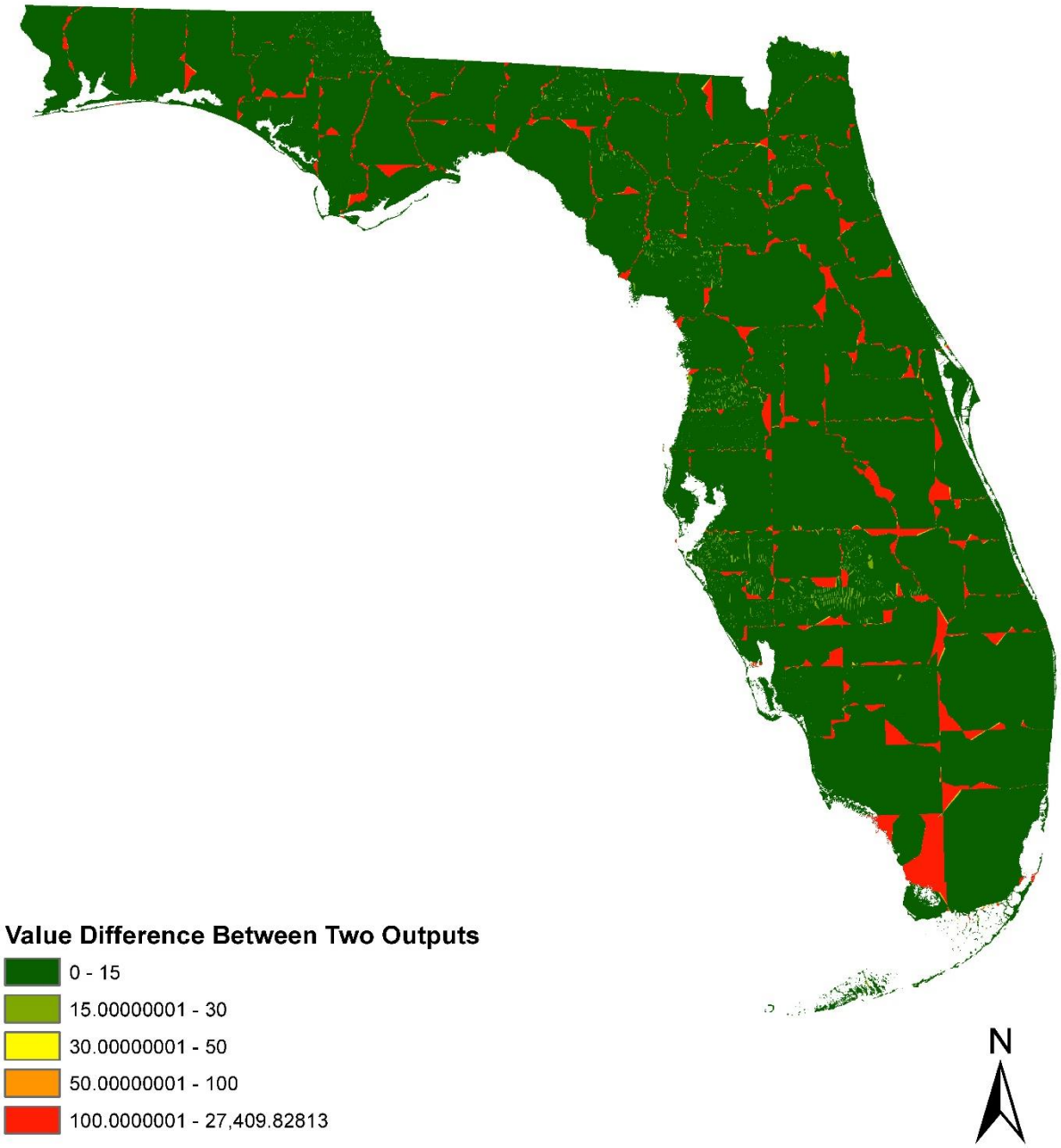

Figure 4-7. Cell Statistics Output

CHAPTER 5
DISCUSSION

Figure 4-6 has distinctly shown the differences between parallelized and non-parallelized outputs, in terms of parallel processing one dataset with large numbers of features. Although a 2-mile buffer was applied to each county, big differences can still be found on the boarders of each county. Increase the distance of the buffer may reduce this effect, however, it will also increase the time consumption. Besides, even though there is a perfect buffer distance that can one-hundred percent eliminate this issue, finding out this number is time consuming and difficult. Therefore, parallelization should not be used in processing one dataset, although it can increase the speed of processing.

On the other hand, the positive results of parallel processing with large numbers of individual datasets indicates a big potential of parallelizing ArcGIS tasks in this way. The performance has apparently improved by using multiple processors. With the capability of the laptop used in the research, performing a 10-meter cell size parallel processing saved the author more than eight hours. It is excited by only imagining how fast the speed would be, if performing the parallelism on a server with hundreds of cores. Besides, the high performance, some other key findings are also worthy to be discussed here.

The heavier the task is, the more time it would be saved by parallelization. Figure 4-3 has vividly explained this point. When the cell size used by Euclidean distance is 40 meters, the difference between using single processor and multiple processors is not very obvious. The differences become bigger as the cell size become smaller.

Parallelizing the tasks requires a little period of time. Instead of two times, using two processors is 1.85 times faster than a single-core process. This means the CPU would take some time before it can enter into the parallelizing mode. In addition, the performance improvement gained by increasing the processor number from two to four, is not as much as it gained from single to two. A possible explanation to this situation would be the performance gained from single to two processors is the biggest performance improvement can be get. It is very similar to a change from nothing to one. However, with a four physical core laptop, there are not so many scenarios can be test to find out the real reason. Future research on servers may be able to give us an answer.

Logical processor is not equivalent to physical processor in terms of processing capability. The performance improvement gained by increasing the processor number from four to eight, is not as much as it gained from two to four. The reason is because the two logical processors share the same resources from one physical processor.

CHAPTER 6
CONCLUSION

Parallelizing with ArcGIS Functions

It is important to point out that parallel processing may not always increase the speed of computation, unless one utilizes it in the right way. First of all, parallel programming is difficult, and it becomes even trickier when it works with ArcPy functions. To avoid fatal errors, parallelism can only be called once in one script. In fact, Programmers always write long scripts to meet their goals. Sometimes, these codes can be longer than hundreds of lines. However, in parallel programming, it is irrational and inefficient to run all the codes in parallelizing mode. Before writing the scripts, researchers have to think about what the most CPU-consuming task is, and only parallelize that part. In other words, only the most difficult part should be handed to multiple processors.

Moreover, when working with ArcPy, researchers should keep in mind couples of additional rules besides what are wrote in Python documents.

Do not use Geodatabase. An ArcGIS geodatabase would apply an exclusive schema lock to prevent other applications or users from accessing it, at any time while the dataset is open or being modified. Similar to GIL applied to CPython, this schema lock blocked the way of doing parallel processing with ArcGIS Geodatabase. The solution to this issue is to create an ArcInfo workplace for storing both the input and output data.

Do not include more than one parallelism in one script. Because of the differences in scale and complexity among different geographic datasets, time consumed to process them varies greatly. Thus, two processes that started together

could be finished at different point of time. If there were two or more parallelisms included in one script, a very likely result would be, with one parallelism not finished, the other parallelism already started. It may lead to a fatal error, if cores are in different parallelisms.

Work with ArcGIS ModelBuilder to keep script organized. Researchers have to think through all the steps required in order to fulfill the final purpose. Besides, it is even more important to be aware of the “major function”, that is, what function really needs to parallelize, because only one ArcPy function is necessary to be parallelized in most cases. In terms of setting an effective workflow, ArcGIS ModelBuilder could be a helpful tool in this circumstances.

Do not use 100% of CPU. The key point of using parallel processing is to keep all the available cores installed in CPU working until the total work is completed. However, with some big projects, it would be good to avoid using one-hundred percent of the CPUs’ capacity, since it would cause the CPU overheating, especially with laptops whose CPU fan is not as powerful as a desktop computers, not to mention servers.

Not only CPU matters. The performance of parallel processing is not only affected by the capability of CPU. The computer performing its functionality with all its components working together as a whole. The author used to only have 4GB of computer RAM (Random-access memory) on the laptop. For the purpose of this research, the laptop has been upgraded to 20 GB RAM, which significantly increase the computation speed of CPU. Another important component is the hard drive which determines the reading and writing speed of a computer. The author also upgraded the

previous hard disk drive (HDD) to a solid state drive (SSD). With the new drive, the computer possesses a 540 MB/s reading speed and 520 MB/s writing speed.

Thirdly, individual tasks should be similar to one another, in terms of time consumption. For example, if a parallelized code had four tasks, one of which was much bigger than the other three, the three smaller tasks would spend most of time in waiting for the big one. Researchers should distribute the general task evenly to each individual processor to leverage all of the CPU's capability.

Last but not least, parallel processing, in most cases, should be used in a pre-analyzing stage. Although it is possible to do higher level analysis with parallel, it is much more efficient and powerful in the data-preparation phase. For one thing, it is not easy to code a whole analytical process, especially included parallelizing. On the other hand, purpose of programming, by its nature, is to utilize the computational power of computers to automate regular tasks, which could save people from doing repeated and meaningless works. Obviously, analysis is not one of those works. Thus, the best scenario is to do parallel processing at the beginning of research. With the power possessed by parallelism, planners can access to all the data that would be possibly used in the research.

Evaluating the Performance

The result of the research proved parallelism can significantly improve the performance of geographic datasets processing. It also told us the best situation to apply parallelization is parallel processing large numbers of individual datasets. Two general conclusions can be made, one of which is that, the more processors are employed, the less time it cost. On the other hand, as the task gets heavier, the more benefits in parallelization can be gained.

The number of tests that can be done in this research is limited by the resources available to the author. With only four physical cores available, the performance of parallel processing with more than four cores cannot be known by current research. This situation hindered us to get deeper understanding with characteristics of parallelizing the ArcGIS functions. Future research on servers may be able to find out the answer. Another imperfection of this research is the application of parallel processing. Serving as a test, only two functions in ArcGIS are tested in the research. However, better applications need to be considered to utilize the big potential of parallelism.

Utilizing parallel processing would make planners more efficient in the data processing stage and save planner both a lot of time and energy, so that they can concentrate more on the real analysis. The research opened a door for planners to leverage the so-called big data analysis to solve planning problems.

CHAPTER 7
FUTURE RESEARCH

This research proved that it is possible to combine Python's capability in parallel programming and the capability of ArcPy in processing geographic data together to leverage the computational power of multiple processors. The results indicated a positive correlation between the processing speed and the number of processors. This conclusion strongly suggests moving the research to the next level, which is, processing big geographic data on servers.

The performance of geographic data processing would be tremendously magnified by thousands of processors installed on server. Specifically, planners can process more than one thousand geographic datasets using the same time as they used to process one dataset. Moreover, possibilities are not only restricted to processing the newly generated data, but also let us go back to the history to process historical datasets with high performance. By doing that, we can get a better understanding of the courses of the development of our cities.However, more researches need to be done to realize parallelization on a server. As most servers are not organized in Windows operation system, the interface and operations are completely different from running applications on a laptop or desktop computer. Predictably, a good amount of tests and explorations is down the road, before we can employ the entire range of capability provided by servers.

Another direction of future research is to parallel programming a series of ArcPy functions to be able to do more advanced data processing with parallelization. In this research, the author tests the feasibility of parallelizing Euclidean distance and Polygon to raster functions respectively. However, the parallelization was performed by only

executing one function at a time. In most cases, the raw data is stored in a .csv or .txt format. This is for the purpose of storing and managing the data easily. But, it is not able to directly use these format to do geographic data process, since the table of geographic datasets are stored in .dbf format or as a geodatabase object.

To be able to use those raw data, several steps of transformations have to be made. Therefore, parallelizing a series of functions to realize the transformations has significant meanings, in terms of utilizing raw data. Although the research concluded that it is not such a good application for using parallel processing in the analytical stage, parallel programming more than one ArcPy functions can be realized. In fact, theoretically, any combination of ArcPy functions following a regular pattern is possible to be parallelized.

Some programmers said: if one thing is worth to do, it is worth to do in parallelization. With further researches being done in this topic, planners will have more capabilities in processing “big data” that supports them to make a better plan.

APPENDIX A
A SIMPLE PARALLEL PROGRAMMING TEST SOURCE CODE

```
import math
import time
from multiprocessing import Process, Queue
from Queue import Empty
def PythagoreanNum(n):
    for c in range (n+1, n+9):
        for a in range (1, c):
            for b in range (a, c):
                c_square = a**2 + b**2
                if c == math.sqrt(c_square):
                    print a, b, c
    return
def do_jobs(q):
    while True:
        try:
            x = q.get(block=False)
            PythagoreanNum(x)
        except Empty:
            break
if __name__ == '__main__':
    t1 = time.time()
    workqueue = Queue()
```

```
for i in range(0, 3000, 8):
    workqueue.put(i)
jobs = [Process(target = do_jobs, args=(workqueue,)) for i in range(8)]
for job in jobs:
    job.start()
for job in jobs:
    job.join()
t2 = time.time()
TimeTaken = (t2-t1)/60
print TimeTaken
```

APPENDIX B
PARALLEL PROGRAMMING *arcpy.Buffer_analysis* SOURCE CODE

```
import arcpy
from arcpy import env
import arcpy.sa
from multiprocessing import Process, Queue
from Queue import Empty
import time
import os
def select(fc, fieldname, attribute, output):
    env.overwriteOutput = True
    whereclause1 = '"' + fieldname + '" = ' + "\'" + attribute + "\'"
    out = output
    arcpy.Select_analysis(fc, out, whereclause1)
    return
def GetAttributes(fc, field):
    valueList = []
    valueSet = set()
    rows = arcpy.SearchCursor(fc)
    for row in rows:
        value = row.getValue(field)
        if value not in valueSet:
            valueList.append(value)
        valueSet.add(value)
```

```
    valueList.sort()
    return valueList
def do_buffer(q, location, distance):
    while True:
        try:
            x = q.get(block=False)
            cntfolder = location + "\\" + x
            arcpy.env.workspace = cntfolder
            arcpy.Buffer_analysis(x.upper() + "_MASK.shp", x.upper() + "_BUFF.shp",
distance, "", "ROUND", "", "")
        except Empty:
            break
    return
if __name__ == "__main__":
    T1 = time.time()
    env.overwriteOutput = True
    env.workspace = "E:\\UFL\\Thesis Relative\\Parallel with Mosaic\\WorkingData"
    arcpy.CreateFolder_management(env.workspace, "output")
    outputfolder = "E:\\UFL\\Thesis Relative\\Parallel with Mosaic\\WorkingData\\output"
    for cnt in GetAttributes("cntbnd_jul11.shp", "NAME"):
        arcpy.CreateArcInfoWorkspace_management(outputfolder, cnt.split('-')[0])
        cntfolder = outputfolder + "\\" + cnt.split('-')[0]
```

```
        select("cntbnd_jul11.shp", "NAME", cnt, cntfolder + "\\" + cnt.split('-')[0] +
"_MASK.shp")
        arcpy.Copy_management("majrds_feb11.shp", cntfolder + "\\majrds_feb11.shp",
"")
    workqueue = Queue()
    for cnt in os.listdir(outputfolder):
        workqueue.put(cnt.upper())
    jobs = [Process(target = do_buffer, args=(workqueue, outputfolder, "2 Miles")) for i in
range(8)]
    for job in jobs:
        job.start()
    for job in jobs:
        job.join()
    T2 = time.time()
    print "Total time taken: {} minutes.".format((T2-T1)/60)
```

APPENDIX C
PARALLEL PROGRAMMING *arcpy.sa.EucDistance* SOURCE CODE

```
import arcpy
from arcpy import env
import arcpy.sa
from multiprocessing import Process, Queue
from Queue import Empty
import time
import os
def Extent(fc):
    extent = arcpy.Describe(fc).extent
    west = extent.XMin
    south = extent.YMin
    east = extent.XMax
    north = extent.YMax
    return [west, south, east, north]
def euc(location, cnt, fc, cellsize):
    arcpy.CheckOutExtension("spatial")
    arcpy.env.workspace = location
    arcpy.env.mask = location + "\\" + cnt + "_MASK.shp"
    bufferextent = location + "\\" + cnt + "_BUFF.shp"
    arcpy.env.extent = arcpy.Extent(Extent(bufferextent)[0], Extent(bufferextent)[1],
Extent(bufferextent)[2], Extent(bufferextent)[3])
    print arcpy.env.extent
```

```
    EUC = arcpy.sa.EucDistance(fc, "", cellsize, "")
    EUC.save(location + "\\ed" + cnt[:5])
    return
def do_euc(q, location, fc, cellsize):
    while True:
        try:
            x = q.get(block=False)
            cntfolder = location + "\\" + x
            euc(cntfolder, x.upper(), fc, cellsize)
        except Empty:
            break
    return
if __name__ == "__main__":
    T1 = time.time()
    arcpy.env.overwriteOutput = True
    arcpy.env.workspace = "E:\\UFL\\Thesis Relative\\Parallel with Mosaic\\WorkingData"
    outputfolder = "E:\\UFL\\Thesis Relative\\Parallel with Mosaic\\WorkingData\\output"
    workqueue = Queue()
    for cnt in os.listdir(outputfolder):
        workqueue.put(cnt)
    jobs = [Process(target = do_euc, args=(workqueue, outputfolder, "E:\\UFL\\Thesis
Relative\\Parallel with Mosaic\\WorkingData\\majrds_feb11.shp", 10)) for i in range(4)]
    for job in jobs:
```

```
    job.start()
for job in jobs:
    job.join()
T2 = time.time()
print "Total time taken: {} minutes.".format((T2-T1)/60)
```

APPENDIX D
PARALLEL PROGRAMMING *arcpy.PolygonToRaster_conversion* SOURCE CODE

```
import arcpy
from arcpy import env
import arcpy.sa
from multiprocessing import Process, Queue
from Queue import Empty
import time
import os
def Extent(fc):
    extent = arcpy.Describe(fc).extent
    west = extent.XMin
    south = extent.YMin
    east = extent.XMax
    north = extent.YMax
    return [west, south, east, north]
def grid(location, tbl, fc, cellsize):
    arcpy.CheckOutExtension("spatial")
    arcpy.env.workspace = location
    arcpy.env.mask = fc
    arcpy.env.extent = arcpy.Extent(Extent(fc)[0], Extent(fc)[1], Extent(fc)[2],
Extent(fc)[3])
    print arcpy.env.extent
    arcpy.PolygonToRaster_conversion(fc, tbl, tbl + "", "", "", cellsize)
```

```
    return
def do_grid(q, location, fc, cellsize):
    while True:
        try:
            x = q.get(block=False)
            folder = location + "\\" + x
            arcpy.env.workspace = folder
            grid(folder, x.upper(), fc, cellsize)
        except Empty:
            break
    return
if __name__ == "__main__":
    arcpy.env.overwriteOutput = True
    arcpy.env.workspace = "E:\\UFL\\Thesis Relative\\Parallel with
Census\\WorkingData"
    arcpy.CreateFolder_management(env.workspace, "output")
    outputfolder = "E:\\UFL\\Thesis Relative\\Parallel with Census\\WorkingData\\output"
    fieldnames = [f.name for f in arcpy.ListFields("FL_WAC.dbf")]
    for field in fieldnames:
        if field <> "CREATEDATE" and field <> "BLKGRPID" and field <> "OID":
            print field
            arcpy.CreateArcInfoWorkspace_management(outputfolder, field)
            tblfolder = outputfolder + "\\" + field.lower()
```

```
            fm = arcpy.FieldMap()
            fm1 = arcpy.FieldMap()
            fms1 = arcpy.FieldMappings()
            fm1.addInputField("FL_WAC.dbf", "BLKGRPID")
            fm.addInputField("FL_WAC.dbf", field)
            fms1.addFieldMap(fm)
            fms1.addFieldMap(fm1)
            arcpy.TableToTable_conversion("FL_WAC.dbf", tblfolder, field + ".dbf", "", fms1,
""")
            arcpy.MakeFeatureLayer_management ("cenblkgrp10.shp", "blkgrp_lyr")
            arcpy.AddJoin_management("blkgrp_lyr", "GEOID10", tblfolder + "\\" + field +
".dbf", "BLKGRPID", "KEEP_ALL")
            fm2 = arcpy.FieldMap()
            fm3 = arcpy.FieldMap()
            fms2 = arcpy.FieldMappings()
            fm2.addInputField("blkgrp_lyr", "cenblkgrp10.GEOID10")
            fm3.addInputField("blkgrp_lyr", field + "." + field)
            fms2.addFieldMap(fm2)
            fms2.addFieldMap(fm3)
            arcpy.FeatureClassToFeatureClass_conversion("blkgrp_lyr", tblfolder,
"CBG10.shp", "", fms2, "")
    T1 = time.time()
    workqueue = Queue()
```

```
    for tbl in os.listdir(outputfolder):
        workqueue.put(tbl)
    jobs = [Process(target = do_grid, args=(workqueue, outputfolder, "CBG10.shp", 40))
for i in range(2)]
    for job in jobs:
        job.start()
    for job in jobs:
        job.join()
    T2 = time.time()
    print "Total time taken: {} minutes.".format((T2-T1)/60)
```

LIST OF REFERENCES

A Quick Tour of ArcPy. (2014, March). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/#/A_quick_tour_of_ArcPy/000v000000001000000/

Allen, D. W. (2011). *Getting to Know ArcGIS ModelBuilder.* Redlands, CA: Esri Press.

Allen, D. W. (2014). *GIS Tutorial for Python Scripting.* Redlands, CA: Esri Press.

Application Programming Interface. (2014, September). Retrieved from Wikipedia, The Free Encyclopedia: http://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=623906555

Athanasias, D. (2014, 5 11). *Global Interpreter Lock*. (Python Software Foundation) Retrieved from Python Wiki: https://wiki.python.org/moin/GlobalInterpreterLock

Batty, M. (2013). Big data, smart cities and city planning. *Dialogues in Human Geography, 3*(3), 274-279.

Beazley, D. (2012, October). Secrets of the Multiprocessing Module. *Login, 37*(5).

Beazley, D., & Jones, B. K. (2013). *Python Cookbook.* Sebastopol, CA: O'Reilly & Associates.

Bettencourt, L. M. (2013, September 17). The Uses of Big Data in cities. Santa Fe, NM, United States of America.

Buffer (Analysis). (2014, March). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//000800000019000000

Carr, M. H., & Zwick, P. D. (2007). *Smart Land-Use Analysis.* Redlands, CA: Esri Press.

Cell Statistics (Spatial Analyst). (2014, April). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//009z0000007q000000

Create ArcInfo Workspace (Data Management). (2014, August). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//0017000000ps000000

Create Mosaic Dataset (Data Management). (2014, August). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//00170000008n000000

Crim, S. (2012, July). *I'll take you there -- Using Census Longitudinal Employer-Household Dynamics Data For Assessing Transit Service Coverage.* Retrieved from Ride New Orleans: http://rideneworleans.org/

Dalcin, L. D., Paz, R. R., Kler, P. A., & Cosimo, A. (2011). Parallel Distributed Computing Using Python. *Advances in Water Resources, 34*(9), 1124-1139.

(2013). *Designing with data: Shaping our future cities.* Royal Institute of British Architects; ARUP. London: ARUP.

Durgaprasad, P. (2011, December). Parallel Computing: High Performance. *International Journal of Emerging Technology and Advanced Engineering, 1*(2), 97-101.

Essential ArcPy vocabulary. (2014, March). Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/#/Essential_ArcPy_vocabulary/000v000000v6000000/

Euclidean Distance (Spatial Analyst). (2014, April). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//009z0000001p000000

FieldMap (arcpy). (2014, March). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//018z0000007p000000

FieldMappings (arcpy). (2014, March). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//018z0000007800000

Flynn, M. J. (1966). Very High-Speed Computing Systems. *IEEE. 54*, pp. 1901-1909. Evanston: IEEE. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1447203&isnumber=31091

Forum, M. (1994). Mpi: A message-passing interface standard. *International Journal of Supercomputer Applications*.

Geodatabase Locks. (2012, October). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.1: http://resources.arcgis.com/en/help/main/10.1/index.html#//019000000004000000

Hinsen, K. (n.d.). Parallel scripting with python. *Computing in Science & Engineering, 9*(6), pp. 82-89. doi:10.1109/MCSE.2007.117

History of Python. (2014, August 15). (Wikipedia, The Free Encyclopedia) Retrieved October 3, 2014, from http://en.wikipedia.org/w/index.php?title=History_of_Python&oldid=621406482

Hyper-threading. (2014, September). Retrieved from Wikipedia, The Free Encyclopedia: http://en.wikipedia.org/wiki/Hyper-threading

I/O bound. (2014, September). Retrieved from Wikipedia, The Free Encyclopedia: http://en.wikipedia.org/w/index.php?title=I/O_bound&oldid=625147905

Kerr, N. T. (2009). *Alternative Approaches to Parallel GIS Processing.* Master Thesis, Arizona State Universtiy, Department of Computer Science, Tempe.

LEHD Origin-Destination Employment Statistics (LODES) Dataset Structure Format Version 7.0. (2013, June 6). *LODES 7.0 Tech Doc*. Washington D.C. Retrieved from United States Census Bureau: http://lehd.ces.census.gov/data/lodes/LODES7/LODESTechDoc7.0.pdf

ListFeatureClasses (arcpy). (2014, March). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#/ListFeatureClasses/03q300000023000000/

Longitudinal Employer-Household Dynamics. (2014). Retrieved from United States Census Bureau: http://lehd.did.census.gov/

Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media, Inc.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011, June). *Big data: The next frontier for innovation, competition, and productivity.* McKinsey Global Institute. McKinsey & Company.

Matloff, N. (2012). Programming on Parallel Machines. Davis, CA.

Mattson, T. (2009, August 10). *How to sound like a Parallel Programming Expert Part 1: Introducing concurrency and parallelism.* Retrieved from Intel Developer Zone: https://software.intel.com/sites/default/files/m/d/4/1/d/8/09MC03_Part_1_Concurrency_par_expert_intro_2.pdf

Miller, P. (2002, May 24). *Parallel, Distributed Scripting with Python*. Retrieved from http://www.osti.gov/scitech/servlets/purl/15013331

Multiprocessing - Process-based "threading" interface. (2014, September). Retrieved from Python 2.7.8 standard library: https://docs.python.org/2/library/multiprocessing.html#module-multiprocessing

Noller, J. (2008). Getting Started with Concurrency - Using Multiprocessing and Threading. Atlanta, GA. Retrieved August 20, 2014

Noller, J. (2009, February 1). Python Threads and the Global Interpreter Lock. Retrieved from http://jessenoller.com/blog/2009/02/01/python-threads-and-the-global-interpreter-lock

Pankratius, V., Schutle, W., & Keutzer, K. (2011, January). Parallelism on the Desktop. *IEEE SOFTWARE, 28*(1), 14-16. doi:10.1109/MS.2011.8

Parallel Processing. (2014, May). Retrieved from Wikipedia, The Free Encyclopedia: http://en.wikipedia.org/w/index.php?title=Parallel_processing&oldid=610389787

Peters, A., & MacDonald, H. (2004). *Unlocking the Census with GIS.* Redlands, CA: Esri Press.

Polygon to Raster (Conversion). (2014, March). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//00120000003000000

Strohmaier, E. (2013, November). *November 2013*. Retrieved from Top 500 Supercomputer Sites: http://www.top500.org/system/178260

Supercomputer. (2014). Retrieved from Wikipedia, The Free Encyclopedia: http://en.wikipedia.org/w/index.php?title=Special:Cite&page=Supercomputer&id=628507120

Thread - Multiple threads of control. (2014, September). Retrieved from Python 2.7.8 Standard Library: https://docs.python.org/2/library/thread.html?highlight=thread%20module#module-thread

Walters, G., & Winiberg, M. (2014). *The Python Quick Syntax Reference.* New York: Apress. doi:10/1007/978-1-4302-6479-8

What is a mosaic dataset? (2014, September). (Esri) Retrieved from ArcGIS Resources - ArcGIS Help 10.2, 10.2.1, and 10.2.2: http://resources.arcgis.com/en/help/main/10.2/index.html#//009t00000037000000

Worboys, M. F., & Duckham, M. (2004). *GIS: A Computing Perspective* (2nd ed.). Boca Raton, FL: CRC Press.

Zandbergen, P. A. (2013). *Python Scripting for ArcGIS* (1st ed.). Redlands, CA: ESRI Press.

BIOGRAPHICAL SKETCH

Mr. Chen received his Master of Arts in Urban and Regional Planning from the University of Florida in December 2014. He has an educational background in urban and regional planning, which specialized in spatial analysis. Mr. Chen's research interest has been directed at the land use and transportation modeling and analyzing. His research emphasis has been concentrated on customizing and automatizing GIS (Geographic Information System) tools and applications to improve the performance of data processing in the planning field.

Mr. Chen got his bachelor's degree in urban and regional planning in Liaoning Technical University in China. Upon his completion of his master's degree, he will continue pursuing a doctoral degree in the Department of Urban and Regional Planning at the University of Florida.