교육학 석사학위논문

Massively Parallel Computing Using General Purpose Graphics Processing Units: An Application to Spatial Data Analysis

그래픽 처리 장치를 이용한 대규모 병렬 컴퓨팅:
공간데이터분석에의 적용

2013년 2월

서울대학교 대학원

사회교육과 지리전공

이 종 일

Massively Parallel Computing Using General Purpose Graphics Processing Units: An Application to Spatial Data Analysis

By
Jong I. Lee

February 2013

At the

Graduate School

Seoul National University

Massively Parallel Computing Using General Purpose Graphics Processing Units: An Application to Spatial Data Analysis

지도교수 이 상 일

이 논문을 교육학 석사학위논문으로 제출함

2013 년 2월

서울대학교 대학원

사회교육과 지리전공

이 종 일

이종일의 석사학위논문을 인준함

2013 년 2 월

위 원 장 박 병 익 (인)

부위원장 이 상 일 (인)

위 원 Douglas R. Gress (인)

Abstract

Parallel computing is a productive method to distribute a workload into reasonable pieces for a higher performance. Although the efficiency of parallel computing has been proven in various disciplines, geography has ignored the resource. The causes for this ignorance are absence of geographical problem that requires high performance computing, inadequate accessible resource to parallel computing, and absence of training and tradition. In this research, solutions for these causes are explored through an easily adoptable parallel computing on personal computers.

Although computing resources are largely implemented in spatial analysis, geographers focused on automation of the processes and developing new algorithms to result in better performance using the resources. When the focuses are combined with parallel computing, both the utilization of underemployed resources and a significantly better performance can be achieved. This thesis provides a method to utilize parallel computing resources in geography. More specifically, to integrate current trend in parallel computing, which is personal parallel computing using graphics processing units (GPU), with geography, a simple and an easily accessible method to build a massively parallel spatial analyst tool on a personal computer is suggested.

This thesis reviews several studies that implemented GPU parallelization for spatial analysis and extracts useful method to reorganize the spatial data

structure for parallel computing. Using the suggested method to build a parallel application for spatial analysis using a GPU, an environment that integrates two open sources are introduced. To test the newly developed environment, existing parallel applications with modification of spatial data restructuring for better performance are developed. The two applications are a tool for map algebra and a slope analysis tool. The map algebra application showed 8x faster improved performance. The slope analysis tool also showed a performance increase. Compare to a commercial GIS package's performance, the slope analysis result showed about 10x to 11x faster performance.

Keywords: parallel computing in geography, parallel spatial data analysis, parallel slope analysis, CUDA, GPGPU

Student Number: 2011-21548

Table of Content

List of Figures

List of Tables

I. Introduction

Near the end of the 20th century, two geographers brought up issues in geography related to a technological development, parallel computing. A study by Turton and Openshaw (1998) discusses the poor awareness of high-performance computing (HPC) in geography. The study begins by explaining what HPC is and continues by addressing the questions, ‘Why are geographers indifferent,?’ and ‘How is it applicable to geography?’ to discuss issues in geography related to parallel computing.

The study served as a non-technical introduction to parallel computing. This study also serves a similar purpose that they have attempted, but a little more technical descriptions for setting the environment used in this study is described. While their study (Turton and Openshaw, 1998) had focused on parallel computing with HPC, this study is focused on personal parallel computing using a graphics processing unit. To justify the necessity of the parallel computing, the limitations and challenges from both spatial data analysis and computation are discussed in the first two sub-sections of the introduction. The last two sub-sections of the introduction address the causes of the neglect in parallel computing in geography and the introduction of the relatively new parallel computing method.. In the body of this thesis, a method to utilize parallel computing on a graphics processing unit is provided with examples to encourage parallel computing in geography.

1. Limitations from Spatial Data

The development of digital technology and the continuous attention to its improvement have supported increases in both spatial (geographically referenced) data's quality and quantity in the past few decades (Wang, 2010; Frank, 1994). Increases in spatial data's quality or resolution, and quantity have been accompanied by the technological and methodological development in data storing and collecting capability.

The availability of more has data stimulated researchers to search for better geographical understanding of the world based on a finer resolution or larger scale. However, this explosion of spatial data and attention to them has not only provided a better environment for precise analysis, but has also left tremendous challenges to geographic information scientists.

As spatial analysis integrated with, and supported by, geographic information systems (GIS) is becoming increasingly important in scientific inquiry and decision support systems in many fields (Lee and Kim, 2012), the complexity of the spatial analysis and the volume of information that researchers desire to analyze are simultaneously increasing up to the point where conventional GIS approaches useable on personal computers cannot handle them (Wang, 2010). The trend to larger spatial data is not only led by the technological development in data acquisition, but also by the additional information derived from spatial analysis. One of the major challenges from the increase in data volume is that

methods or tools built to analyze the data are unable to respond in practical time. Turton and Openshaw (1998: 1841) have pointed out, "…there is not a single grand-challenge computational project that is explicitly geographical or in the social science domain." However, this statement is not true anymore. The next two sections discuss what have caused this challenge from two different perspectives; limited computing power due to the change in computer architecture and neglected adaptation of parallel computing and programming in geography.

2. Changes in Computer Architecture

The cause for the increase in time of spatial data analysis can be decomposed into two but interrelated reasons. First, the development of computer technology stagnated; therefore, the technology required to handle the computation intensity necessary for spatial data and spatial analysis has not been facilitated (Buchau *et al.*, 2008; McKenney *et al.*, 2011). In other words, a primary factor that decides computing power or time efficiency, central processing unit (CPU), has halted its increment in clock speed, which was roughly doubling every two year for the last two decades or more because of power limitations.

The computing ability or performance of Central Processing Unit (CPU) has advanced up to about 100 GFLOPS of computing ability on personal computers; 1 GFLOPS is 1 billion floating point operations per second. However,

for GIS, geospatial scientists and geographers who are the prime users, collectors of spatial data, and those who attempt to ambitiously use voluminous spatial data for intensive computation, the computing resource of the newly developed multicore machines is still insufficient (Hawick *et al.*, 2003) often because of the practicality in time due to the lack of parallelized software or tools. In this context, the next section discusses the neglected adaptation of parallel computing in geography.

3. Neglected Adaptation to Parallel Computing Paradigm

The second perspective of the cause for the challenge comes from geography. Spatial algorithms and GIS applications did not effectively adapt to changing trends in computer architecture while CPU's development shifted its developmental strategy from increasing clock speed to increasing the number of cores to provide more computing power. Many of the geographers who utilize GIS spent their time in data automation, rather than adapting the technological development to build more efficient analysis based on parallel computing (Healey, 1996).

From the perspective of computer development, a way to resolve the obstacle related to power limitations led to a different strategy in the development of CPUs. However, the introduction of multicore processors does not mean that the machine will divide a workload into reasonable pieces and process them

according to a given instruction. In order to effectively employ the new computer architecture, data structure must be adaptable by parallelization, and suitable modifications to algorithms are required (Xiong and Marble, 1996).

Preparing spatial data for spatial analysis entails many more complicated issues in the context of algorithm and data structure because of the following reasons: first, most of the spatial analysis or models are based on serial algorithms. This means that when an algorithm is scripted or programmed, it goes through the data one by one to compute the result. Second, data need to be restructured in a parallel manner in order to utilize spatial dependence in the data structure. Spatial dependence is one of the reasons that make spatial analysis and spatial data unique from other types of data and analysis, and its importance in geographic analysis is emphasized by stating that spatial dependence is inherent in Tobler's First Law of geography (Tobler, 1970). Because of this uniqueness of the spatial data and methods, a guideline for reorganizing the data structure is important for parallelized spatial analysis. The importance of this guideline is provided by testing the restructured data in developed analysis.

4. Era of Personal High-Performance Computing

The continuous attention from researchers who are doubly-informed from geography and computer science has expanded its focus from multicore based parallelization to massively parallel processors on programmable graphics

processing units (also known as general purpose graphics processing units) since the emergence of programmable graphics processing units (GPGPUs) such as NVIDIA's Compute Unified Device Architecture (CUDA).

The shift to massive parallelism may look like something new in geography to those not informed from computer science; however, it already has been exposed by the development of high-performance computing (HPC). As Turton and Openshaw (1998:1840) outline some of the results obtained using a supercomputer, they ask geographers "How would you do geography and think about doing your kind of geography if the workstation on your desk is between 1,000 and 5,000 times faster and bigger like HPC?" From a geographer's perspective, such a machine was not readily available, and Turton and Openshaw provide a justification for this unfairness by suggesting the effectiveness of virtual parallel machines using software.

Today, this kind of justification is not necessary because massive parallelism can be realized on the desk using a desktop or laptop with the newly developed GPGPU architecture. The performance of GPUs had been improving at a much higher rate than the performance of CPUs; in early 2003, the most advanced GPU from NVIDIA and CPU from Intel had approximately the same peak performance, and four years later a GPU vendor's most advanced GPU provided six times the performance of the most advanced CPU from Intel (Boyer *et al.*, 2008).

As noted, this study aims to provide an introduction to parallel computing, but personal parallel computing using GPGPUs for geographers. I begin with a thorough literature review on topics of spatial analysis methods that have utilized relatively newly developed technology, GPGPUs. This leads to a discussion of what is necessary in the reviewed literature to utilize the parallelization technique for geographers and geographic information scientists.

Although there is a growing body of evidence that proves the effectiveness of the integration between spatial analysis and GPGPU, the method to integrate these two fields together has largely been ignored in previous studies, and the studies that implemented parallelization do not explain the environment effectively.

To implement studied techniques, an explicit description of the resources and environments that enable replication is imperative; however, having little or no knowledge on the procedures complicates the beginning. Therefore, an examination on applicable open sources that enable accessing the conventional spatial data format and another open source that enables GPGPU's massive parallel processing on .NET environment is provided. An integrated open source environment that provides flexibility to build a raster based spatial data analysis on massively parallelism is suggested based on examples of map algebra. Furthermore, a GPGPU based slope analysis program is built upon the suggested environment with the consideration of an edge effect that a previous study by

Minhui *et al.* (2012) on parallelized slope analysis did not consider. In the described procedure of the implemented spatial analysis, a guideline for preparing spatial data for parallelization is provided. The performance observation of the developed tool is provided after the guideline.

II. Literature Review

The boom in remote sensing provides decision makers with more available geospatial data as well as more concerns about how to understand, evaluate, search, process, and utilize these overwhelming resources (Xue *et al.*, 2009). The commonly addressed concerns in literature from those overwhelming resources are mainly related to constrained time, time inefficiency of existing models, or methods in spatial analysis.

To overcome these obstacles mainly related to computing power, parallel algorithms applicable to spatial analysis are provided using multicore CPUs. Furthermore, massively parallel algorithms and methods utilizing HPC and GPGPU have been introduced. The shifting trend to this new environment of distributed computing in geography has been emphasized earlier by Openshaw (1998), and the University Consortium for Geographical Information Science (UCGIS) has also updated their research agenda in 2004 by including distributed and mobile computing (UCGIS, 2013).

Implementation of HPC based parallelization has a longer history than the GPGPU based parallelization. Turton and Openshaw (1998) observed the effectiveness of HPC, namely Cray T3D, by parallelizing the entropy-maximizing spatial interaction model. As they outline the result of the parallel implementation, they comment that the HPC awareness in geography is very poor by providing two major reviews of supercomputing usages in the United Kingdom as evidence

at the time of the research. Several reasons for this neglect have been addressed: “inadequate hardware, an emphasis on soft approach, an absence of compute intensive traditions, misplaced philosophical objections to science and computing, deficiencies in research training, and a lack of research council initiatives” (Turton and Openshaw, 1998: 1841).

Today, there are a considerable number of studies related to massively parallelized processors, especially after the debut of programmable GPUs. While it is beyond scope of this thesis to provide a comprehensive review of every GPGPU-based implementation of spatial analysis, here I discuss a few related works by implementation and data type.

1. Implementations in Visualization

In the case of South Korea, Lee and Oh (2009) have observed the computation speed of coordinates using GPU. To display coordinates of spatial data, spatial data needs to be loaded on the shared memory. When there is a user command such as panning, zooming, and orienteering, the coordinate displayed on the monitor needs to be recalculated. The study compares the performance in four different GPUs – NVIDA’s 9800GT and GTX285, ATI’s Radeon 3850 and 4850 – and on two different operating systems (OS). The performance increases up to 10 times of the CPU’s performance in the research. This method has the potential to provide a pleasant visualization environment when implemented as a

GIS package.

Similar research in the context of coordinate calculation involves a map projection computation using CUDA by Yanwei *et al.* (2011). Their method is basically same as the preceding study by Lee and Oh (2009). However, their study has involved recalculation of geographic coordinates regarding to function that they desire to transform to (in this case Universal Transverse Mercator). The research showed improved performance of between 6 to 8 times in projection computation. Transformation of projection sometimes requires a significant amount of time in cases where data contain a vast number of vertices such as road network data.

Using the rendering capability and the computational resource of massively parallel computing, Lambert *et al.* (2010) have researched the visualization of air traffic data vis-à-vis a 3-dimensional surface of the earth. Liao (2011) has implemented GPU parallelization of a volumetric terrain model. When modeling volumetric terrain, generating the voxels is critical for visualizing the solid terrain. His algorithm involves slice sweeping which divides a volume with slices and then processes each slice until all slices are processed.

All the preceding studies are versatile, and they have potential to be implemented in commercial GIS packages or conventional GIS packages. Seemingly, not many GIS package development companies are interested in them (Healey, 1996), but implementation or availability of one or more of the methods

will provide better and more appropriate environment for analysis. Therefore, it is necessary to provide possible directions that enable the integration with conventional GIS packages.

2. Implementations on Vector Data

The effort to utilize the high performance in GPU to vector analysis has been extended by McKenney *et al.* (2011). They use a brute force approach in an overlay analysis to identify intersections of two sample regions. This approach was found to be useful to a certain extent of data, but its effectiveness depreciates when the extent become much larger from the limit that computation benefits from a brute force approach.

Zhang *et al.* (2012) have added empirical evidence that demonstrates the effectiveness of the GPGPU integration with GIS by implementing the parallel spatial join operation on vector data. The method basically uses quadrats that cover the study area and divides the workload of the operation accordance to the quadrats and the performance improved more than 20 times faster computation in their application.

The limitation of parallel computing on vector based data comes from the irregular shape of vector data. Because of this, an approach like brute force is utilized to gain computational speed ups up to certain extent. If it is possible to

provide an efficient way to prepare the vector data into parallelizable structure, a lot more studies on parallelized vector data analysis are expected.

3. Implementations on Raster Data

Much scholarly work has been done on the topics of grid data analysis. Jianbo *et al.* (2010) implements Tomlin (1994)'s Map Algebra using GPU accelerated parallel algorithms. They have tested the algorithm against five different sized Digital Elevation Model (DEM) datasets that range from small to large. The result shows that the GPU accelerated parallel algorithm performed with a great advantage when the data is sufficiently large enough to overcome the computation time over the transmission time between the host memory and the video memory. Xia *et al.* (2011) provide a method to utilize GPGPU to viewshed analysis by decomposing the layer components into two, matrix traversal and ray traversal, and combines these components with the sequential (CPU based) and parallel (GPU based) methods to handle the analysis.

Ortega and Rueda (2010) implemented and tested two different algorithms detecting the drainage network on DEM data. Steinbach and Hemmerling (2012) developed a plug-in for Geographic Resources Analysis Support System (GRASS) and various implementation of spatial analysis such as neighborhood analysis on grid. Minhui *et al.* (2012) implemented CUDA based slope analysis and their results show some significant improvement in

computation time of a randomly created large grid data. However, their slope analysis exploits the GPU only although CPU and GPU have different specialties. Furthermore, their algorithm does not consider the edge effect in the analysis, and correction has been applied in this study.

As this thesis is focused on the raster data analysis utilizing GPGPUs and its implementation, recent studies account for GPGPU-based raster data analysis are examined and compared in Table 1. It is easy to observe the neglected reality of the parallel spatial analysis in geography. Although Table 1 shows a summary of only five raster based studies, only a few studies are researched by geographers. The operations they have implemented are all geographic analysis. Some may argue that these studies are too vague. Indeed, in general, the studies report the hardware specification attentively, but the software specification that informs what kind of training is required is not as attentively reported.

Software specification provided in a journal article serves as a starting point of what kind of training is required to realize a reported technique; however, many studies are indifferent to contributing a space for reporting the environment. This loosely reported software specification may have triggered the deficiency in training, as Turton and Openshaw (1998) have noted causing the impact of parallel GIS research remain insubstantial (Clematis *et al.*, 2003); therefore, it is valuable to explicitly provide an open source environment that enables the integration of spatial data handling with GPGPU utilization for anyone who is

interested in this field. By demonstrating from a simple operation to a high-level analysis that requires heavy computation in an integrated open source environment, this study seeks to erode the boundary of the GPGPU-based parallel computing from the perspective of geography and geospatial analysis.

Table 1. Recent studies utilizing GPGPU on raster data spatial analysis

Author (Year)	Author(s)' Domain	Developed Operation(s)	Software	Tested Data	Max. Data Size
Zhang *et al*. (2010)	Software Engineering	Raster sum operation	Visual Studio 2005 CUDA 2.3	Unknown DEM	2775 x 3660
Xia *et al*. (2010)	Service Engineering	Viewshed analysis	CUDA	Unknown DEM	4996 x 3088
Steinbach and Hemmerling (2012)	Computer Graphics and Ecological Informatics	Neighborhood analysis by convolution, shrink computational region, Euclidian distance transformation, focal mean	OpenCL CUDA	Unknown Dataset	4550 x 4526
Ortega and Rueda (2010)	Informatics	Drainage network analysis	OS: Linux CUDA	DEM of the South of Spain	3704 x 4425
Minhui *et al*. (2012)	Management, Science and Engineering	Slope Analysis	OS: Windows XP Visual Studio 2008	Randomly generated DEM	8192 x 8192

III. Methodology

1. Introduction to CUDA

In the recent past, parallel computing was viewed as a ‘special’ task in computer science though the perception has diminished (Sanders and Kandrot, 2010) as the necessity for multi-processing is emphasized in computer programming concurrently with the emergence of large data sets and parallelization libraries such as Open Multi-Processing (OpenMP). While accessibility of HPC resources to general researchers has conspicuously increased over the past few years (Zhang, 2010), introduction of programmable GPUs in 2006 accelerated both the shift toward the parallel paradigm and the supply of personal high-performance computing for research purpose. In short, the GPU, which had been only capable of computing the coordinates and the color to be rendered on a display device, was enabled to script massively parallel algorithms on users’ desks. This not only provides more parallelism resources to more people, but also stimulates interest in parallel computing by suggesting an efficient way to upgrade algorithm performance.

NVIDIA’s GeForce 8800 GTX was the first CUDA architecture based GPU (Sanders and Kandrot, 2010). For CUDA implementation and application, any GPU higher than 8800 GTX is imperative. A few months after the introduction of CUDA, NVIDIA provided a C programming language based

compiler and library for CUDA. This effort has resulted in many CUDA applications in various discipline areas; for example, medical imaging to scan and render ultrasonic data, fluid dynamics, and environmental science (Sanders and Kandrot, 2010). Advances in GPU computation for compute intensive-parallelizable task compare to CPU is obvious in the structure.

As shown in Figure 1, GPU has more Arithmetic and Logic Units (ALU) compare to CPU, and this structural difference can be described by comparing the main objectives of each part. CPUs contain a larger control unit to control the encompassing flow of a computer. Due to the large amount of different types of data flow into a CPU, the cache memory size of a CPU is larger than GPU's. GPUs have a higher density of ALUs that serve as the brain for compute intensive and parallel tasks such as computing the color of each pixel on the display. As the

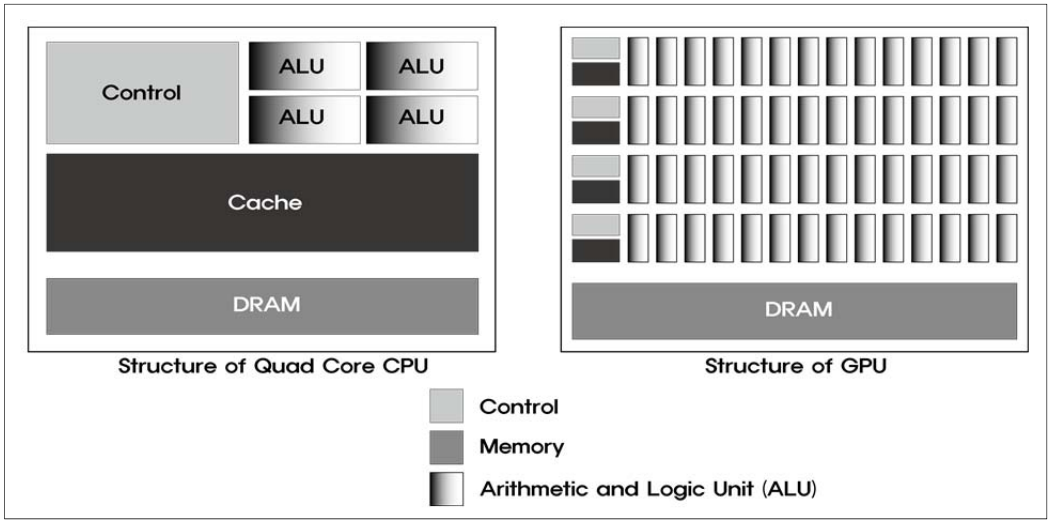

Figure 1. Structural comparison of CPU (left) and GPU (right)

size of the data that require imminent access tends to be smaller, the cache memory located under a control is smaller.

In the previous literature review, I provided a review of CUDA implemented operations in spatial analysis. Studies integrating GPGPU and spatial analysis are expected to become more capable and flexible as more technological development is ahead. For example, the Center for Manycore Programming at Seoul National University has developed a GPU-based supercomputer using multiple GPUs. Their technique provides much cheaper and higher performance that is analogues to one of the top 500 supercomputers in the world (Center for Manycore Programming, 2012). This type of technological innovation contributes to progress by providing more readily accessible parallel computing resources.

In the next section, an approach for personal high-performance computing, CUDA is introduced. A brief instruction on how to install the CUDA Software Development Kit (SDK) for massively parallel computing using GPU is described. Followed by the installation instruction, several GPU parallelization implementation cases are examined, and their performances are compared.

2. Setup for CUDA-based Spatial Analysis Development

NVIDIA's CUDA SDK supports various operating systems (OS) such as Microsoft's Windows, Linux, and OS X from Apple. The current available CUDA SDK version is 5.0. For this research, Table 2 shows the hardware and software specifications. Required files for CUDA can be downloaded at https://developer.nvidia.com/cuda-downloads for free. The package includes CUDA SDK and the latest version of a developer's driver for CUDA enabled GPUs. Installation of the kit sets up the basic settings for CUDA development and anyone who is familiar with visual C or C++ can jump into the CUDA programming with the provided guide from NVIDIA. Unfortunately, direct library from NVIDIA is not available for .NET framework based programming languages such as C# or VB.NET; however, alternatives for .NET programming with CUDA are available and it will be discussed in detail in a later section.

To utilize spatial data with the CUDA environment, there are two methods, one is using a commercial GIS package's SDK such as ESRI's ArcInfo and finding a way to import the CUDA library into it, and another is utilizing an open source library that enables spatial data handling by programming. So far, there is no study done on how to integrate the CUDA's parallel computing environment with a commercial GIS packages. A commercial GIS package, Manifold® has integrated the CUDA supporting surface analysis tool; however, it costs from 200 dollars to nearly a thousand dollars to buy the software.

It has been tried in this study to implement the CUDA library with the ESRI's ArcObject SDK using Visual Studio's C++ Microsoft Foundation Class (MFC) by importing a Dynamic Linking Library (DLL) from CUDA C using C++ (see Figure 2). However, creating a suitable DLL for ArcObject requires much complication. To minimize the complications and to provide more availability, two different libraries, CUDA.NET (CUDA.NET, 2008) and CUDAfy.NET (HybridDSP, 2011), that enable exploiting the CUDA's massively parallelism on the .NET framework environment are examined in this study. Open Source Geospatial Foundation (OSGeo) provides Geospatial Data Abstraction Library (GDAL) that is applicable on the C# environment, GDALCSharp. All these libraries can be downloaded for free and no installation is required.

Table 2. Hardware and software specification for the research

Hardware	
CPU	Intel Core i5 – 2500 @ 3.30 GHz
GPU	NVIDIA GeForce 9500GT (1GB @ 400 MHz)
Hard Disk Drive	Seagate 1TB 7200RPM (32MB buffer)
RAM	4 GB

Software	
OS	Windows 7 Enterprise K
Programming	Visual Studio 2010 C# CUDA SDK 4.2

3. Managed CUDA Libraries

This study is not the first effort to utilize an open source library in conjunction with the CUDA capability. Instead of trying a conventional GIS package, Steinbach and Hemmerling (2012) implement CUDA's parallelization into the open source GIS called GRASS (Neteler and Mitasova, 2008) using OpenCL. In their case, providing an environment for parallel computing was not the objective of the study; however, this study introduces a way to implement open source libraries for CUDA implementation and geospatial data handling on .NET framework. One of .NET framework development environment, C#, is chosen in this study.

According to the programming language popularity indices from the Transparent Language Popularity Index website and Langpop.com website, C# is one of the top ten most popular programming languages. It means that this language has the large volume of easily accessible resources for learning. As noted in the previous section, integrating the CUDA environment with a conventional GIS package is a non-trivial task. This obstacle has been addressed by Clematis *et al.* (2003) as a cause for illiteracy of parallel computing in geography - when added to the significant effort necessary for acquiring spatial data, it is impractical to put forth further effort to employ parallel techniques in

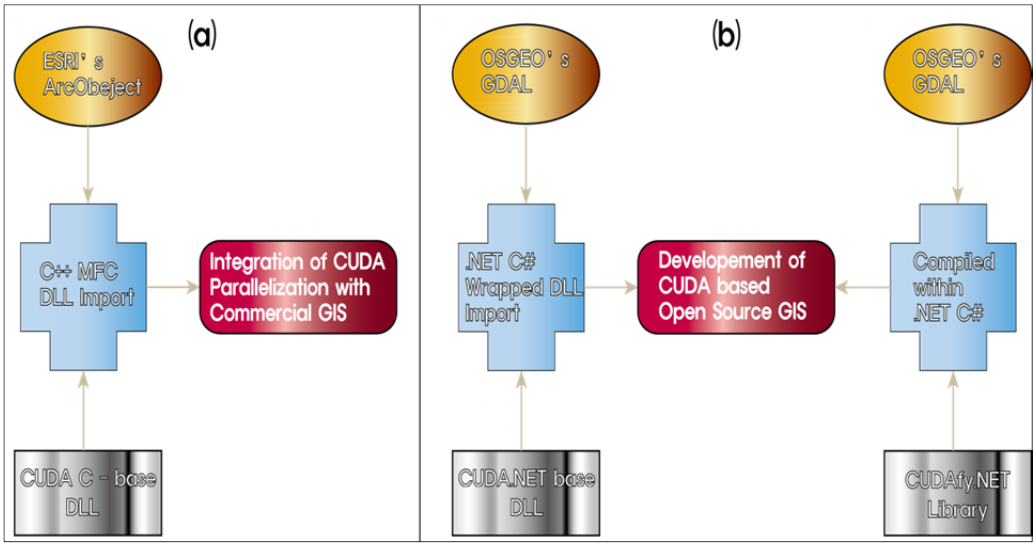

Figure 2. Approaches utilized to integrate the spatial analysis with CUDA. (a) CUDA integrated commercial GIS package extension, (b) stand-alone spatial analysis application with CUDA

the methods to analyze the acquired data. One way to overcome the obstacle causing the parallel illiteracy in geography is to provide an effective and efficient means to implement the technique; therefore, open source libraries that help to articulate the parallelization technique and spatial analysis are explored, and a successful implementation combining these two libraries is demonstrated.

McKenney *et al.* (2011) and Lee and Oh (2009) already have demonstrated the implementation of a spatial analysis and a geographic visualization method using CUDA C environment, but NVIDIA does not provide a .NET framework based SDK; therefore, two managed CUDA libraries applicable against .NET framework is tested to integrate with the open source library for spatial data,

```
1: extern "C" __global__ void IncrementArrayOnDevice(float *a, int N);
2: __global__ void IncrementArrayOnDevice(float *a, int N)
3: {
4:          int idx = blockIdx.x * blockDim.x + threadIdx.x;
5:          if (idx < N) a[idx] = a[idx] + N;
6: }
```

Figure 3. Example of .cu file. The function 'IncrementArrayOnDevice' increases the value allocated in the device memory by N.

OSGeo's GDAL.

To employ the CUDA.NET's library, a CUDA C based linking is required to be exploited on C#. A .cubin file is analogues of DLL and writing the script can be done in a simple text editor by adding the .cu extension instead. A written script can be compiled in the Windows command line using the 'nvcc' compiler provided by the NVIDIA, but the 'nvcc' compiler needs access to the C compiler, and it can be done by adding the location of the cl.exe (C compiler) as an environment variable on the system property in case of Windows. In case of using 64 bit versions of the Windows with Visual Studio 10, variable 'Path' can be edited under the system variable as add a semi colon and 'C:₩Program Files (x86)₩Microsoft Visual Studio 10.0₩Common7₩IDE₩' at the end of the existing variable. For the x 86 versions of the Windows, the parenthesized part is not necessary.

Figure 3 is an example of .cu file increasing floating point numbers given from the machine (computer) by N. When compile with 'nvcc' compiler, the same

filename with the extension of .cubin will be created. The compiling command can be run on the command line of Windows by changing to the directory where the .cu file is located. The use of 'nvcc' when the environment variable for C compiler is correctly set is 'nvcc filename.cu –cubin –arch=sm_XX' where XX is a place to insert the compute capability of the GPU.

A tedious obstacle with the CUDA.NET is that it requires a significant effort to set up the environment and compile the CUDA functions to be called in the C#. This obstacle can be resolved by using CUDAfy.NET. Unlike the CUDA.NET, CUDAfy.NET does not require separate compiler to implement CUDA script. The CUDA part can be written in C# by defining the CUDA functional area by declaring [Cudafy]. CUDAfy.NET library only requires its components to be contained in the bin folder of the project directory.

Before getting into using both libraries at once, a simple sample that shows the potential of the CUDA parallel computing is demonstrated by computing the buffer of given vertices. According to ESRI's technical description of shapefile, a polygon consists of the extent, number of parts, total number of points, index to first point in part, and points for all parts. In this demonstration, the number of 180 coordinates (2 degrees interval) that forms a circle around the given vertices are calculated to compute the points for all parts in a shapefile. For a given vertex that has a coordinate of x and y, coordinates for x_n and y_n by a given degree r_n is computed as $x_n = x + d \times \sin(r_n \times \pi/180)$ and $y_n = y + d \times \cos(r_n \times \pi/180)$,

respectively. *n* is defined by dividing 360° by the interval (for this case 2 degree interval).

Figure 4 illustrates the flow chart of the CPU model and GPU model. Six array variables are defined in the beginning to contain the randomly created *x* and *y* coordinates for vertices, computed coordinates, and accumulated result. Randomly created numbers that represent the coordinates are created and stored in the arrays. In GPU model, two more steps are required before get into the computation procedure, allocation of GPU memory and copying the values to compute into the allocated memory (see Figure 4's procedures in the dotted box). The major difference in these models is the box with the thick outline. In the CPU model, *x* and *y* coordinates for a vertex is called from the array filled with random numbers. The coordinates are computed for every 2 degree interval and stored in an array that contains the result. When the computation for a vertex is done, it goes back to call another coordinates for a vertex, and then loops the same procedure until every vertex has the result (see the shaded shape in Figure 4). On the other hand, GPU model calls in all numbers at once and computes the coordinates at once. For 10 vertices, 1,800 computations are done for each *x* and *y*

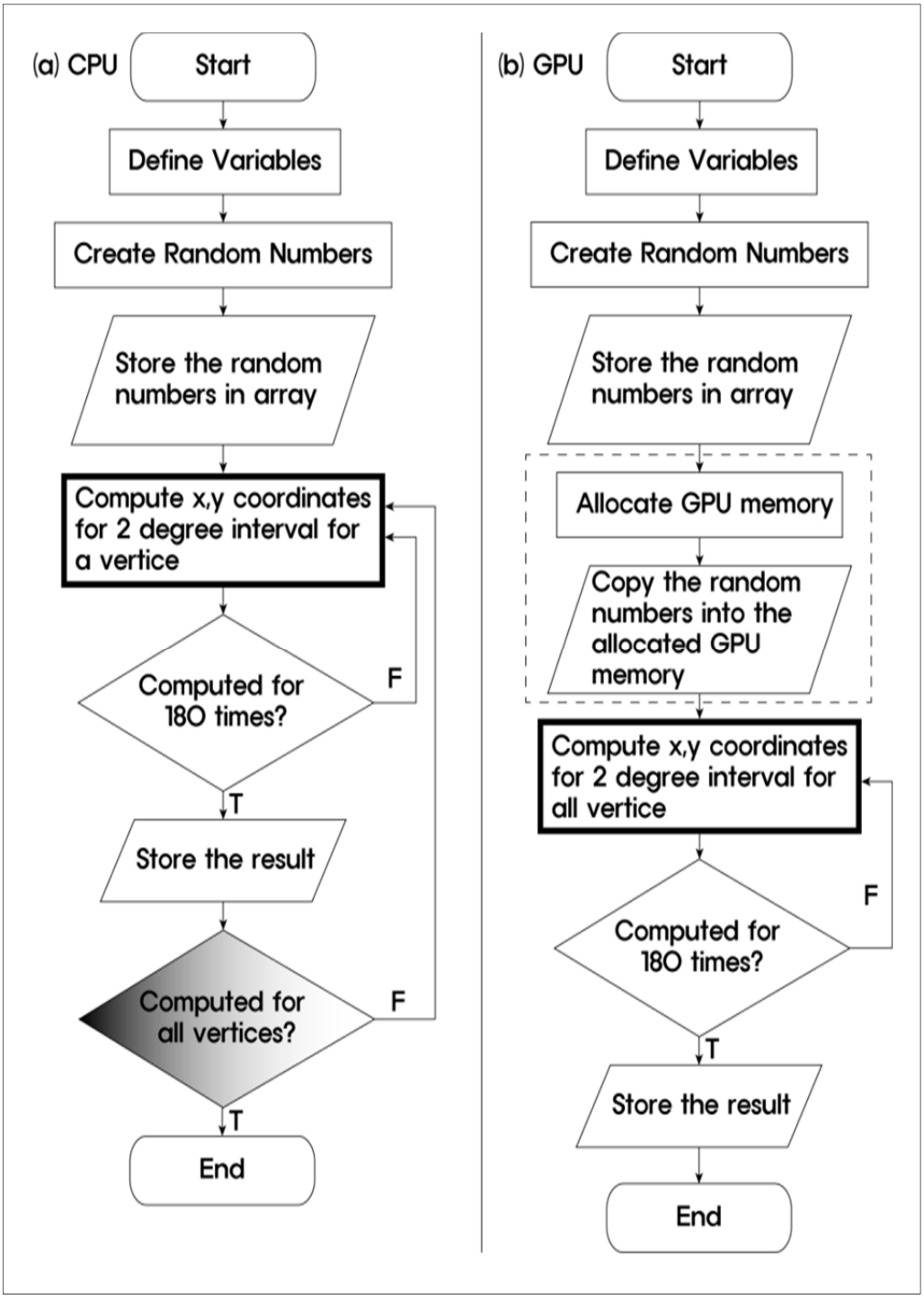

Figure 4. Flowchart comparison of (a) CPU model and (b) GPU model

coordinates in this test; therefore, it results the total of 3,600 computations. The test is done for varying number of vertices to see compare the computation time required for each model.

For each number of vertices, test was conducted for ten times and the averaged time of the test is reported in Figure 5. It shows that the computation time did not gain any advantages from parallelization before the number of vertices reached to 10,000; the GPU model took more time than the CPU model up to this point. From the 10,000 point, the computation time starts showing discrepancy. When the number of vertices to compute reached over 10,000, the GPU model than the CPU model and the advantage continues on. The slower timing of GPU model before 10,000 is caused by the Figure 4 (b)'s 5th step, where

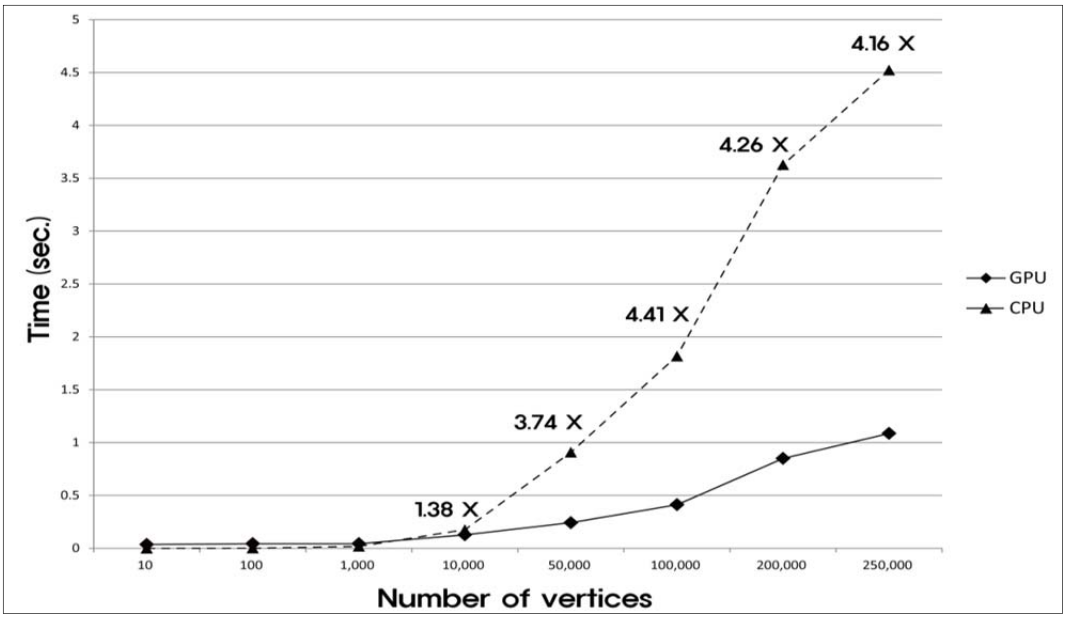

Figure 5. Computing time for buffer points' coordinates comparison between CPU and GPU

the values are copied from computer's memory to GPU memory. In other words, the gain from the parallelization starts when the computation time compensates the loss from the transferring the value.

The buffer test was a simple demonstration to show the potential of parallelization in spatial operation; however, it was not a complete spatial operation because it only considered one element of a standard geospatial data (in this case location of all points) and the result was not stored in an extensible format applicable in other GIS packages for further analysis. GDAL is used in this study as the library that enables the extensibility.

4. Integration of Managed CUDA with GDAL for Raster Data Analysis

Integrating two or more libraries often accompanies complexity caused by the compatibility among the libraries. Each library has required environment settings to work properly; however, when there is a difference in the settings, one of the libraries does not function properly. This study set the framework as .NET framework 4 Client Profile and the implementations are developed as a Windows form application.

When a project is created, adding reference to Cudafy.NET.dll and gdal_csharp.dll, and adding the namespaces (see Figure 6 line 1 to 4) completes the preparation. To bring a dataset for analysis, GDAL needs initiation, and the

```
1:using OSGeo.GDAL;
2:using Cudafy;
3:using Cudafy.Host;
4:using Cudafy.Translator;
5:OSGeo.GDAL.Gdal.AllRegister();
6: CudafyModule km = CudafyTranslator.Cudafy(ePlatform.x86,
eArchitecture.sm_11, new Version(4, 0), true, typeof(Form1));
```

Figure 6. Namespaces and initialization required for GDAL and CUDAfy.NET libraries

method to initiate is in line 5 of Figure 6. CUDAfy also requires initiation by declaring the module as line 6 of Figure 6. Declaration of CUDAfy module requires the specification of platform, identification of compute capability of the device, version of CUDA SDK, compiling status, and type of class the module belongs to. Some of the specifications are already noted in the introduction of 'nvcc' compiler.

According to the example shown in line 6 of Figure 6, the initiation specification tells the module to be compiled for the environment of 32bit platform, compute capability of 1.1, CUDA SDK version of 4.0, compile on execution, and the name of the class is Form1 in this case. These specifications are subject to change regarding to the GPU device that a user have.

IV. Implementation of CUDA Integrated Spatial Analysis

To examine the implementation of the discussed methods, map algebra's arithmetic functionality, slope analysis is developed with .NET based CUDAfy and GDAL libraries. Map algebra was first introduced in the late 1970s and its analytical techniques have been widely adopted and incorporated in many GIS and remote sensing image processing packages because of its simple syntax and the ability to combine multiple functions to create more complex models (Mennis *et al.*, 2005). It classifies all operations of rasters into four basic classes; local – examines rasters cell by cell, focal – compares the value in each cell with the neighboring cells, global – produces a value that encompasses the entire layer, zonal – the computing target becomes the zone of contiguous cells that share the same value (Longley *et al.*, 2005). In this implementation, local operations are scripted and the performance is compared with CPU results.

Implementing map algebra in GPGPU has already been done by Jianbo *et al.* (2010); however, since the CUDAfy and GDAL integration environment has not been used at all, easily scriptable algorithm like map algebra's local operation is suitable to test the capability.

Slope analysis is a terrain analysis that involves heavy computation; therefore, implementation of slope analysis in this environment can be considered as an extension of seeing the promise of the integrated environment. In the

performance result of the slope analysis, the elements needed to consider when reporting the computing performance is also discussed.

1. Map Algebra's Arithmetic

The approach taken to implement local arithmetic operations of map algebra is similar to the buffer demonstration. A bit more complication appears in this implementation is the use of GDAL library to utilize an actual raster dataset to read the values from a raster dataset and return a resulting raster layer in a compatible format with commercial GIS packages.

The implementation is divided into three stages described in Table 3. The preprocessing stage consists of accessing to a raster dataset and identifying the general property of the dataset to establish a space for resulting dataset. Stage 2 involves reading the data from each cell to store every cell values in an array. This stage is inevitable since GPU applies a single function to every element stored in the GPU memory (Lee and Oh, 2009; Zhang *et al.*, 2012). This capability of GPU is called single instruction, multiple data (SIMD) design. In stage 3, the first step is allocating a space in GPU to copy the values stored in an array, and then the function declared in CUDAfy area is launched for each value in the allocated memory. When the computation is done, the result is copied back to the CPU memory to write the result back into the prepared raster dataset space in Stage 1's step 6.

Table 3. Procedure for map algebra parallelization

	Stage 1. Preprocessing
1	Open raster data to be analyzed from disk
2	Read number of columns and rows and store
3	Read geographic transformation information and store
4	Read projection information and store
5	Specify the data format of the resulting layer
6	Declare space to store the resulting layer using the size obtained in step 2
7	Copy the geographic transformation and projection information to the resulting layer
	Stage 2. Iteration to read value from each cell
1	Read the band and store it as variable
2	For each row x
3	For each cell in column y (x,y)
4	Read the value and store them in an array and location of each cell is defined by [x + y * number of columns]
5	End For
6	End For
	Stage 3. CUDA computation
1	Allocate GPU memory
2	Copy the array to GPU memory
3	Launch CUDA and compute the arithmetic function
4	Copy the result GPU memory back to array
5	Fill in the resulting layer created in step 6 with the array values by utilizing the iteration used in Stage 2

Jianbo *et al.* (2010) have also implemented CUDA based local map algebraic functions earlier; however, this implementation revealed that the new environment suggested by this study has the same ability to implement the functionality used in their research, but this research tested against the larger datasets; maximum of 5,406 x 5,406. Although their implementation shows about

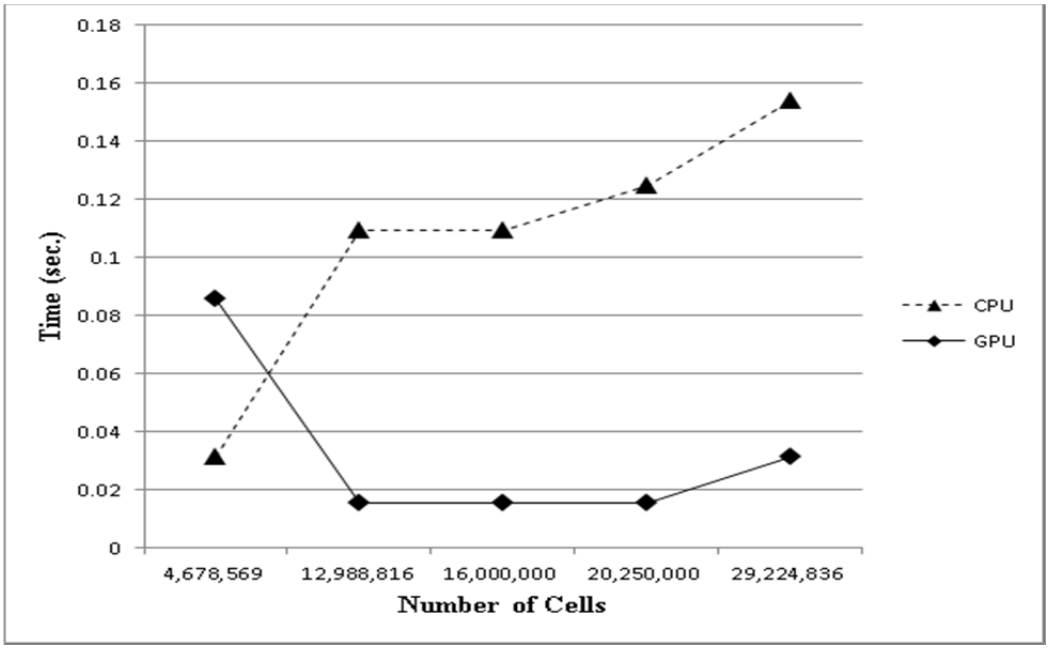

Figure 7. GPU and CPU's comparison of local SUM operation

3x speed up at the maximum data size, this implementation have shown 8x speed up using a GeoTIFF (Tagged Image File Format) format. The result is represented in Figure 7. As the number of cells in a raster dataset reaches to close to twelve hundred million, the performance gains from GPU are more obvious.

2. Slope Analysis

Terrain analysis based on DEM consists of several analyses such as aspect, direction of steepest downhill descent, irradiance, watershed, and viewshed. The result of terrain analysis acts as crucial factor for decision making such as emergency management, location-allocation, environmental evaluation, and so on. This part demonstrates a GPGPU based parallel implementation of slope analysis.

The importance of parallelization of slope analysis is that it requires heavy computation; thus, it causes inefficiency of the whole terrain analysis (Minhui *et al.*, 2012). As the size of spatial data has risen, a tool enabled dealing with a larger scale slope analysis in practical time is crucial.

Slope is derived from the maximum rate of change in elevation from a cell to its neighboring cells (Lloyd, 2009). Conceptually, slope computation is finding a fitting plane regarding to the 3 by 3 adjacent cells (see Figure 8). There are a variety of slope computation methods, and the average maximum technique is selected for slope analysis parallelization because it is one of the most popular (Lloyd, 2009) and generally known to have the best precision among slope computation methods (Minhui *et al.*, 2012). The computed result is easy to compare and verify because the spatial analyst extension of ESRI's ArcMap is also based on the same technique.

The slope of a cell is determined by computing the rate of change in horizontal and vertical directions. The horizontal rate of change of a given cell, *H*, is computed with the following equation:

$$H = \frac{(z_3 + 2z_5 + z_8) - (z_1 + 2z_4 + z_6)}{8d_x} \quad (1)$$

where, d_x is the horizontal distance of a cell or horizontal cell size and z_n are values of neighboring cells (see Figure 8). Similarly, the vertical rate of change of a given cell, *V*, is computed with the following equation:

$$V = \frac{(z_6 + 2z_7 + z_8) - (z_1 + 2z_2 + z_3)}{8d_y} \quad (2)$$

where, d_y is the vertical distance of a cell or vertical cell size. To transform the slope in a common measure, degrees, using the computed *H* and *V*, the following equation can be used:

$$\text{Slope } = \tan^{-1}\sqrt{H^2 + V^2} \times 1\, radian \quad (3)$$

where, 1 radian is approximately 57.29578.

As noted in the previous section, the GPGPU structure is SIMD, therefore, all neighbors and the value of original raster dataset must present simultaneously in the GPU memory at the time of GPU computation. Minhui *et al.* (2012) defines the neighboring cells after moving all values to the GPU memory. This may have some advances in a sense that it fully utilizes the GPU; however, as GPU is specialized in floating point computation, the gain from the GPU utilization while the data needs to define the neighbors may not so obvious; therefore, this study have chosen to define the neighbors in CPU because restructuring the data to define neighbors does not require heavy computation.

The neighbors are defined with the following method:

1. Store a DEM's z values as an array.
2. Make an array to store z_1 for each z_0. (Ex. When z is at (x,y), z_1 is placed at (x-1, y-1) for every cell.

3. Make an array to store z_2 for each z_0. (Ex. When z is at (x,y), z_2 is placed at (x, y-1) for every cell.

4. Make an array to store z_3 for each z_0. (Ex. When z is at (x,y), z_3 is placed at (x+1, y-1) for every cell.

5. Define other neighbors (z_4, z_5, z_6, z_7, and z_8) in this way and set exceptions for the ones that are under or over the minimum or maximum of the array index size, respectively.

Restructuring the dataset into eight different corresponding neighbors before copying into the GPU memory not only provides easy computing function to be implemented in the GPGPU but also means letting the part do what it is specialized for. CPU is specialized for indexing and restructuring involves more of indexing than computation; therefore, restructuring part is assigned to CPU and the time it takes to define the neighbors according to the steps suggested above is a tenuous part.

One crucial effect that terrain analysis involved with neighboring cells of a given cell is how to consider neighbors of the cells located at the edges. Unless the algorithm is carefully designed considering the effect, cells at the border have fewer neighbors, thus it leads to unreliable values at the edges. As it can be examined in Figure 9, the borders at (a) shows significantly high values compare

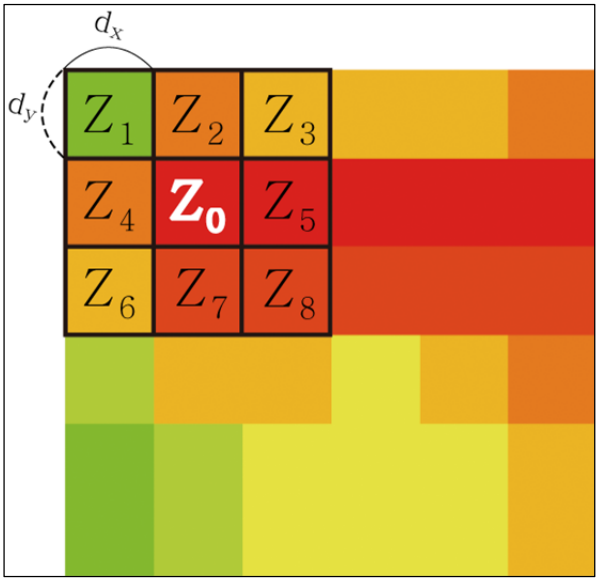

Figure 8. 3 x 3 moving window used to compute the slope. Slope at Z_0 is computed by considering neighbors to estimate the rate of change.

to the borders at (b). The rate of change is exaggerated as the missing neighbors for the edges are considered as zero.

There are several ways to settle this problem. The first is running it as it is and then throwing away the borders considering them as unreliable; however, when the size of a raster dataset is too small, this may result in removing much of the raster dataset. Another way to fix this problem is giving weights to cells to compensate the neighbors with no values; however, it is difficult to consider the appropriate weights to impose. Declaring the bottom edge as the neighbor of the top edge is another commonly used way. In this implementation, the edges are considers as having another of themselves where there is no neighbors.

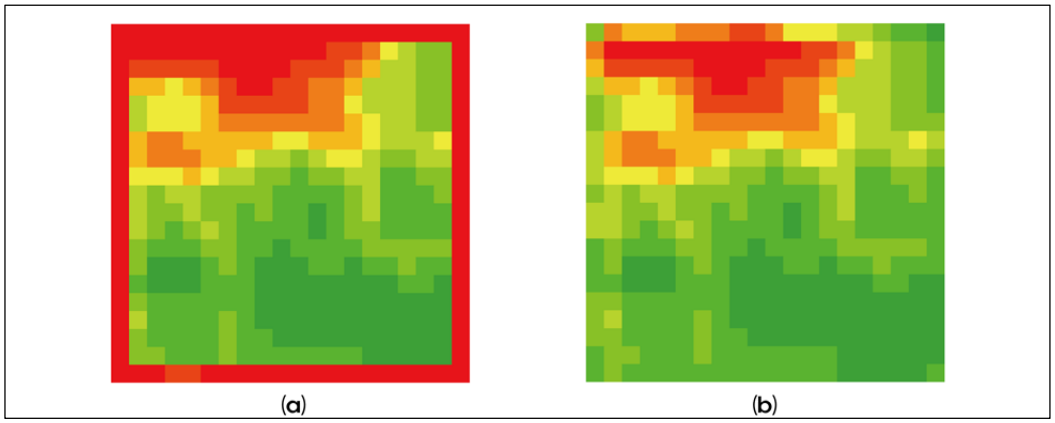

Figure 9. (a) slope analysis without edge effect fix, (b) slope analysis with edge effect correction

To declare the top, bottom, right, and left edges as neighbors of themselves, they are stored in separate arrays and copied back into appropriate places in the process of restructuring. Of the 8 neighbor arrays to be corrected, neighbors declared as four corners involve two pre-stored correction arrays and the other four sides involve only one pre-stored correction array for each.

The parallel model for slope analysis with edge effect correction is implemented with the procedure described in Table 4. In Stage 1, what has been changed compare to map algebra is step 8 and 9 are added to declare variables to store neighborhoods and variables necessary for edge effect correction. The result of edge effect is shown in Figure 9 (a).

Stage 2 is basically the same as the map algebra's procedure. Stage 3 involves defining the neighborhoods by shifting the row and columns accordingly.

```
[Cudafy]
1: public static void Slope(GThread thread, float[] a, float[] b, float[] c,
   float[] d, float[] f, float[] g, float[] h, float[] i, float[] result,
   float cellSizex, float cellSizey)
2:        {
3:            int tid = thread.threadIdx.x + thread.blockIdx.x *
              thread.blockDim.x;
4:            result[tid] = GMath.Atan(GMath.Sqrt(GMath.Pow(((((c[tid] + 2 *
              f[tid] + i[tid]) - (a[tid] + 2 * d[tid] + g[tid])) / (8f *
              cellSizex))), 2) + GMath.Pow(((((g[tid] + 2 * h[tid] + i[tid]) -
              (a[tid] + 2 * b[tid] + c[tid])) / (8f * cellSizey))), 2))) *
              57.29578f;
5:         }
```

Figure 10. Declaration of CUDAfy slope function

As the procedure shifts row or columns, it detects whether a corresponding value exists or not. If no input was made, it fills in the corresponding pre-stored edge value to the empty space. This iteration is done until all eight neighbors are defined. The stored variables are copied in the allocated memory of the CPU and the computation is processed according to the defined function in CUDAfy area.

Figure 10 shows the exact script implemented in the CUDAfy area. Table 4's stage 4 – step 3 processes this function to compile and compute the data. The index of each data are declared using the number of threads and blocks in a GPU and assigned to thread index (tid) in line 3. In line 4, the arctangent, square root, and power functions are used from GMath library because this area is an independent area compiled by CUDAfy compiler. The single function instructed in line four with the 'tid' indexing is conceptually same as putting many three by three windows for thousands of cells at once using every idle resource in the GPU.

Table 4. Procedure for slope analysis parallelization

Stage 1. Preprocessing	
1	Open raster data to be analyzed from disk
2	Read the number of columns and rows and store
3	Read the geographic transformation information and store
4	Read the projection information and store
5	Specify the data format of the resulting layer
6	Declare a space to store the resulting layer using the size obtained in step 2
7	Copy the geographic transformation and projection information to the resulting layer
8	Declare variables to store top, bottom, left, and right edges
Stage 2. Iteration to read value from each cell	
1	Read the band and store it as variable
2	For each row x
3	For each cell in column y (x,y)
4	Read the value and store them in an array, and location of each cell is defined by [x + y * number of columns]
5	End For
6	End For
Stage 3. Iteration to define neighborhoods and edge effect correction	
1	For each row x
2	For each cell in column y(x,y)
3	Read the value and store them in an array, and location of each cell is defined by (x-1,y-1) for the case of z_1
4	If (value at x,0 is zero)
5	Fill in the pre-stored top side
6	End If
7	If (value at 0,y is zero)
8	Fill in the pre-stored right side
9	End If
10	End For
11	End For
12	Iterate to define other neighbors
Stage 4. CUDA computation	
1	Allocate GPU memory
2	Copy the arrays to GPU memory
3	Launch CUDA and compute the arithmetic function
4	Copy the result GPU memory back to array
5	Fill in the resulting layer created in step 6 with the computed result

3. Result of the Parallel Slope Analysis

The implemented script tested its performance to seven different datasets in TIFF format. In this test, another model of GPU device, GTS450, is installed to compare the performance of different GPU models. The result and the comparison with CPU computed slope analysis is presented in Figure 11. For each raster dataset, the processing time has been measured for ten times and the average is taken in the resulting figure.

As it has been noted and other GPGPU based implementations addressed, the performance advantage is not obvious for a small size dataset; however, when the number of cell reaches to 2.7 million, the speed ups shown in Figure 11 are comparison between the GTS 450 device and the CPU. The performance difference between GTS 450 and 9500 GT is similar for the smaller datasets and it diverges to about 1.3x to 1.7x as the size of the dataset gets larger.

The average performance gain using the GPGPU slope analysis shows that it computes about 7.6 times faster than the CPU and the maximum speed up is 12.22 times. Previous study on parallelization of slope analysis by Minhui *et al.* (2012) has shown about the maximum of four times of speed up with Quadro FX 1800 GPU device, which has been developed for professional parallel computing. Their device is a model specifically designed for professional parallel computing;

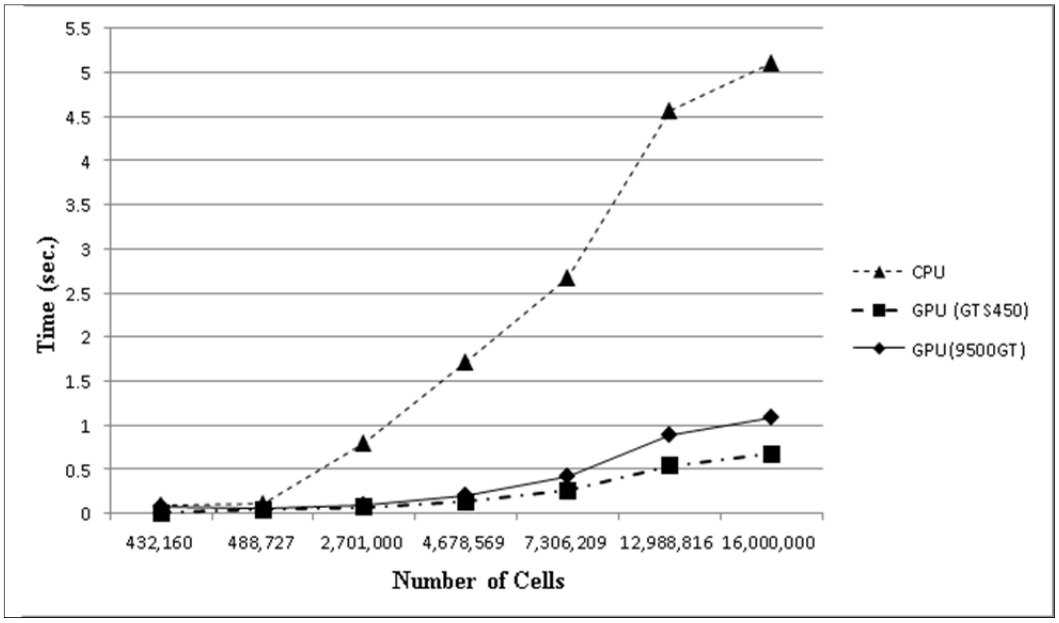

Figure 10. Performance comparison of GPUs and CPU. Speed ups (x) are comparison between GTS450 and CPU)

however, the two devices tested in this study are gaming GPUs. One pitfall that has to be considered from the result is that the performance increase depreciates after it hits the maximum. The time provided in Figure 11 is the measure of time from the beginning of the computation to the result written on the disk; therefore, the depreciation of the performance for larger data is imperative since accessing disk to write the data takes the most time in general.

Many studies on GPU provide the improved performance by separating the disk access time from the processing time or just the processing time like shown in Figure 12. However, spatial data usually involve large size data and the disk

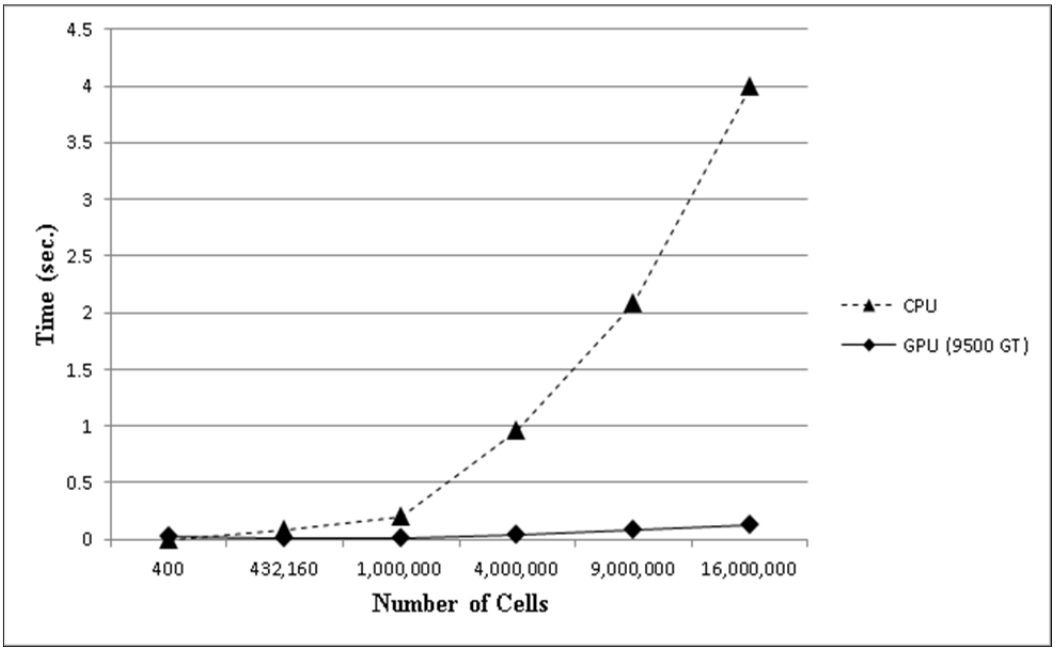

Figure 11. Processing time comparison of GPU and CPU

access time takes a considerable part in many spatial analysis methods. The time it takes to write the result back to the disk needs to be included because it is an important variable to optimize the efficiency of algorithms. The implementation in the new environment has achieved up to 30 times of speed up when only the processing time is considered as shown in Figure 12.

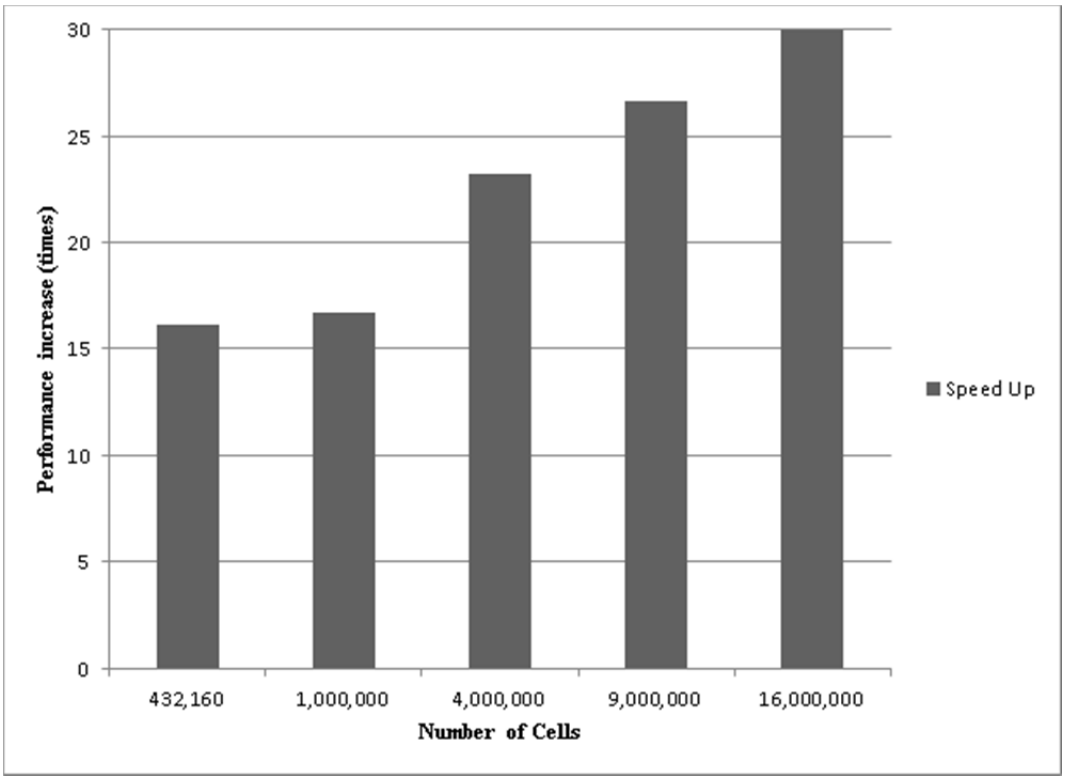

Figure 12 Processing time speed up result

V. Conclusion

This study has addressed the issues of parallel computing in geography in the past, and by reviewing a number of studies, it has proven that the interest in parallel computing today is not very enthusiastic in geography. Although the algorithms and methods that have been implemented in the studies are geographical, most of the researchers were from the field of computer engineering or science. Several reasons for this low valued perception in parallel computing from geography are discussed in studies. Of those, lack of training and limited access to the HPC resources is focused. To provide a solution to these obstacles, explicitly reviewed guide for massively parallel computing using GPGPU has been described. By suggesting a way to turn into the desktop computer sitting on the desk as a personal parallel computing machine, this study has attempted to add resources for parallel computing.

To provide an environment for training, an integrated environment of CUDA and GDAL is suggested. Especially, CUDA environment is acquired from an open source library based on .NET framework, CUDAfy. Utilizing .NET framework to develop a spatial analysis tool provides a better compatibility with platforms, and there are a lot of free learning resources available for this developing tool. To demonstrate the promise of the new environment, map algebra is implemented, and the comparison of performance between CPU and GPU is examined. For more compute intensive analysis, a slope analysis tool based on DEM is

implemented, and the method to fix the edge effect has been employed in the restructuring process. Although the slope analysis has been implementation by Minhui *et al.* (2012), their result showed about four times of speed up and the study has not dealt with the edge effect.

In summary, this study have provided a method to face the challenges from the voluminous spatial data and HPC illiteracy in geography by providing a .NET framework based spatial analysis environment for massively parallel computing using open sources. A method to restructure the spatial data to define neighborhoods with edge effect correction has been described and implemented in slope analysis to validate the capability.

The limitation in the implemented slope analysis is that it has not been enabled to display the result; therefore, the result needs to be examined with a conventional GIS package. Some specifications required for launching CUDAfy modules are not auto detected. It means that the script where it declares the module needs modification according to the GPU device installed on the computer. Although the result of the largest raster dataset is not included, 4,300 x 4,300 sized raster dataset has been tested on the machine described in Table 2, and any larger dataset could not be handles due to the out of memory exception. In order to settle this problem, like Jianbo *et al.* (2010) have utilized the batch processing, the implementation has been modified to process in batch mode. The size of 10,812 by 10,812 raster dataset (approximately equal to 117 million cells) was

divided into nine tiles with the size of 3,604 by 3,604 to test the batch processing. The time it took to compute slope was 2.99 seconds with the GPU and 34.05 seconds with the CPU.

As Openshaw and Turton (1999) have concerned, many may question: Why bother with parallel computing in geography? New computers come out nowadays are installed with multicore CPUs, but there are not many spatial analysis tools that utilize the multicore architecture. To gain more efficiency, the effort to search for an algorithmic modification or a new algorithm is going on. At the same time, the volume of spatial data keeps enlarging. Searching for new algorithms is a good answer for this problem, and parallelizing the existing algorithm or model is another good answer, but a better answer will be discovered when these two good answers converge. Without these efforts, it is likely to face the future with GIS packages that we have little or no control over (Openshaw and Turton, 1999).

GPU-based massively parallel spatial analysis is not just about speed. The benefits this system offers are interconnected. First, the speed saves time. As with the saved time, the effectiveness in transformation of information to wisdom is obvious outcome. It can also save energy by shortening the time leaving the computers on; thus, it contributes to the sustainable development. All these benefits can begin from a geographer’s desk affordably by developing parallel algorithms and tools integrated with and supported by GIS that can efficiently

handle computationally heavy tasks such as spatial interaction data analysis, spatio-temporal data analysis, environmental or statistical modeling in global scale (Healey, 1996) and so on, to name but a few examples.

References

Boyer, M., Skadron, K., and Weimer, W., 2008, Automated dynamic analysis of CUDA programs, *Third Workshop on Software Tools for Multicore Systems*, April 6, Boston, MA.

Buchau, A., Tsafak, S. M., Hafla, W., and Rucker, W. M., 2008, Parallelization of a Fast Multipole Boundary Element Method with Cluster OpenMP, *IEEE Transactions on Magnetics*, 44(6)**,** 1338-1341.

Clematis, A., Mineter, M., and Marciano, R., 2003, Guest editorial: high performance computing with geographical data, *Parallel Computing*, 29(10)**,** 1275-1279.

Frank, S., 1994, Cataloging digital geographic data in the information infrastructure: A literature and technology review, *Information Processing and Management*, 30(5)**,** 587-606.

Hawick, K. A., Coddington, P. D., and James, H. A., 2003, Distributed frameworks and parallel algorithms for processing large-scale geographic data, *Parallel Computing*, 29(10)**,** 1297-1333.

Healey, R. G., 1996, Special issue on parallel processing in GIS, *International Journal of Geographical Information Systems*, 10(6)**,** 667-668.

Mennis, J., Viger, R., and Tomlin, C. D., 2005, Cubic map algebra functions for spatio-temporal analysis, *Cartography and Geographic Information Science*, 32(1)**,** 17-32.

Jianbo, Z., Wenxin, Y., Jing, S., and Yonghong, L., 2010, GPU-accelerated parallel algorithms for map algebra, *International Conference on Environmental Science and Information Application Technology,* July 17-18, Wuhan, China, 882-885.

Lambert, A., Bourqui, R., and Auber, D., 2010, 3D edge bundling for geographical

data visualization, *Information Visualisation (IV), 2010 14th International Conference*, July 26-29, London, U.K., 329-335.

Lee, J.-I. and Oh, B.-W., 2009, An efficient technique for processing of spatial data using GPU, *The Journal of GIS Association of Korea,* 17(3)**,** 371-379.

Lee, S.-I. and Kim, K., 2012, Geospatial analysis and modeling in Korea: a literature review, *Journal of the Korean Geographical Society,* 47(4)**,** 606-624.

Liao, D., 2011, GPU-based fast volumetric terrain modeling for volumetric GIS, *Proceedings of the 2nd International Conference on Computing for Geospatial Research and Applications*, May 23-25, Washington DC, Article No. 28.

Lloyd, C. D., 2009, *Spatial Data Analysis: An Introduction for GIS users*, 1st edition, Italy: Oxford University Press.

Longley, P., Goodchild, M., Maguire, D., and Rhind, D., 2005, *Geographic Information Systems and Science*, 2nd edition, Hoboken, NJ: John Wiley & Sons.

McKenney, M., Luna, G. D., Hill, S., and Lowell, L., 2011, Geospatial overlay computation on the GPU, *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, November 1-4, Chicago, IL, 473-476.

Minhui, L., Wei, X., and Lei, C., 2012, A GPU-based parallel processing method for slope analysis in Geographic computation. *Advanced Materials Research,* 538-541.

Neteler, M.and Mitasova, H., 2008, *Open source GIS : a GRASS GIS approach*, New York: Springer.

Openshaw, S., 1998, Some trends and future perspectives for spatial analysis in

GIS, *Geographic Information Sciences,* 4(1-2)**,** 5-13.

Openshaw, S. and Turton, I., 1999, *High Performance Computing and the Art of Parallel Programming: An Introduction for Geographers, Social Scientists, and Engineers*, New York, NY: Routledge.

Ortega, L. and Rueda, A., 2010, Parallel drainage network computation on CUDA, *Computers & Geosciences,* 36(2)**,** 171-178.

Sanders, J. and Kandrot, E., 2010, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Boston, MA: Addison-Wesley Professional.

Steinbach, M. and Hemmerling, R., 2012, Accelerating batch processing of spatial raster analysis using GPU, *Computers & Geosciences,* 45**,** 212-220.

Tobler, W. R., 1970, A computer movie simulating urban growth in the detroit region. *Economic Geography,* 46**,** 234-240.

Tomlin, C. D., 1994, Map algebra: one perspective, *Landscape and Urban Planning,* 30(1-2)**,** 3-12.

Turton, I. and Openshaw, S., 1998, High-performance computing and geography: developments, issues, and case studies, *Environment and Planning A,* 30(10), 1839-1856.

Wang, S., 2010, A CyberGIS framework for the synthesis of cyberinfrastructure, GIS, and spatial analysis, *Annals of the Association of American Geographers*, 100(3)**,** 535-557.

Xia, Y.-j., Kuang, L., and Li, X.-m., 2011, Accelerating geospatial analysis on GPUs using CUDA, *Journal of Zhejiang University SCIENCE C,* 12(12)**,** 990-999.

Xiong, D. and Marble, D. F., 1996, Strategies for real-time spatial analysis using massively parallel SIMD computers: an application to urban traffic flow

analysis, *International Journal of Geographical Information Systems*, 10(6)**,** 769-789.

Xue, Y., Hoffman, F., and Liu, D., 2009, GeoComputation 2009, *Computational Science – ICCS 2009*, November 30-December 2, Sydney, Australia, 345-348.

Yanwei, Z., Zhenlin, C., Hui, D., Jinyun, F., and Liang, L., 2011, Fast map projection on CUDA, *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*, July 24-29, Vancouver, Canada, 4066-4069.

Zhang, J., 2010, Towards personal high-performance geospatial computing (HPC-G): perspectives and a case study, *Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems*, November 3-5, San Jose, CA, 3-10.

Zhang, J., You, S. and Gruenwald, L., 2012, High-performance spatial join processing on GPGPUs with Applications to Large-Scale Taxi Trip Data, Technical report online at http://geoteci.engr.ccny.cuny.edu/pub/nnsp_tr.pdf , City University of New York, New York.

Center for Manycore Programming at Seoul National University, 2012, http://aces.snu.ac.kr/Center_for_Manycore_Programming/Home.html (Accessed on January 2, 2013).

CUDA.NET, 2008, http://www.cass-hpc.com/solutions/libraries/cuda-net (Accessed on January 2, 2013).

HybridDSP, 2011, http://www.hybriddsp.com/ (Accessed on January 2, 2013).

UCGIS (University Consortium for Geographic Information Science), 2013, Research Priorities. Available at: http://ucgis2.org/publication/research-priorities (Accessed on January 2, 2013).

국문초록

병렬컴퓨팅은 주어진 일을 다수의 전산자원에 분산하여 결과를 생산하는 작업이다. 많은 학문에서 병렬컴퓨팅이 적용되어 그 효율성이 입증되어 왔지만, 지리학에서의 병렬컴퓨팅은 도외시 되어 왔다. 공간데이터의 수집 기술이 진보하면서 그 크기가 증가하고 있는 반면, 효율적으로 공간데이터를 분석할 수 있는 병렬컴퓨팅에 대한 지속적인 무관심은 결국 지리학자들이 필요로 하는 분석도구의 부재로 이어질 전망이다. 따라서 이러한 문제를 인식하고 해결하기 위해 손쉽게 병렬컴퓨팅을 어플리케이션을 제작할 수 있는 환경을 제공하는 것은 지속적인 지리학의 발전을 위하여 꼭 필요한 일이다.

지리학 그리고 공간적 분석에서 병렬컴퓨팅이 도외시 된 이유로는 고성능 컴퓨팅을 요구로 하는 지리학전 문제의 부재, 제한적 병렬컴퓨팅 자원으로의 접근, 병렬컴퓨팅을 강조하는 훈련 또는 전통의 부재 등이 있다. 본 논문에서는 이러한 이유들에 대한 해결책을 개인용 컴퓨터의 그래픽 처리장치를 이용한 병렬화 기법을 통하여 모색하였다.

공간분석에서 컴퓨터 자원은 지속적으로 활용되고 있으나, 지리학자들은 컴퓨터를 이용하여 분석과정을 자동화 하고 새로운 알고리즘을 모색하는데 더 많은 관심을 쏟고 있다. 하지만 이러한 관심이 병렬컴퓨팅으로까지 확대한다면 사용 가능한 자원을 모두 사용하여 그 효율을 극대화 시킬 수 있다. 본 논문에서는 이러한 자원을 활용하는 방법을 제시한다. 더 자세히 말하자면, 오늘날의 병렬컴퓨팅 트렌드인 범용 그래픽 처리장치를 지리학과 접목할 수 있는 쉽고 간단한 대규모 병렬 공간분석 도구를 개발 할 수 있는 오픈소스 기반의 환경을 제공하는 것이다.

기존의 논문들을 살펴본 결과, 공간데이터를 병렬컴퓨팅에 적합하도록 재구조화 하면 성능을 더 향상시킬 수 있었다. 따라서 이와 같은 공간데이터 재구조화 방법을 채택하고 본 연구에서 개발된 오픈소스 기반 환경을 통해 지도대수 (Map Algebra)와 경사도 분석 도구를 개발하여 CPU기반의 처리

성능과 GPU기반의 처리 성능을 비교해 보았다. 그 결과 전체적인 분석 과정을 모두 고려하였을 경우, 지도대수는 최대 약 8배의 속도 향상을 보였다. 경사도 분석은 최대 약 11배 정도의 속도 향상을 보였으나 연산 시간만을 고려하였을 때는 약 30배 정도의 성능향상을 보였다.

이러한 개발환경이 지속적으로 배포된다면 많은 병렬화된 공간데이터분석 도구들이 개발 될수 있어 분석 시간의 절감과 공간의사결정지원체계 (Spatial Decision Support System)의 발전에도 크게 기여 할 수 있다. 더 나아가 이러한 분석도구의 개발은 에너지 절감효과도 있어 지속가능한 성장을 지향하는 기술이며, 지리학에서 사용하는 데이터의 크기가 기하급수적으로 증가하고 있기에 그 필요성이 다시한번 강조된다.

Keywords: parallel computing in geography, parallel spatial data analysis, parallel slope analysis, CUDA, GPGPU

Student Number: 2011-21548

Appendix A

- List of Abbreviations

Abbreviation	Full
GIS	Geographic Information Systems
CPU	Central Processing Unit
GFLOPS	Giga Flops
GPU	Graphics Processing Unit
GPGPU	General-Purpose Graphics Processing Unit
CUDA	Compute Unified Device Architectrue
HPC	High-Performnace Computing
UCGIS	University Consortium for Geographical Information Science
OS	Operating System
DEM	Digital Elevation Model
GRASS	Geographic Resources Analysis Support System
ALU	Arithmetic and Logic Unit
SDK	Software Development Kit
VB	Visual Basic
MFC	Microsoft Foundation Class
DLL	Dynamic Linking Library
OSGeo	Open Source Geospatial Foundation
GDAL	Geospatial Data Abstraction Library
SIMD	Single Instruction, Multiple Data
GeoTIFF	Geographically Tagged Image File Format