PARALLEL COMPUTING ON GEOSTATISTICAL DATA USING CUDA

BY

FENG SHAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Advisor:

Professor John Hart

ABSTRACT

Data analysis is receiving considerable attention with the design of new graphics processing units (GPUs). Our study focuses on geostatistical data analysis, which is currently applied in diverse disciplines such as meteorology, oceanography, geography, forestry, environmental control, and agriculture. While geostatistical analysis algorithms are applied in varied branches, those analyses can be accelerated by applying parallel computing using modern GPUs. The highly parallel structure makes modern GPUs more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel.

In our study, we compared the performance between serial and parallel computation on four texture features, including average local variance (ALV), angular second moment (ASM), entropy, and inverse difference moment (IDM). The later three features (ASM, Entropy and IDM) are features obtained using Gray Level Coocurrence Matrices (GLCM). We parallelized the computation by using multiple sliding windows on two-dimensional data concurrently. Our approach also includes, in addition to comparing to serial implementation, measuring the parallelized performance under different data sizes. As a result, parallel computation on geostatistical analyses using GPU can significantly increase the performance and efficiency. It has also demonstrated the possibility to provide solutions for specific needs by reducing the time of computation.

ACKNOWLEDGMENTS

I wish to express sincere appreciation to Professor John Hart for agreeing to work with this thesis project. In addition, special thanks to Dr. Eric Shaffer and Donald Keefer for their assistance in the preparation of this manuscript. Their words of encouragement for continually learning and growing inspired this study.

Having coming so far, my academic research progress could not be achieved without my friends who cared me so much in my five year journey. I hereby thank to my “family” in Urbana-Champaign, Yanshu Guo, Weikun Xiao, Dongjin Wang, Zekun Liu, Simeng Yin, Xuewei Zhang, Xi Wu, Sidi Li, Yingjie Lin, Yueming Zhao. In addition, I would also like to thank my friends all over the world who encouraged my so much, Chen Zhao, Fangqing Xie, Wei Du, Chenlong Zhang, Yifeng Yuan, Chen Wang, Luchun Li, Luyao Ma, Cong Sun, George Petukhov, Xinke Zhang, Yi Song, Yifei Li and Landi Zhang. All of you have helped me through the process of earning my Master’s Degree.

Special thanks to my aunt Yanni Shan, hope you rest in peace in heaven. You raised me up and made me strong. Last but not least, I sincerely thank to my family in Beijing, Wei Shan, Ping Yang, Rong He, Jinying Wei, Changqi Shan, Haipeng Zhang, Hongbin Zhao and Jiansong He.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 GPU and CPU

In this study, we are interested in comparing the execution time between serial and parallel approaches on processing geostatistical data. GPU computation has provided a huge edge over the CPU with respect to computation speed. Hence it is one of the most interesting areas of research in the field of modern industrial research and development [1]. The comparison of the CPU and GPU is shown in Figure 1 and Table 1. As we can see, the CPU is more efficient for handling different tasks of the Operating systems such as job scheduling and memory management while the GPU's forte is the floating point operations.

1.2 CUDA

The evolution of GPU over the years has been towards a better floating point performance. NVIDIA introduced its massively parallel architecture called "CUDA" in 2006-2007. The CUDA programming model provides a straightforward means of describing inherently parallel computations, and NVIDIA's Tesla GPU architecture delivers high computational throughput on massively parallel problems [2].

CUDA allows the programming of GPUs for parallel computation without any graphics knowledge [3][4]. A GPU is presented as a set of multiprocessors, each with its own stream processors and shared memory (user-managed cache). The stream processors are fully capable of executing integer and single precision floating point arithmetic, with additional cores used for double-precision. All multiprocessors have access to global device memory, which is not cached by the hardware. Memory latency is hidden by executing thousands of threads concurrently. Register and shared

memory resources are partitioned among the currently executing threads. There are two major differences between CPU and GPU threads. First, context switching between threads is essentially free. State does not have to be stored/restored because GPU resources are partitioned. Second, while CPUs execute efficiently when the number of threads per core is small (often one or two), GPUs achieve high performance when thousands of threads execute concurrently. CUDA arranges threads into threadblocks. All threads in a threadblock can read and write any shared memory location assigned to that threadblock. Consequently, threads within a threadblock can communicate via shared memory, or use shared memory as a user-managed cache since shared memory latency is two orders of magnitude lower than that of global memory. A barrier primitive is provided so that all threads in a threadblock can synchronize their execution [5].

A CUDA program consists of one or more phases that are executed on either the host (CPU) or a device such as a GPU. As shown in Figure 2 in host code no data parallelism phase is carried out. In some cases little data parallelism is carried out in host code. In device code phases which has high amount of data parallelism are carried out. A CUDA program is a unified source code encompassing both, host and device code.

The GPU we used is a NVIDIA Tesla c2050 (CUDA capability 2.0), which contains 448 CUDA processors with 384-bit bus width and 3072 Mb memory size.

1.3 Figures and Tables

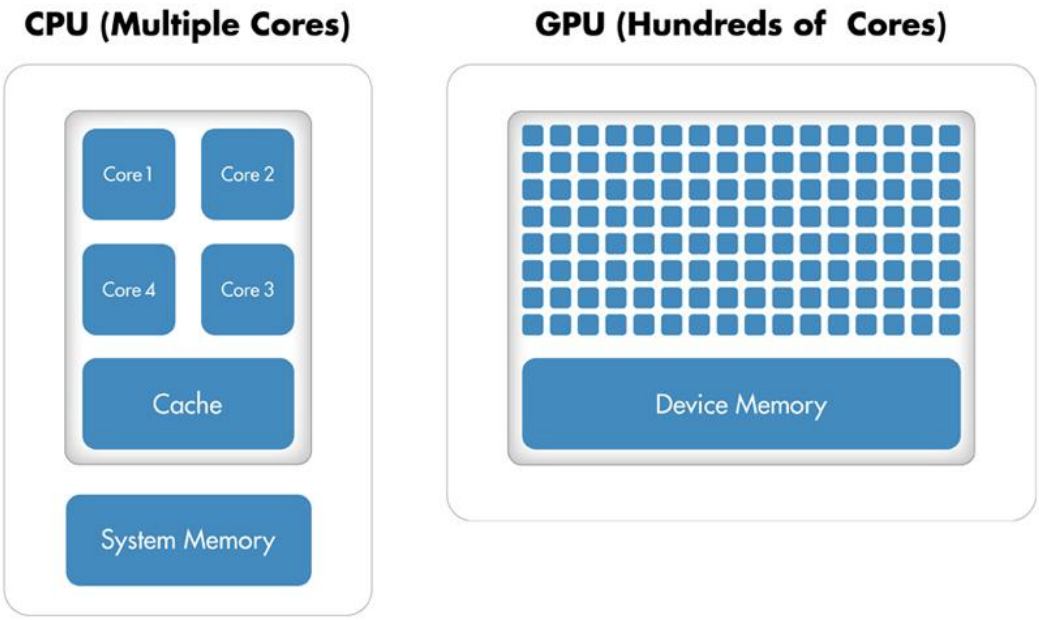

Figure 1: Core comparison between CPU and GPU

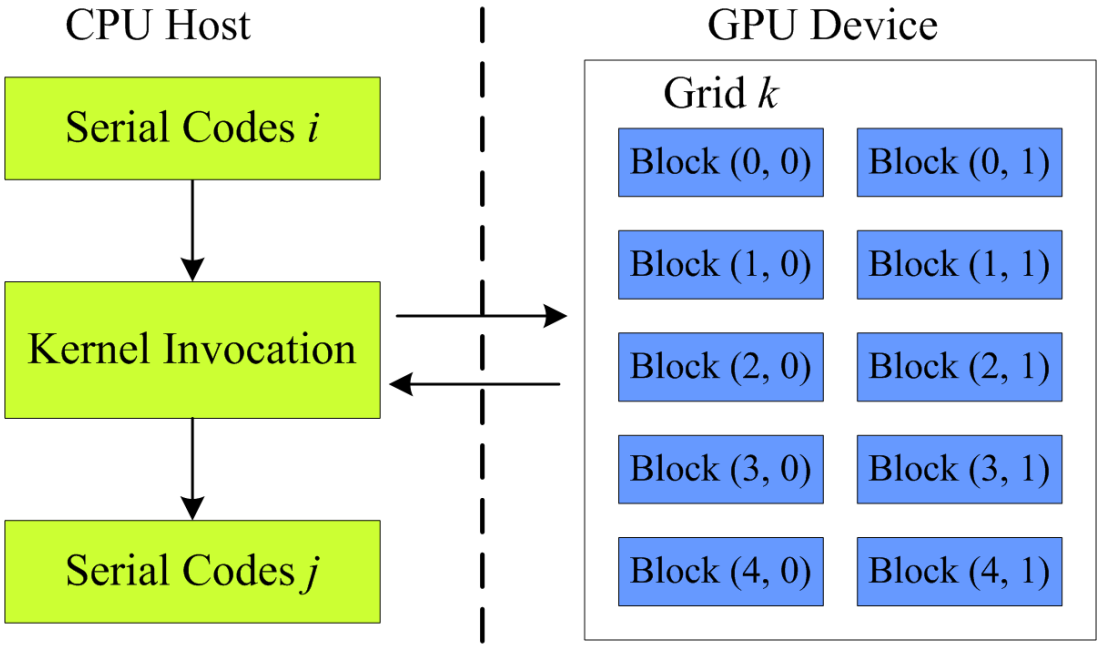

Figure 2: Flow of execution of GPU [6]

CPU	GPU
Fast caches for data reuse	Many math units
Good branching granularity	Fast access to onboard memory
Run a program on different processes/threads	Run a program on different fragment/vertex
Good performance on single thread execution	High throughput on parallel execution
Good for task parallelism	Good for data parallelism
High performance on sequential codes	High performance on parallel codes

Table 1: Comparison between CPU and GPU

CHAPTER 2: METHODS

Texture and spatial pattern are important attributes of images and can be used as features in image classification. Texture metrics measure properties such as roughness/smoothness and regularity. In order to be clinically useful, a texture metric should be robust to changes in image acquisition and digitization. It should be a multi-scale technique and be scale invariant (i.e., independent of magnification). If it is not invariant to transformations in gray scale (i.e., independent of image brightness and contrast), images will need to be histogram equalized prior to computation of the metric. Computational tractability would be an advantage. Other properties may also be desirable. For instance, for computed tomography and MRI images it would be an advantage if the texture metrics were reasonably independent of slice thickness, system noise and reconstruction and post-processing algorithms [7]. The best texture metric for a particular application will combine specific advantages with sufficient sensitivity to be able to discriminate between normal and pathological conditions.

2.1 Average Local Variance

2.1.1 Concepts of ALV

There are a variety of different approaches to characterizing and quantifying stationary texture. They can be divided into two broad categories: either the pattern is analyzed in the spatial domain, or it is analyzed in the spatial frequency domain. Both approaches are complementary and give essentially the same information, although one or the other may be more convenient for a particular type of pattern. A basic tool in the spatial domain is to compute the local variance of pixel values in a square window of given size (say, $N \times N$) around each pixel. Variance measures

distribution about the mean. Low values of local variance indicate smoothness, high values characterize roughness. The process is repeated at different scales to deliver a multiscale measure of texture.

In 1987 the concept of local variance analysis was introduced by Woodcock and Strahler [8]. Graphs of local variance in images as a function of spatial resolution were used to measure spatial structure in images. Construction of these graphs was achieved by degrading the image under study to successively more coarse spatial resolutions, while measuring the local variance value at each of these resolutions.

2.1.2 Methods of ALV

In the average local variance (ALV) method that we used, the local variance of each pixel value is computed for an $N \times N$ (for example 2×2) window and the average of all local variances is taken for the image. We can repeat the process at different levels of spatial resolution by successively aggregating neighboring pixels; there is no need to change the window size. The window operates either (i) by moving over the image by one pixel at a time ("moving" window) or (ii) by slicing the image into the size of the window ("jumping" window) to cover the whole image. A plot of ALV values against spatial resolution constitutes the ALV plot [7]. In previous study, researchers have shown that the plots using jumping windows are more jagged than those using overlapping moving windows, as expected. The results using moving window sampling are essentially low-passfiltered (by a moving average filter) compared to the results using jumping window sampling [7].

This paper reports on work done to explore the speedup for calculating the ALV in parallel using CUDA comparing to serial implementation based on characteristics of the various forms of the ALV function. The work was conducted using synthetically generated image data with the use of 2 by 2 "moving" window.

2.1.3 Calculation of ALV

For a specific window of size $N \times N$, we calculate the sum of all elements in the current window and obtain the mean value of this window:

$$Mean_{ij} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}}{N^2} ,$$

where a_{ij} is the element at row i, column j.

We calculate local variance of a specific window of size N x N using the formula:

$$Local\ Variance_{ij} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij} - Mean_{ij}}{N^2} .$$

Finally, we obtain the ALV value by calculating the average value of all local variances:

$$ALV = \frac{\sum_{i=0}^{nRows-N+1}\sum_{j=0}^{nCols-N+1} LocalVariance_{ij}}{(nRows-N+1)\times(nCols-N+1)} ,$$

where $nRows$ is the total number of rows and $nCols$ is the total number of columns.

2.2 Texture Features From Gray Level Co-occurrence Matrices

2.2.1 Concepts of ASM, Entropy and IDM

Angular second moment (ASM) is a measure of local homogeneity and the opposite of entropy [9]. High values of ASM occur when the pixels in the current

window are very similar. Low values of ASM occur when the pixels are different from each other. ASM is also called Uniformity and its square root is also used as a texture measure which is called Energy. But for this study, we will stay focused on ASM itself.

Entropy is a quantity which is used to describe the "randomness" of pixels in the current window, i.e. the amount of information which must be coded for by a compression algorithm. Low entropy window, such as those containing a lot of black pixels, have very little contrast and large runs of pixels with the same or similar values. For example, an image that is perfectly flat will have entropy of zero. Consequently, it can be compressed to a relatively small size. On the other hand, high entropy windows such as an image of heavily cratered areas on the moon have a great deal of contrast from one pixel to the next and consequently cannot be compressed as much as low entropy windows.

Inverse difference moment (IDM) is a measure of local uniformity present in the current window. It can be seen as the inverse of contrast feature which is a measure of local variation [9]. IDM is high for windows having low contrast and low for windows with high contrast.

2.2.2 Methods for ASM, Entropy and IDM

We used the Gray Level Co-occurrence Matrices (GLCM) to calculate the three texture features [10]. A GLCM is a matrix where the number of rows and columns is equal to the number of gray levels, G, in the image. The matrix element $P(i,j \mid \Delta x, \Delta y)$ is the relative frequency with which two pixels, separated by a pixel distance $(\Delta x, \Delta y)$, occur within a given neighborhood, one with intensity i and the other with intensity j. One may also say that the matrix element $P(i,j \mid d, \theta)$ contains the second order statistical probability values for changes between gray

levels i and j at a particular displacement distance d and at a particular angle θ.

Given an $M \times N$ neighborhood of an input image containing G gray levels from 0 to G − 1, let $f(m,n)$ be the intensity at sample m, line n of the neighborhood. Then

$$P(i,j \mid \Delta x, \Delta y) = W\, Q(i,j \mid \Delta x, \Delta y)$$

where

$$W = \frac{1}{(M-\Delta x)(N-\Delta y)}$$

$$Q(i,j \mid \Delta x, \Delta y) = \sum_{n=1}^{N-\Delta y} \sum_{m=1}^{M-\Delta x} A$$

and

$$A = 1 \; if \; f(m,n) = i \; and \; f(m+\Delta x, n+\Delta y) = j,$$
$$A = 0 \; elsewhere$$

Using a large number of intensity levels G implies storing a lot of temporary data, i.e. a $G \times G$ matrix for each combination of $(\Delta x, \Delta y)$ or (d,θ). One sometimes has the paradoxical situation that the matrices from which the texture features are extracted are more voluminous than the original images from which they are derived. It is also clear that because of their large dimensionality, the GLCM's are very sensitive to the size of the texture samples on which they are estimated. Thus, the number of gray levels is often reduced.

In our study, quantization into 4 gray levels is sufficient for discrimination or degmentation of textures. Even if we can increase the number of gray levels, but since few levels is equivalent to viewing the image on a coarse scale, whereas more levels give an image with more detail. However, the performance of a given GLCM-based feature, as well as the ranking of the features, may depend on the number of gray levels used. So in order to effectively test on large data sets, we minimize the

computation using 4 gray levels with a window size of 3×3.

Because a $G \times G$ matrix (or histogram array) must be accumulated for each sub-image/window and for each separation parameter set (d, θ), it is usually computationally necessary to restrict the (d, θ)-values to be tested to a limited number of values. Figure 3 shows the geometrical relationship of GLCM measurement made for four angles $\theta = 0^o$, 45^o, 90^o and 135^o [10]. Figure 4 illustrates the construction of the four directional spatial co-occurrence matrices for a 3×3 window from an example image which is normalized to four gray levels. Pairs of adjacent pixels are considered in orientation, and the normalized value of those pixels forms the index for incrementing an entry of the co-occurrence matrix. The final matrix for a given point location in the image contains the number of times each possible pair of pixel values occurred in the selected orientation [11].

2.2.3 Calculation of ASM, Entropy and IDM

We use the following notation for the calculation:

P is the spatial co-occurrence matrix.

R is the frequency normalization constant for the selected orientation.

$$ASM = \sum_i \sum_j \left(\frac{P(i,j)}{R}\right)^2$$

ASM is a measure of homogeneity of an image. A homogeneous scene will contain only a few gray levels, giving a GLCM with only a few but relatively high values of $P(i, j)$. Thus, the sum of squares will be high [10][11][12].

$$Entropy = -\sum_i \sum_j \left(\frac{P(i,j)}{R}\right) log\left(\frac{P(i,j)}{R}\right)$$

Inhomogeneous scenes have low first order entropy, while a homogeneous scene has a high entropy [10][11].

$$IDM = \sum_i \sum_j \frac{1}{1+(i-j)^2} \left(\frac{P(i,j)}{R}\right)$$

IDM is also influenced by the homogeneity of the image. Because of the weighting factor $(1 + (i - j)^2)^{-1}$ IDM will get small contributions from inhomogeneous areas. The result is a low IDM value for inhomogeneous images, and a relatively higher value for homogeneous images [10][11][12].

2.3 Figures

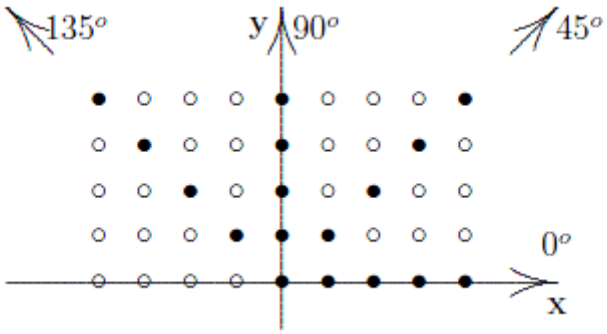

Figure 3: the geometrical relationship of GLCM measurement made for four angles $\boldsymbol{\theta = 0^o, 45^o, 90^o}$ and $\boldsymbol{135^o}$ [10]

(a) $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$

(b) $\begin{bmatrix} \#(0,0) & \#(0,1) & \#(0,2) & \#(0,3) \\ \#(1,0) & \#(1,1) & \#(1,2) & \#(1,3) \\ \#(2,0) & \#(2,1) & \#(2,2) & \#(2,3) \\ \#(3,0) & \#(3,1) & \#(3,2) & \#(3,3) \end{bmatrix}$

(c) Horizontal $\boldsymbol{\theta = 0^o}$, $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ (d) Vertical $\boldsymbol{\theta = 90^o}$, $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

(e) Left Diagonal $\boldsymbol{\theta = 135^o}$, $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 1 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ (f) right diagonal $\boldsymbol{\theta = 45^o}$, $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 1 \\ 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

Figure 4: (a) 3 by 3 window with gray tone range 0 to 3; (b) general form of any spatial co-occurrence matrix for window with gray tone range 0 to 3. $\#(\boldsymbol{i},\boldsymbol{j})$ represents number of times gray tones $\boldsymbol{i}$ and $\boldsymbol{j}$ were neighbors. (c) to (f) spatial co-occurrence matrices derived for four angular orientations.

CHAPTER 3: SOFTWARE IMPLEMENTATION

3.1 Data

Our study focuses on comparing the computation performance between serial and parallel geostatistical data analysis. In order to compare the computational cost, we generated data for arbitrary number of rows and columns in two dimensions. For example we generated 512 by 512, 1024 by 1024 and up to 8192 by 8192 floating point numbers in plain text. Although the program can execute any arbitrary size of data, we are more interested in the relation between number of computations and the time it takes.

For the three features using gray level co-occurrence matrix (GLCM), quantization into 4 gray levels is sufficient for discrimination of textures, namely 0, 1, 2 and 3. Even if we can increase the number of gray levels, but since few levels is equivalent to viewing the image on a coarse scale, whereas more levels give an image with more detail. However, the performance of a given GLCM-based feature, as well as the ranking of the features, may depend on the number of gray levels used. So in order to effectively test on large data sets, we minimize the computation using 4 gray levels.

3.2 ALV Implementation

In the program of serial implementation for average local variance (ALV), the local variance of each pixel value is computed for an N x N window. By adding up the local variances of all windows, we can obtain the average local variance by dividing the sum by the number of windows. Although the program works on arbitrary window size, we use 2×2 window size [7].

Because we are using a relatively small window size (2×2), the time complexity of this program is proportional to the size of data, since it has two for-loops for the two dimensional data. Suppose we have R rows and C columns of data, it takes $O(4RC)$ time. If it runs on $N \times N$ window size, it will take $O(N^2RC)$ time because we use sliding window instead of jumping window.

In the parallel implementation, we parallelized the sliding windows using multiple threads. We parallelized the implementation by the row of the two dimensional data. We copied the two dimensional data on host side to one dimensional data on the device side. We set the block size to be 512 and we parallelized those sliding windows. We can easily locate the data for each window by using the block index and thread index.

3.3 ASM, Entropy and IDM Implementation

For angular second moment (ASM), Entropy and inverse difference moment (IDM), we construct the gray level co-occurrence matrix (GLCM) to obtain each of the features. In order to effectively test on large data sets, we minimize the computation using 4 gray levels. We increased the window size from 2×2 to 3×3. Because it is a more practical implementation of using GLCM by having the co-occurrence matrices in four directions [12].

In serial implementation, the program slides the window frame by frame in two dimensional loops. Then in each 3×3 window, it calculates the co-occurrence matrices in all four directions by scanning all the 9 pixels and updates the matrices on each pixel. Finally, it calculates the ASM, entropy or IDM based on the co-occurrence matrix. For entropy, since the result of taking the logarithm on zero is infinity, we skipped the zero values when taking the logarithm.

Parallel implementation is similar to what we have done in ALV section. We parallelized the sliding windows using multiple threads. We parallelized the implementation by the row of the two dimensional data. We copied the two dimensional data on host side to one dimensional data on the device side. We set the block size to be 512 and we parallelized those sliding windows. We can easily locate the data for each window by using the block index and thread index.

CHAPTER 4: RESULTS

4.1 ALV

In Table 2, we recorded the average execution time of serial implementation of ALV and parallel implementation of ALV. We took the average from 10 independent executions. For 512 by 512 data, we can get 3.41 times of speedup on average. For 8192 by 8192 data, we can obtain 16.06 times of speedup on average.

4.2 ASM

In Table 3, we recorded the average execution time of serial implementation of ASM and parallel implementation of ASM. We took the average from 10 independent executions. For 512 by 512 data, we can get 2.29 times of speedup on average. For 8192 by 8192 data, we can obtain 5.48 times of speedup on average. The speedups are lower comparing to ALV, because in ALV calculation, we only took the average value of all local variances. But in ASM, Entropy and IDM, we need to write the result of each window into file. Writing to file takes more time and result in heavier overhead.

4.3 Entropy

In Table 4, we recorded the average execution time of serial implementation of Entropy and parallel implementation of Entropy. Again, we took the average from 10 independent executions. For 512 by 512 data, we can get 2.05 times of speedup on average. For 8192 by 8192 data, we can obtain 4.73 times of speedup on average. The computation of Entropy is slightly more complicated comparing to ASM and IDM, so it took more time for both serial and parallel execution.

4.4 IDM

In Table 5, we recorded the average execution time of serial implementation of Entropy and parallel implementation of Entropy. We took the average from 10 independent executions. For 512 by 512 data, we can get 2.44 times of speedup on average. For 8192 by 8192 data, we can obtain 6.86 times of speedup on average. IDM computation is slightly less complicated comparing to ASM and Entropy. So it took less time for both serial and parallel execution.

The ALV execution has higher speedup values because it only computes the average of all local variances and does not write each local variance into file. For ASM, Entropy and IDM, we are more interested in the value of each window so we need to write the result of each window into file. Writing to file takes more time and result in heavier overhead for the parallel implementation.

For the ALV computation, the speedup curve is shown in Figure 5. For the three GLCM based computations, the speedup curves are shown in Figure 6. We can see for each of the curves, the slope decreases as the data size becomes larger as all threads are fully loaded.

4.5 Figures and Tables

Data Size	Execution Time (Serial)	Execution Time (Parallel)	SpeedUp
512 x 512	0.638s	0.187s	3.41
1024 x 1024	2.531s	0.302s	8.38
2048 x 2048	10.157s	0.789s	12.87
4096 x 4096	40.609s	2.651s	15.32
8192 x 8192	161.225s	10.040s	16.06

Table 2: ALV Execution time comparison and speedup

Data Size	Execution Time (Serial)	Execution Time (Parallel)	SpeedUp
512 x 512	0.710s	0.305s	2.29
1024 x 1024	2.817s	0.664s	4.24
2048 x 2048	11.283s	2.220s	5.08
4096 x 4096	45.561s	8.505s	5.35
8192 x 8192	184.242s	33.641s	5.48

Table 3: ASM Execution time comparison and speedup

Data Size	Execution Time (Serial)	Execution Time (Parallel)	SpeedUp
512 x 512	0.725s	0.353s	2.05
1024 x 1024	2.828s	0.848s	3.33
2048 x 2048	11.993s	2.913s	4.12
4096 x 4096	49.853s	11.128s	4.48
8192 x 8192	208.716s	44.126s	4.73

Table 4: Entropy Execution time comparison and speedup

Data Size	Execution Time (Serial)	Execution Time (Parallel)	SpeedUp
512 x 512	0.708s	0.290s	2.44
1024 x 1024	2.815s	0.551s	5.11
2048 x 2048	11.279s	1.801s	6.26
4096 x 4096	45.471s	6.773s	6.71
8192 x 8192	183.751s	26.781s	6.86

Table 5: IDM Execution time comparison and speedup

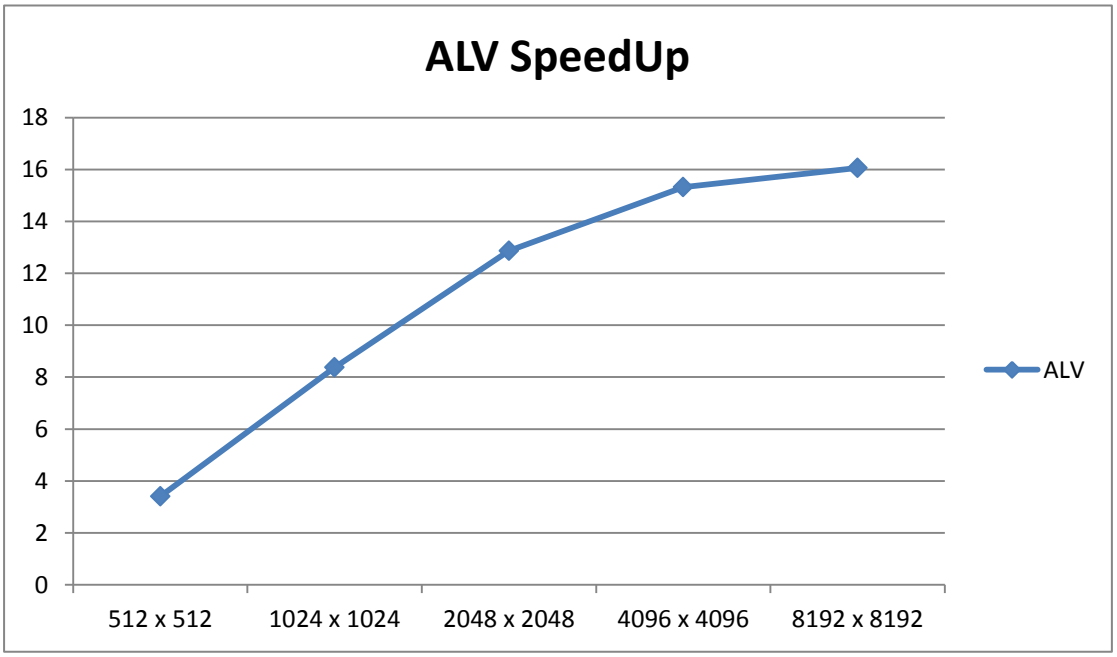

Figure 5: ALV Speedup on different data sizes

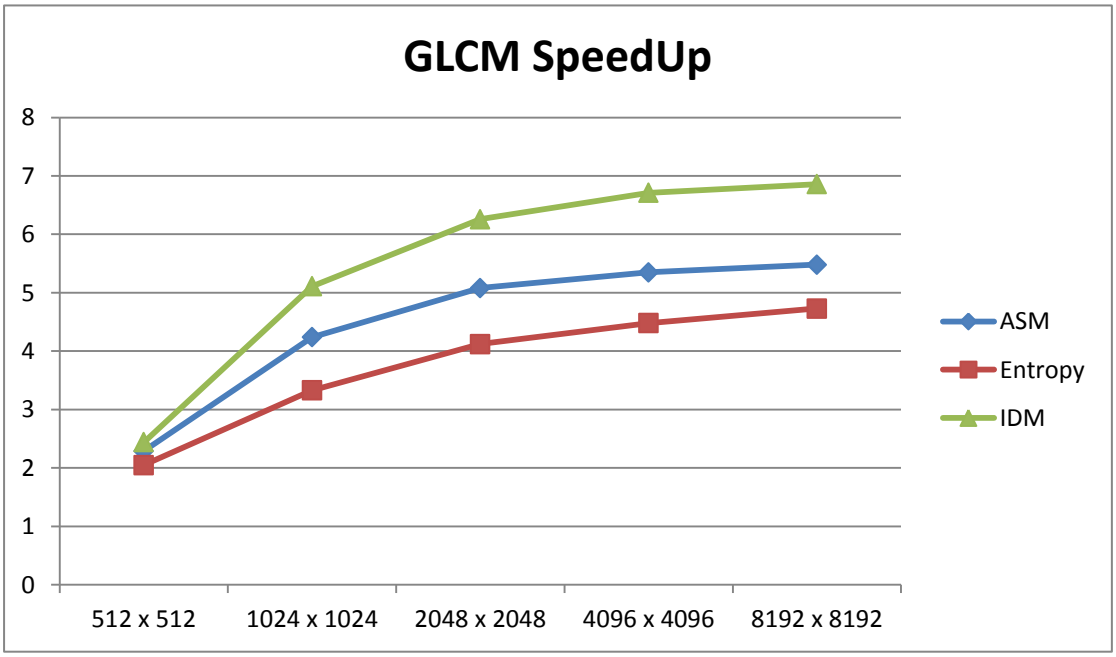

Figure 6: ASM, Entropy, IDM Speedup on different data sizes

CHAPTER 5: DISCUSSION

There are a total of $(R - N + 1) \times (C - N + 1)$ sliding windows in our implementations. If we switch to jumping windows, there will be $\frac{RC}{N^2}$ windows in total. But it is found by geostatistical researchers that the plots using jumping windows are more jagged than those using sliding (overlapping) windows [7].

We copied the two dimensional data on host side to one dimensional data on the device side. The access of data will not be consecutive any more. Because $grid[x][y]$ became $grid[x \times ncols + y]$. The reason why we chose 2 x 2 window size is that the number of windows is proportional to the size of data. But the number of computations is exponential to the size of window, because we used sliding window instead of jumping window.

In order to work with large size of data, we read in the data for the current window. But this would delay the execution time because if we use sliding window, we have to read in the data multiple times for the overlapping windows. The reason why we did not achieve as much speedup for angular second moment (ASM), Entropy and inverse difference moment (IDM) is because of the heavy overhead by constructing the gray level co-occurrence matrices (GLCM).

For the three features using gray level co-occurrence matrix (GLCM), quantization into 4 gray levels is sufficient for discrimination of textures. Even if we can increase the number of gray levels, but since few levels is equivalent to viewing the image on a coarse scale, whereas more levels give an image with more detail. For the 4 gray levels implementation, we have a 4 by 4 matrix for each of the four directions. Suppose we have 16 gray levels to give more detailed expression of the image, we would need a 16 by 16 matrix for each direction. Having more directions would also give more detail, but in this case, parallelizing the implementation in

different directions would be a topic for future studies.

The reason why geostatistical computation could take advantage of parallel computing using CUDA is that, there is no need to write the whole program using CUDA technology. If writing a large application, complete with a user interface, and many other functions, and then most of the code will be written in C++ or any other languages. When we really need to do large mathematical computations, we can simply write kernel call to call CUDA functions we have written. In this way instead of writing complete program you can use GPU for some portion of the code where we need huge mathematical computations.

In order to obtain a statistically reliable estimate of the joint probability distribution, the matrix must contain a reasonably large average occupancy level. This can be achieved either by restricting the number of gray value quantization levels or by using a relatively large window. The former approach results in a loss of texture description accuracy in the analysis of low amplitude textures, while the latter causes uncertainty and error if the texture changes over the large window [10]. In our study on the GLCM based features, we only have 4 gray levels and 3 by 3 window sizes. Future studies could take consideration of more complex features and increase the number of gray levels and window sizes.

CHAPTER 6: CONCLUSION

The highly parallel structure makes modern GPUs more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. As a result, parallel computation on geostatistical analyses using GPU can significantly increase the performance and efficiency. Our study has also demonstrated the possibility to provide solutions for specific needs by reducing the time of computation.

REFERENCES

[1] Ghorpade, J. 2012. GPGPU Processing In CUDA Architecture. Advanced Computing: An International Journal (ACIJ), Vol.3, No.1.

[2] Garland, M. Parallel Computing Experiences with CUDA. IEEE Micro 28, 4, 13-27.

[3] Lindholm, E., Nickolls, J., Oberman, S., Montrym, J. 2008. NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro 28, 2, Mar. 2008, 39-55.

[4] Nickolls, J., Buck, I., Garland, M., and Skadron, K. 2008. Scalable Parallel Programming with CUDA. Queue 6, 2, Mar. 2008, 40-53.

[5] Micikevicius, P. 3D Finite Difference Computation on GPUs using CUDA. Proceeding. GPGPU-2 Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, 79-84.

[6] CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. (http://www.biomedcentral.com/1756-0500/2/73).

[7] Manikka-Baduge, D.C. and Dougherty, G. 2009. Texture analysis using lacunarity and average local variance. Proc. SPIE 7259, Medical Imaging: Image Processing, 725953 (March 27, 2009).

[8] Bocher, P.K., McCloy, K.R. 2003. Analysis of spatial structure in images using local variance. Geoinformation for European-wide Integration, Benes. 185-186.

[9] Haralick, R.M., K. Shanmugam, and I. Dinstein 1973. Textural features for image classification. IEEE Transactions on Systems, Man, and Cybernetics, SMC-3(6):610-621.

[10] Albregtsen, F. 1995 Statistical texture measures computed from gray level coocurrence matrices. University of Oslo.

[11] Peddle, D.R. and Franklin, S.E. 1992. Image texture processing and data integration for surface pattern discrimination. Photogrammetric Engineering and Remote Sensing 57,

[12] Hall-Beyer, M. (2007). The GLCM Tutorial Home Page (Grey-Level Co-occurrence Matrix texture measurements). University of Calgary, Canada