# PERFORMANCE COMPARISON OF GPU AND CPU FOR HIGH-RESOLUTION SATELLITE IMAGE PROCESSING

In-KyuJeong[a],Eun-Jin Im[b],JoonsooChoi[c], Yong-SeungKim[d], ChoenKim[e*]

[a] Graduate student, Department of Applied Information Technology, Kookmin University,
Seoul 136-702, Korea; Tel.: +82-2-910-5080;E-mail: jikyu@kookmin.ac.kr

[b]Professor, Department of Computer Science, Kookmin University,
Seoul 136-702, Korea; Tel.: +82-2-910-4792;E-mail: ejim@kookmin.ac.kr

[c] Professor, Department of Computer Science, Kookmin University;
Seoul 136-702, Korea; Tel.: + 82-2-910-4798;E-mail:jschoi@kookmin.ac.kr

[d]Researcher, Satellite Information Research Center, Korea Aerospace Research Institute;
Daejeon 305-806, Korea; Tel.: +82-42-860-2164;E-mail: yskim@kari.re.kr

[e*]Professor, College of Forest Science, Kookmin University,
Seoul 136-702, Korea; Tel.: +82-2-910-4813;E-mail: choenkim@kookmin.ac.kr

**KEY WORDS**:GPGPU, CUDA, Parallel Processing, High-ResolutionSatellite Imagery

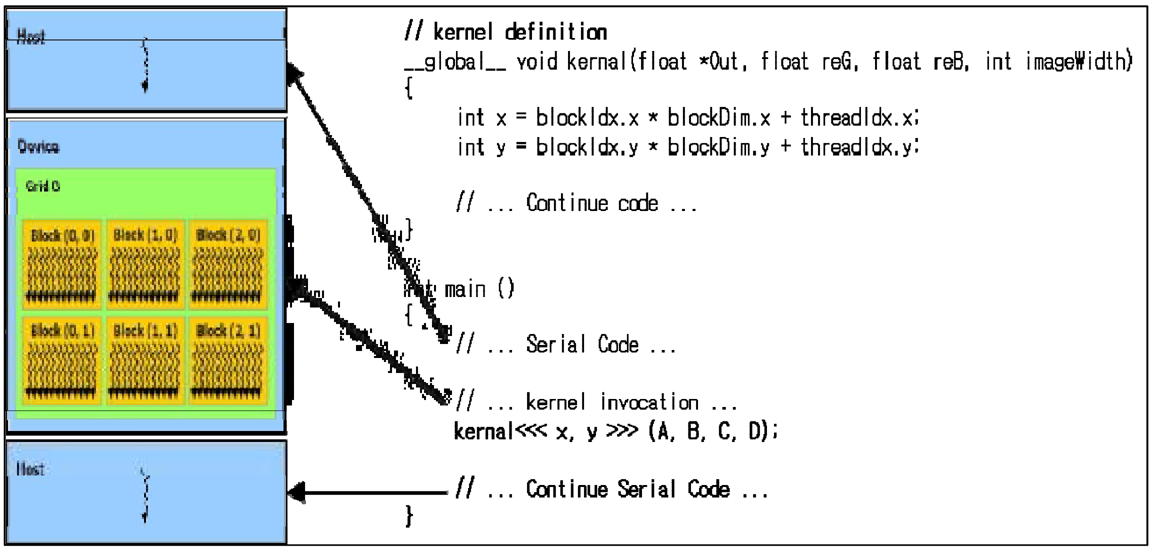
**Abstract:**High resolution satellite images are now widely used for a variety of mapping applications including photogrammetry, GIS data acquisition and visualization. As the spectral and spatial data size of satellite images increase, a greater processing power is needed to process the images. A conventional CPU-based parallel computing is often not good enough for the demand for computational speed to process the images. The graphic processing unit (GPU) is good candidate to achieve this goal. Recently GPUs are used in the field of high complexity processing including many loop operations such as mathematical transforms, ray tracing, etc. In this study we proposed a technique for parallel processing of high resolution satellite images using GPU. We implemented a spectral radiometric processing algorithm on Landsat 7 EMT imagery using CUDA, a parallel computing architecture developed by Nvidia for GPU. Also performance of the algorithm on GPU and CPU is compared.

## 1. INTRODUCTION

Recent tendency of GPU have gained great popularity in the field of high-performance processing for scientific and engineering applications such as image processing (Song et al., 2011). GPU-based desktop computer's advantage of low cost, compact size hardware, high memory bandwidth and high parallelism are what make a this system appealing alternative to massively parallel system made up of commodity CPUs (Song et al., 2011; Lindholm et al., 2008).A good example to GPU parallelism programming language CUDA (compute Unified Device Architecture) is given in NVIDIA. CUDA is remarkably increased programmability for escape from classical General-Purpose computing on Graphics Processing Units (GPGPU) way of transformation to Graphics API, for example OpenGL and DirectX. If you want to overcome the previously mentioned limitations, you can be run some code on GPU while develop applications using the programming language.

The CUDA provides a programming model that is C++, extended with several keywords and constructs (NVIDIA, 2012). The researchers encode a single program that contains both the CPU (host) and the GPU (device) code. These two parts are automatically separated and compiled by the CUDA compiler tool chain (NVIDIA, 2011). Researchers using CUDA allows write device code in C++ functions called ***kernels***.Kernel is disparate from a regular function in that it is executed by many GPU threads in a Single-Instruction Multiple-Data (***SIMD***) style.This style is called Single-Instruction Multiple-Threads (***SIMT***) that each thread executes the entire kernel once.**Figure 1** shows an example that performs serial code executes on the CPU while parallel code executes on the GPU.Each GPU thread is given a special thread ID that is accessible within the kernel, through the built-in variables ***blockIdx*** and ***threadIdx***(Han and Abdelrahman, 2011).

Threads have access to varied GPU memories during execution to kernel.Each thread can read and write or read or write its private ***registers*** and ***local memory***. In addition, single-cycle access time, registers in the GPU memory hierarchy are the fastest. In contrast local memory in the GPU memory hierarchy is the slowest. Each thread block has its private ***shared memory***. All threads have read and write access to the ***global memory***, and read-only access to the ***constant memory*** and the ***texture memory***(Han and Abdelrahman, 2011).Since GPU threads can't access the host memory, the data by a kernel must be copy to above mentioned GPU memories before it is executed.Our goal in this paper of inspired by CUDA is to examine the effectiveness of CUDA as a tool to express parallel computation with performance characteristics on GPU. And we propose a new platform in this study for improving the speed of high-resolution satellite image data processing



```
// kernel definition
__global__ void kernal(float *Out, float reG, float reB, int imageWidth)
{
     int x = blockIdx.x * blockDim.x + threadIdx.x;
     int y = blockIdx.y * blockDim.y + threadIdx.y;

     // ... Continue code ...
}

int main ()
{
     // ... Serial Code ...

     // ... kernel invocation ...
     kernal<<< x, y >>> (A, B, C, D);

     // ... Continue Serial Code ...
}
```

**Figure 1. Host code and kernel code running process.**

## 2. ENVIRONMENT AND METHODS

In this chapter is evaluating the degree of performance improvement for high-resolution image processing method using GPU.In this study, we measured the processing speed for thread-block model and image size changing.Experiments for the optimization of the performance differences depending on the configuration of blocks and threads are needed due to the nature of parallel processing to use a GPU.We change image's pixel size in order to measure processing speed. **Table 3**shows results of speedchanges measurement.

In our experiment, we tested the speed of both the CPU and GPU versions of the conversion to Landsat 7 ETM+ spectral radiance calibration algorithm. During radiometric calibration, pixel values from raw, unprocessed imaged data are converted to units of absolute spectral radiance using 32-bit floating-point calculations(Markham et al. 2004; Chander et al. 2009). The absolute radiance values are scaled to 8-bit numbers representing before output to distribution media.Conversion from $\mathrm{Q_{cal}}$ in Level 1 products back to ETM+ sensor spectral radiance ($\mathrm{L_\lambda}$) requires knowledge of the lower and upper limit of the original rescaling factors. The following equation is used to perform the $\mathrm{Q_{cal}}$-to-$\mathrm{L_\lambda}$ conversion for Level 1 products(Chander et al. 2009):

$$\mathrm{L_\lambda} = \left(\frac{\mathrm{LMAX_\lambda - LMIN_\lambda}}{\mathrm{Q_{cal\,max} - Q_{cal\,min}}}\right)(\mathrm{Q_{cal}} - \mathrm{Q_{cal\,min}}) + \mathrm{LMIN_\lambda} \text{or} L_\lambda = G_{rescale} \times Q_{cal} + B_{rescale}$$

equation's element $\mathrm{G_{rescale}}$ and $\mathrm{B_{rescale}}$ can be expressed as follows:

$$\mathrm{G_{rescale}} = \frac{\mathrm{LMAX_\lambda - LMIN_\lambda}}{\mathrm{Q_{cal\,max} - Q_{cal\,min}}} \text{and} \mathrm{B_{rescale}} = \mathrm{LMIN_\lambda} - \left(\frac{\mathrm{LMAX_\lambda - LMIN_\lambda}}{\mathrm{Q_{cal\,max} - Q_{cal\,min}}}\right)\mathrm{Q_{cal\,min}}$$

The meaning of the equation elements and measure units are shown in **Table 1**. Other elements data to post-calibration dynamic ranges, and mean solar irradiance is given in Markham et al. 2004.Landsat7 ETM+ sensor has a spatial resolution of 30m for the six reflective bands, 60m for the thermal band, and includes a panchromatic (pan) band with a 15m resolution.

**Table 1. Meaning of conversion equation elements**

| Elements | Mean [measure unit] |
|---|---|
| $L_\lambda$ | Spectral radiance at the sensor's aperture $[W/(m^2 sr\ \mu m)]$ |
| $Q_{cal}$ | Quantized calibrated pixel value [DN] |
| $Q_{cal\ min}$ | Minimum quantized calibrated pixel value corresponding to $LMIN_\lambda$ [DN] |
| $Q_{cal\ max}$ | Maximum quantized calibrated pixel value corresponding to $LMAX_\lambda$ [DN] |
| $LMIN_\lambda$ | Spectral at-sensor radiance that is scaled to $Q_{cal\ min}$ $[W/(m^2 sr\ \mu m)]$ |
| $LMAX_\lambda$ | Spectral at-sensor radiance that is scaled to $Q_{cal\ max}$ $[W/(m^2 sr\ \mu m)]$ |
| $G_{rescale}$ | Band-specific rescaling gain factor $[W/(m^2 sr\ \mu m)/DN]$ |
| $B_{rescale}$ | Band-specific rescaling bias factor $[W/(m^2 sr\ \mu m)]$ |

**Table 2. Development environment**

| | CPU | GPU |
|---|---|---|
| Model | Intel Core i5-760 | GTX 550Ti |
| Num of cores | 4cores / 4threads | 192cores/ 4MPs |
| Frequency | 3.20 GHz | 0.98 GHz |
| Shared memory | N/A | 48KB per MP |
| Tool Kit | Visual Studio 2010 | CUDA 4.2 |

If you want to using the NVIDIA's GPU library CUDA, you required the CPU host code and GPU kernel code.Host code is responsible of role for overall handling such as GPU variable declarations, initialization and kernel code execution.In this paper, the kernel code is executed by the internal threads in the GPU at the same time for operations that convert the digital number values(DN) of the image processing.In this system is called the kernel function in the host code using variable and thread and block for parallel processing. In this paper, we perform experiment the processing speed for thread-block model and image size changing.**Table 2**showsthe environment of our experiment.

## 3. RESULTS

**Table 3**showsthe results for experiments the processing time of the CPU and GPU.In the table shows the time results of experiment by average 10 running.Experiments have increased by 1000pixel image size from 1000x1000 to 10000x10000.There is a difference in speed depending on the configuration of the block and threads. Therefore blocks and threads configuration is very important. In this study configuration of block and threads is called one-dimensional block and thread model.In this study we have applied the block and threads model to one-dimensional because, algorithm is a simple operation of single band image.

**Table 3. Comparing the processing time (sec) for image size (pixel) and thread-block model**

| | Structure | | Processing time for each image size | | |
|---|---|---|---|---|---|
| | Block | Thread | 1000 X1000 | 5000 X5000 | 10000 X10000 |
| GPU | 4 | 48 | 0.68 | 3.12 | 10.23 |
| | 16 | 96 | 0.58 | 2.87 | 9.86 |
| | **32** | **192** | **0.41** | **2.49** | **8.54** |
| | 48 | 192 | 0.42 | 2.53 | 8.74 |
| | 64 | 256 | 0.45 | 2.88 | 8.58 |
| CPU | **N/A** | **N/A** | **0.52** | **2.88** | **11.58** |

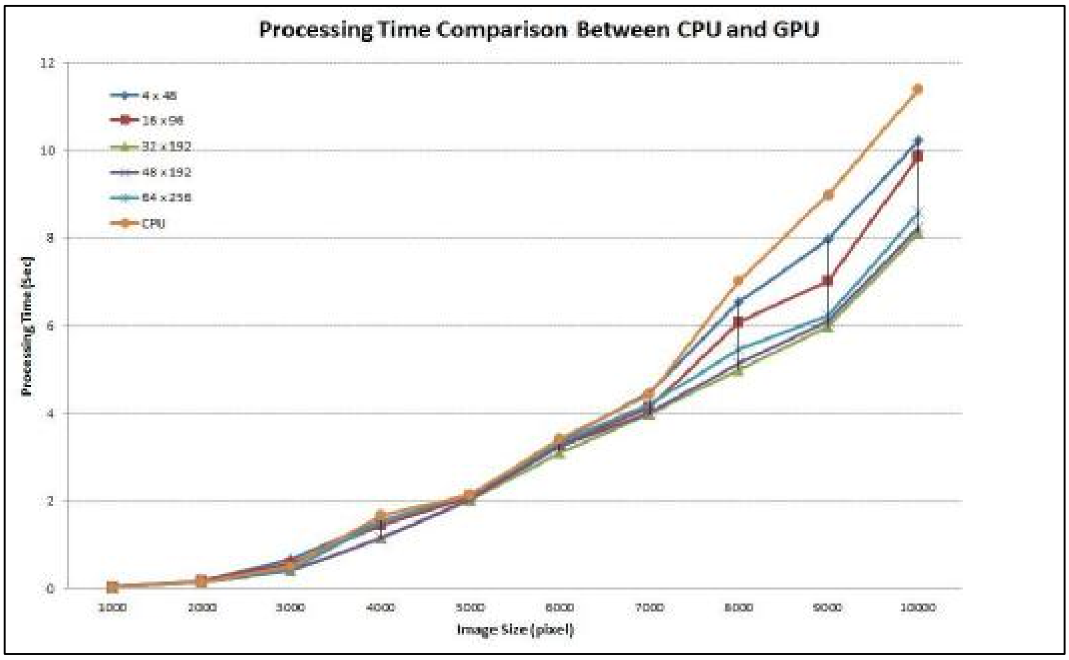


**Figure 2. Processing time comparison between CPU and GPU**

## 4. CONCLUSION

High resolution satellite images are now large capacity for higher spatial resolution.We must perform various calibrations to acquire information.If you want to the repeatedly same-operations for each pixel, you can perform parallel processing method for multiple data.In this paper, we proposed to efficient parallel processing using CUDA.Now we mostly use as general-purpose high-performance graphics cards on the PC.Therefore parallel processing by GPU is an important technology for dramatically improve the efficiency of data processing. Of course, single-CPU is a current trend that supports parallel processing and to configure more than four ALU.However, CPU have lower performance compared to the GPU's more than hundreds ALU of specialized in floating-point arithmetic.

The data transfer rate than processing speed was one of the old problems of parallel system, and the problem still exists in a GPU parallel system. Simultaneously using multiple processors can handle large amounts of data and of course faster.However it is difficult to achieve the desired speedup due to data bottlenecks.In this paper, we conducted experiments changing the threads-block model in order to minimize these problems.In this study, it didn't show a big speed improvement compared with the C programming because, processing of performed by the GPU kernel function is relatively simple.However, it is clearly demonstrated the speed improvements.In the future will be consider as meaningful research if optimizing the performance of CUDA.

**REFERENCES:**

Asanovíc, K., R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick, 2006. The landscape of parallel computing research: A view From Berkeley, Electrical Eng. And Computer Sci., University of California, Berkeley, Tech. Rep. No.UCB/EECS-2006-183.

Chander, G., B. L. Markham, and D. L. Helder, 2009. Summary of current radiometric calibration coefficients for Landsat MSS, TM, ETM+, and EO-1 ALI sensors, Remote Sensing of Environment, 113, pp. 893-903.

Christophe, E., J. Michel, and J. Inglada, 2011. Remote sensing processing: From multicore to GPU, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 4(3), pp. 643-652.

Han, T. D., and T.S. Abdelrahman, 2011.hiCUDA: High-level GPGPU programming, IEEE Transactions on Parallel and Distributed Systems, 22(1), pp. 78-90.

Lindholm, E., J. Nickolls, S. Oberman, and J. Montrym, 2008. NVIDIA Tesla: A unified graphics and computing architecture, IEEE Micro, 28(2), pp. 39-55.

Markham, B. L., K. Thome, J. Barsi, E. Kaita, D. Helder, J. Barker, and P. Scaramuzza, 2004. Landsat-7 ETM+ On-orbit reflective-band radiometric stability and absolute calibration, IEEE Transactions on Geoscience and Remote Sensing, 43, pp. 2810-2820.

NVIDIA, 2012.NVIDIA CUDA C Programming Guide v4.2,http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf.

NVIDIA, 2011, The CUDA Compiler Driver NVCC V4.1, http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/nvcc.pdf.

NVIDIA, 2012.CUDA Occupancy Calculator.http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation.

Song, C., Y. Li, and B. Huang, 2011. A GPU-accelerated wavelet decompression system with SPIHT and Reed-Solomon decoding for satellite images, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 4(3), pp. 683-690.