ORTHORECTIFICATION BY USING GPGPU METHOD

H. Sahin [a, *], S. Kulur [b]

[a] General Command of Mapping, 06100 Dikimevi, Ankara, Turkey – hakan.sahin@hgk.msb.gov.tr
[b] ITU, Civil Engineering Faculty, 80626 Maslak, Istanbul, Turkey - kulur@itu.edu.tr

Commission IV, WG IV/3

KEYWORDS: Processing, Programming, Orthorectification, Technology, Orthoimage.

ABSTRACT:

Thanks to the nature of the graphics processing, the newly released products offer highly parallel processing units with high-memory bandwidth and computational power of more than teraflops per second. The modern GPUs are not only powerful graphic engines but also they are high level parallel programmable processors with very fast computing capabilities and high-memory bandwidth speed compared to central processing units (CPU). Data-parallel computations can be shortly described as mapping data elements to parallel processing threads. The rapid development of GPUs programmability and capabilities attracted the attentions of researchers dealing with complex problems which need high level calculations. This interest has revealed the concepts of "General Purpose Computation on Graphics Processing Units (GPGPU)" and "stream processing". The graphic processors are powerful hardware which is really cheap and affordable. So the graphic processors became an alternative to computer processors. The graphic chips which were standard application hardware have been transformed into modern, powerful and programmable processors to meet the overall needs. Especially in recent years, the phenomenon of the usage of graphics processing units in general purpose computation has led the researchers and developers to this point. The biggest problem is that the graphics processing units use different programming models unlike current programming methods. Therefore, an efficient GPU programming requires re-coding of the current program algorithm by considering the limitations and the structure of the graphics hardware. Currently, multi-core processors can not be programmed by using traditional programming methods. Event procedure programming method can not be used for programming the multi-core processors.

GPUs are especially effective in finding solution for repetition of the computing steps for many data elements when high accuracy is needed. Thus, it provides the computing process more quickly and accurately. Compared to the GPUs, CPUs which perform just one computing in a time according to the flow control are slower in performance. This structure can be evaluated for various applications of computer technology.

In this study covers how general purpose parallel programming and computational power of the GPUs can be used in photogrammetric applications especially direct georeferencing. The direct georeferencing algorithm is coded by using GPGPU method and CUDA (Compute Unified Device Architecture) programming language. Results provided by this method were compared with the traditional CPU programming. In the other application the projective rectification is coded by using GPGPU method and CUDA programming language. Sample images of various sizes, as compared to the results of the program were evaluated. GPGPU method can be used especially in repetition of same computations on highly dense data, thus finding the solution quickly.

1. INTRODUCTION

The graphic processing units (GPU) on the graphic cards integral parts of computers are really developed today according to the last ten years. The development was the increase of the GPUs performance and capabilities. The modern GPUs are not only became powerful graphic engines and also they are high level parallel programmable processors with very fast computing capabilities and high memory bandwidth speed comparing to central processing units (CPU). The rapid development of GPUs programmability and capabilities attracted the researchers dealing with complex problems who need highly level calculation. This interest has revealed the concepts of "General Purpose Computation on Graphics Processing Units (GPGPU)" and "stream processing".

Real time processing of imagery airborne data will be very important in the near future (Thomas et al. 2008). For rapid evaluating data coming from unmanned air vehicles (UAV) in military applications, for supporting rescue and security forces, and also for obtaining surveys in disaster scenarios or mass events an airborne real time image processing system is required. So the need is speed for processing the imagery data.

Thanks to the state of the art GPUs, there is now commodity hardware, providing peak performances more than teraflops per second, and such tremendous computational resource certainly helps the goal of orthorectification in real time.

The announcements of immense computational speedups and fascinating developments in GPU hardware inspired to use general purpose parallel processing in image processing. In literature, there exists limited work about the orthorectification

* Corresponding author. This is useful to know for communication with the appropriate person in cases with more than one author.

with GPGPU and photogrammetric using GPU parallel processing.

General purpose parallel programming can use GPUs not only for graphics but also for removing the burden of the non-graphic computational workload which is traditionally handled by a CPU. Significant computational speedups have been achieved by various researchers from different disciplines using general purpose parallel programming. Although GPU-based non-graphics computation is well suited to data-parallel tasks such as image processing kernels and matrix operations, it is also possible to accelerate many other applications by adapting existing algorithms to the general purpose parallel programming (Yılmaz, 2010). Therefore it seems reasonable to exploit tremendous computing power of GPUs for orthorectification, since computational power is an important concern,

In this study we currently used the GPGPU method for orthorectification procedure.

2. GPGPU AND STREAM PROCESSING

The main reason for coming to the agenda the GPUs is really powerful and as well as cheap hardware available. These chips were standard application equipment in near future but they evolved into powerful and programmable processors to meet general needs today. Especially in recent years, GPUs can be used in general purpose calculations phenomenon attracted the attentions of researchers dealing with complex problems which need high level calculations. The biggest problem here is; GPUs uses different programming algorithm. Because of that reason, the effective GPU programming requires the re-writing existing program algorithm using graphical terms again considering to hardware structure and limitations. Today, the multicore processors can not be programming using traditional programming methods. So the usage of typical event programming procedure can not be possible for programming the multicore processors.

Programming model changed to stream computing and processing. In this new model for identifying the kernel functions, that apply intensive calculation each element in the flow, all the input and output data qualified as stream. There are lots of processors on the GPU that process these streams.

For example Nvidia GTX580 series card has 512 unit stream processors (CUDA processors). So we can consider such as 512 computers stay side by side. The graphic cards can do multiple intensive processes with these stream processors at the same time.

GPUs can make more parallel calculation than CPUs. This can be shown in Figure 1. “Flop” term defines the processor speed. It means “the number of floating points per second”. We can see clearly from the chart, Nvidia graphic processors about ten times faster than Intel processors in year 2010.

Memory bandwidth term means, amount of the data transferring in per second between GPU and graphic card memory. Theoretical maximum memory bandwidth is typically computed by multiplying the width of the interface by the frequency at which it transfers data. This term speedup is a factor that improves the graphic card performance. Given the floating-point operation per second to increase capacity in line with the years, has been an improvement in memory bandwidth. Figure 2 show that, GPUs bandwidth reached a rate 6 times more than CPUs bandwidth.

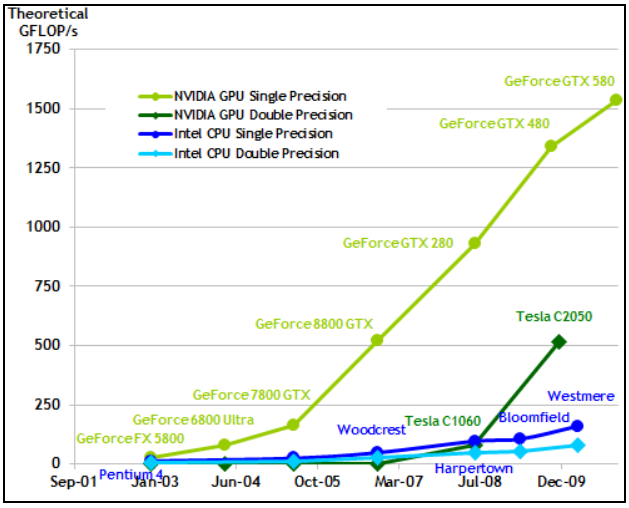

Figure 1. Development of floating-point operations per second for the CPU and GPU (Nvidia, 2011).

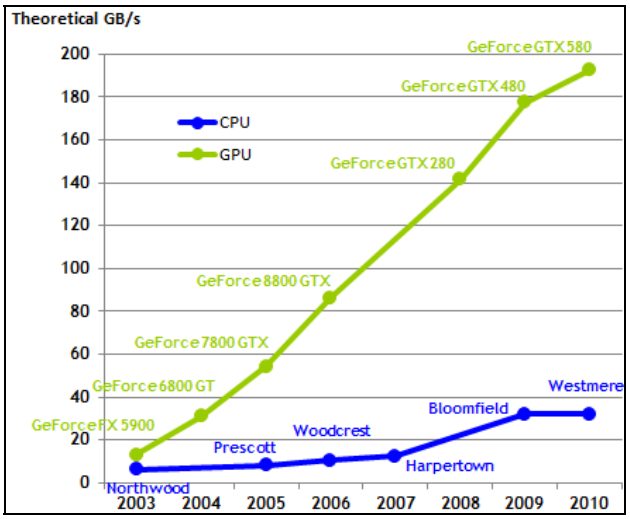

Figure 2. Development of memory bandwidth for the CPU and GPU (Nvidia, 2011a).

The reason behind the discrepancy in floating-point capability between the CPU and the GPU is that the GPU is specialized for compute-intensive, highly parallel computation – exactly what graphics rendering is about – and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control, as schematically illustrated by Figure 3.

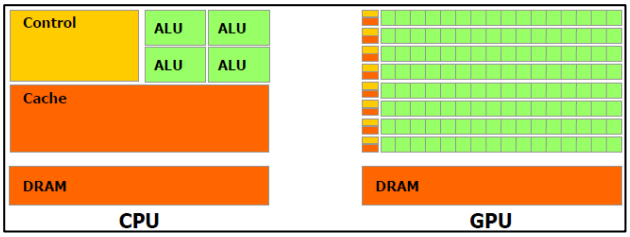

Figure 3. The general structure of CPU and GPU and difference in the number of transistors they have (Nvidia, 2011b).

More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations – the same program is executed on many data elements in parallel (with high arithmetic intensity) – the ratio of arithmetic operations to memory operations. Because the same program is executed for each data element, there is a lower requirement for sophisticated flow control; and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches.

Data-parallel processing maps data elements to parallel processing threads. Many applications that process large data sets can use a data-parallel programming model to speed up the computations. In 3D rendering, large sets of pixels and vertices are mapped to parallel threads. Similarly, image and media processing applications such as post-processing of rendered images, video encoding and decoding, image scaling, stereo vision, and pattern recognition can map image blocks and pixels to parallel processing threads. In fact, many algorithms outside the field of image rendering and processing are accelerated by data-parallel processing, from general signal processing or physics simulation to computational finance or computational biology.

3. CUDA (COMPUTE UNIFIED DEVICE ARCHITECTURE

Long time ago, the developers have tried to use GPUs for parallel computing. The initial use of these initiatives (such as rasterizing and Z-buffering) is very primitive and limited to fully utilize the hardware functions. But the shading calculations have accelerated the matrix calculations.

There was a session called “GPGPU” for “GPU computing” in SIGGRAPH conference in 2003. But this session has been almost no participation. In this session the best known topic was “BrookGPU” as an stream programming language. Before the publication of this programming language there were two software development applications known Direct3D and OpenGL. However, limited number of GPU applications can be developed with these languages. After that, “Brook project” made it possible to using GPUs as a parallel processor and can be programming with C language. This project was developed by Stanford University and has attracted attention of graphic cards companies “NVIDIA” and “ATI” who are the two different designer and manufacturer. Later, some people who developed “Brook”, was joined to NVIDIA Company and started offering a new marketing strategy as a unit of parallel computation. Thus, direct use of graphics hardware has emerged, and on behalf of a structure called the NVIDIA CUDA.

Although announcements were made earlier, Nvidia introduced CUDA to the public in February, 2007. This technology was designed to meet several important requirements for a wide audience’s use. One of the most important requirement is the ability to program GPUs easily. Simplicity is necessary to ease GPU parallel programming and enable its use in more disciplines. Before CUDA, GPU parallel programming was limited to shader models of the graphics APIs. Thus, only the problems well-suited to the nature of vertex and fragment shaders were computed by using GPU parallel processing. Additionally, expressing general algorithms in terms of textures and GPU provided 3D operations by using only float numbers were among the issues that limited the popularity of the GPU computing. To achieve the goal of making GPU parallel programming easy and practical, Nvidia offered to use C programming language with minimal extensions. Another important issue is the heterogeneous computing model, which takes it possible to use CPU and GPU resources together. CUDA lets programmers divide the code and data into sub-parts, considering their suitability to the CPU/GPU architecture and respective programming techniques. Such a division is possible because the host and device have their own memories. In this sense, it also becomes possible to port existing implementations gradually, from the CPU to the GPU (Yilmaz, 2010). Briefly, CUDA technology is a software-hardware computing architecture developed by NVIDIA and based on the C programming language for parallel calculation to controls GPU commands and video memory.

CUDA works with all Nvidia GPUs from the G8x series onwards and new series including GeForce, Quadro and the Tesla line. The data-parallel and thread-parallel architecture introduces scalability. Since no extra effort is necessary to run existing solution, the new GPUs are capable of running more processing threads. It means that the code designed for the Nvidia 8 series runs faster in Nvidia GTX series without any additional coding. Nvidia states that programs developed for the G8x series will also work without modification on all future Nvidia video cards, due to binary compatibility.

The three abstractions offered by Nvidia ensure the granularity required for good data parallelism and thread parallelism. These below listed abstractions are designed to make CUDA programmers life easy.

- Thread Group hierarchy: Threads are packed into blocks which are also packed into a single grid.
- Shared memories: CUDA let threads use six different memories that are designed to meet different requirements.
- Barrier synchronization: This abstraction synchronizes threads within a single block and makes a thread wait the others to finish related computing, before going further.

C for CUDA makes it possible to write functions that run on the GPU by using C language. These functions are called “kernels”, which are executed for each thread in a parallel manner unlike the conventional serial programming functions that run only once.

CUDA’s architecture offers thread hierarchy in top-down order as follow:

1. Grid: A grid contains one or two dimensional blocks.
2. Blocks: A block contains one, two or three dimensional threads. Current GPUs allow a block to contain 512 threads at most. The blocks are executed independently, and they are directed to available processors to provide scalability.
3. Thread: A thread is the basic execution element.

This hierarchy and the structure are depicted by Figure 4. For example if it is assumed that 1048576 pixels to be processed independently in parallel manner and the block size is determined as 512, the there are 2048 grids.

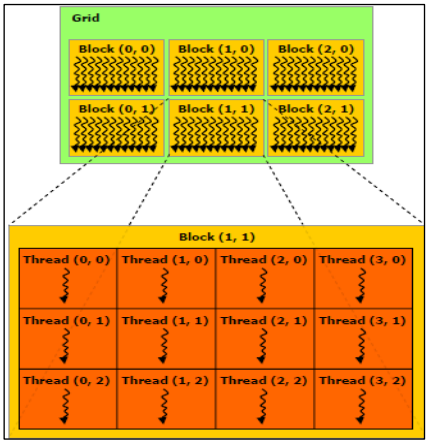

Figure 4. CUDA thread hierarchy.

CUDA processing flow shown in Figure 5 as follow:

1. Copy data from main mem to GPU mem
2. CPU instructs the process to GPU
3. GPU execute parallel in each core
4. Copy the result from GPU mem to main mem

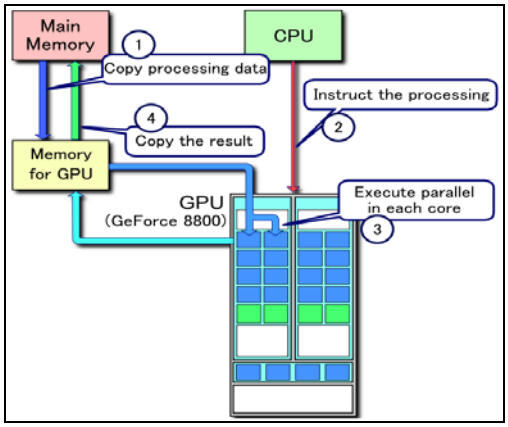

Figure 5. Processing flow on CUDA (URL 1).

Some of the applications made by parallel processing with CUDA shows that GPUs processing time is 5-150 times faster than CPUs. Some Nvidia research results given in Figure 6.

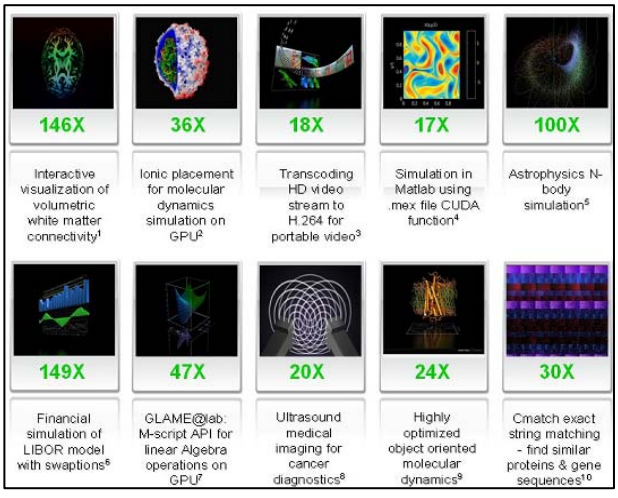

Figure 6. The results of some applications made by Nvidia.

4. PROJECTIVE RECTIFICATION WITH CUDA

To perform a projective rectification, a geometric transformation between the image plane and the projective plane is necessary. For the calculation of the eight unknown coefficients of the projective transformation, at least four control points in the object plane are required. Projective transformation is applicable to rectifying imagery of flat terrain or images of facades of buildings, since it does not correct the relief displacement. The equations for projective rectification are given as follows: E.g.

$$X = \frac{a_1 x + b_1 y + c_1}{a_3 x + b_3 y + 1} \qquad (1)$$

$$Y = \frac{a_2 x + b_2 y + c_2}{a_3 x + b_3 y + 1}$$

where X, Y = rectified coordinates
x, y = tilted image coordinates
a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3 = transformation parameters

Projective rectification algorithm used as an example. In this example we used 4096x4096 pixel size image which taken on a plane area. Four image and object coordinates known points chosen from the image. With these points we solved the equation and calculated the coefficients of the equation. With these coefficients, object-space coordinates of each pixel on the image is calculated. All these calculation procedures parallelized and coded with using CUDA programming language. The same code runs on CPU and also GPU. So we can compare the results with processing time. When the program runs, the computer screen snapshot shown in Figure 7.

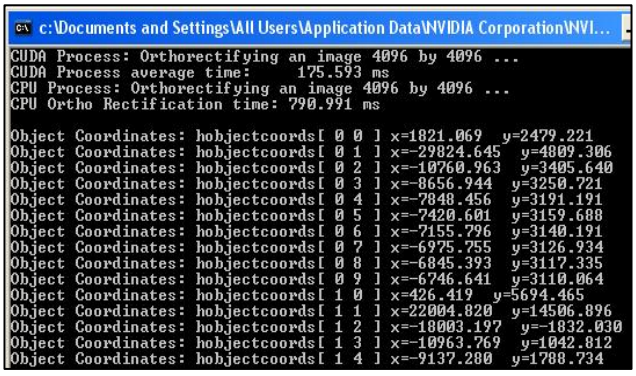

Figure 7. Projective rectification screen snapshot.

In this study, projective rectification procedure applied to each pixel of the sample image. In this situation, the problem could be separated to threads with CUDA and the graphical processors processed the data at the same time. So we could take the calculation result quickly.

If we analyze the program output from Figure 7, calculation GPU time 175,593 millisecond, for the same procedure CPU time is 790,991 millisecond. The discrepancy with GPU and CPU time shows that GPU 4.5 times faster than CPU. In this example we used 4096x4096 pixel size image and Nvidia Geforce 8600M GT graphic card with Core 2 Duo 2.2GHz CPU. If the image size getting smaller, CPU performance increase. So with the very huge and repetitive calculation problems using GPU is more efficient and produces results rapidly.

Image Size (pixel)	GPU Time (millisecond)	CPU Time (millisecond)	CPU / GPU (rate)
1024 x 1024	14,09	48,917	3,47
2048 x 2048	75,994	190,626	2,51
4096 x 4096	175,593	790,991	4,50

Table 1. Calculation results with using GPU and CPU.

5. DIRECT GEOREFERENCING WITH CUDA

Direct Georeferencing is the direct determination of the position and orientation parameters of a sensor. It is an enabling technology for quantitative data acquisition and mapping applications where precise orientation and the position of the sensor are required. A direct georeferencing system provides the position and orientation of the sensor required to register the acquired data in geographic coordinates. In photogrammetry, direct georeferencing is used to produce measurement of the exterior orientation parameters for each image without use of ground control points or aerial triangulation.

Direct georeferencing is based on colinearity equations to provide a relationship between pixel positions and corresponding positions on ground. The procedure of differential rectification is applied in combination with the forward projection (direct) method of orthoimage reprojection. This is based on the colinearity principle, which states that the projection center of a central perspective image, an object point, and its photographic image lies upon a straight line.

Direct Georeferencing for aerial digital frame cameras consists of six stages and the data needed for each step are shown in Figure 8 (Kiracı, 2008).

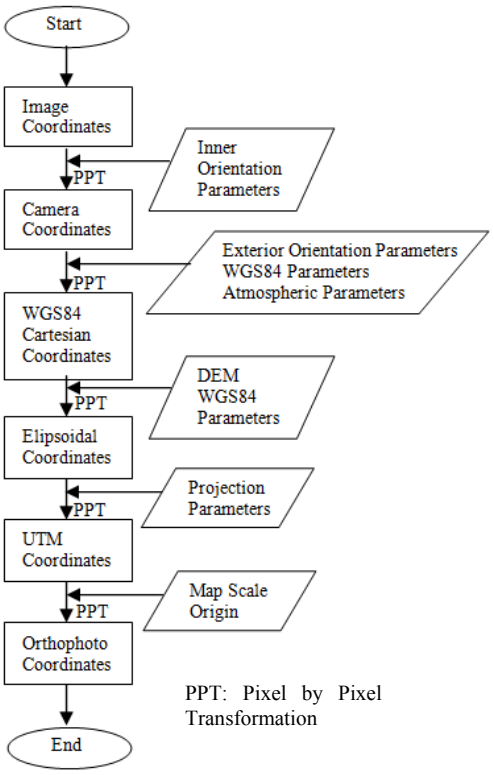

Figure 8. Direct Georeferencing algorithm.

It is shown in figure that the procedure is really suitable for GPGPU and CUDA programming. Because of that the procedures must be done for each pixel of image. In every loop we must do pixel by pixel transformation. So all these calculation procedures parallelized and coded with using CUDA programming language.

The program running results are; calculation GPU time 1328234 millisecond, for the same procedure CPU time is 6827282 millisecond. The discrepancy with GPU and CPU time shows that GPU 5.14 times faster than CPU (Table 2). In this example we used 4096x4096 pixel size image and Nvidia Geforce 8600M GT graphic card with Core 2 Duo 2.2GHz CPU. If the image size getting smaller, CPU performance increase. So with the very huge and repetitive calculation problems using GPU is more efficient and produces results rapidly.

Image Size (pixel)	GPU Time (millisecond)	CPU Time (millisecond)	CPU / GPU (rate)
1024 x 1024	84675	324563	3,83
2048 x 2048	342634	934547	2,73
4096 x 4096	1328234	6827282	5,14

Table 2. Calculation results with using GPU and CPU.

6. RESULTS

In this study covers how general purpose parallel programming and computational power of the GPUs and GPGPU method can be used in photogrammetric orthorectification applications especially direct georeferencing and projective rectification. These two methods coded with CUDA programming language. The results obtained are evaluated; the method is really suitable for image processing and photogrammetry especially if we do the same calculations to per image pixels. Also it is suitable for intensive calculation procedures. GPGPU and CUDA programming method make the calculation really fast. We can increase the number of applications which can be adapted to photogrammetry and image processing that require intensive computation and speed.

In our study we didn't make any optimization of the CUDA programming code. So in future studies we will focus on optimization of the coding procedures. Also our development software doesn't have a user interface yet. We will make a user interface for this software.

Today, with this method the images obtained through a variety of platforms, to be georeferenced correctly and quickly. Especially if it is important that real time processing of imagery airborne data for natural disasters, for rapid evaluating data coming from unmanned air vehicles (UAV) in military applications, for supporting rescue and security forces, and also for obtaining surveys in disaster scenarios or mass events an airborne real time image processing system is required. For this purpose we can use this GPGPU method for rectifying process of imagery data.

7. REFERENCES

Bettemir O.H., 2006. "*Sensitivity and Error Analysis of a Differential Rectification Method for CCD Frame Cameras and Pushbroom Scanners*", Master Thesis, METU, Ankara.

Biesemans, J and Everaerts, J., 2006. "Image Processing Workflow for the Pegasus HALE UAV Payload." *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences,* Antwerp, Belgium, Vol. XXXVI-1/W44.

Gruen, A. and Beyer, H., 2001. "Calibration and Orientation of Cameras in Computer Vision", *Springer Series in Information Sciences.* Vol. 34, Springer-Verlag Berlin Heidelberg.

Jacobsen, K., 2002. "Calibration aspects in direct georeferencing of frame imagery" *In: Int. Archives PhRS (34)*, 1 I, pp. 82-89, Denver.

Karslioglu, M.O., Friedrich J., 2005. "A New Differential Geometric Method to Rectify Digital Images of the Earth's Surface Using Isothermal Coordinates", *IEEE Transactions on Geoscience and Remote Sensing,* Vol. 43, No. 3, March.

Kiracı, A.C., 2008. "*Direct Georeferencing and Orthorectification of Airborne Digital Images*", Master Thesis, METU, Ankara.

Kraus, K., 2007. *Fotogrametri Cilt 1 Fotoğraflardan ve Lazer Tarama Verilerinden Geometrik Bilgiler.* Istanbul Technical University, Nobel Yayın Dağıtım, 1.Basım.

Mercedes Marqu´es, Gregorio Quintana-Ort´ı, Enrique S. Quintana-Ort´ı, Robert van de Geijn. 2009. Using graphics processors to accelerate the solution of out-of-core linear systems *8th IEEE International Symposium on Parallel and Distributed Computing*, Lisbon.

Novak, K. 1992. Rectification of Digital Imagery, *Photogrammetric Engineering and Remote Sensing*, 339-344.

Nvidia, 2009. CUDA Architecture, Introduction and Overview, Nvidia Corp., California, USA.

Nvidia, 2011a. OpenCL Programming Guide for the CUDA Architecture, Nvidia Corp., California, USA.

Nvidia, 2011b. CUDA C Programming Guide, Nvidia Corp. California, USA.

U. Thomas, F. Kurz, D. Rosenbaum, R. Mueller, P. Reinartz, 2008. "GPU-based Orthorectification of Digital Airborne Camera Images in Real Time", *The International Archives Of The Photogrammetry, Remote Sensing And Spatial Informatıon Sciences, ISPRS Congress Beijing,* Volume XXXVII Part B1 Commission I.

White, S. and M Aslaksen, 2006. "Use of Direct Georeferencing to Support Emergency Response". *NOAA's PERS Direct Georeferencing Column*.

Yılmaz, E., 2010. *Massive Crowd Simulation with Parallel Processing*, PhD Thesis, Information Systems Department, METU, Ankara.

URL 1, http://en.wikipedia.org/wiki/CUDA, 14 April 2012.