# Fast Non-Iterative PCA computation for spectral image analysis using GPU

***Jukka Antikainen[1], Markku Hauta-Kasari[1], Timo Jaaskelainen[2] and Jussi Parkkinen[1], School of Computing[1], Department of Physics and Mathematics[2], University of Eastern Finland, Finland.***

## Abstract

*In this study, we implement a fast non-iterative Principal Component Analysis computation for spectral image analysis by utilizing Graphical Processing Unit GPU. PCA inner product computation efficiency between Central Processing Unit CPU and GPU was examined. Performance was tested by using spectral images with different dimensions and different PCA inner product image counts. It will be shown that the GPU implementation provides about seven times faster PCA computation than the optimized CPU version. Difference to the commonly used scientific analysis software Matlab is even higher. When spectral image analysis is needed to make in real-time, CPU does not offer the necessary performance for larger spectral images. Therefore, powerful GPU implementation is needed.*

## Introduction

Principal Component Analysis (PCA) is commonly used in spectral data analysis [1,4]. Usually PCA computation is not so heavy to compute, but when it is needed to do in real-time, CPU computation efficiency is not enough. For example, when measuring is done on the industrial line with a spectral camera, the spectral image analysis must be fast and efficient. Therefore the GPU computation can be taken into consideration.

The graphical processing unit (GPU) was primary developed for gamers use. However the power of the GPU was also expanded for the scientific use and now its exploitation is rapidly increasing in many complex computational tasks [2, 3, 5, 6, 7].

One GPU card can contain several hundreds of streaming processors with thousands of threads which can be utilized to concurrent calculations. This enables very high speed computation for parallel tasks. The latest GPU graphics cards can achieve hundreds of GFlops per second computation power. The GPU computation unit Tesla S1070 can achieve massive 4 TFlops of computation power by utilizing four GPU cards in one processing unit. Therefore it is very reasonable to be used in scientific computations.

This paper will show how the non-iterative Principal Component Analysis is performed and how it is implemented in the GPU. Implementation performance is tested by using two different GPU cards. Results are compared with highly optimized CPU computation and to the generally used scientific computation tool Matlab.

## Principal Component Analysis for Spectral Images

The general implementation of the PCA algorithm for the spectral images is as follows. One spectrum *s* from the spectral image is defined as

$$s(\lambda) = \{s(\lambda_1)s(\lambda_2)\ldots s(\lambda_n)\}^T, \tag{1}$$

where *n* is equal to the number of wavelengths in the spectral image, λ is the wavelength component and *T* is a matrix transpose. The spectral image, which is normally defined in 3D form, has to be transformed. The spectral image is transformed from 3D-spectral image to 2D-spectral image to column wise order where one pixel wavelengths of the spectral image are forming one column of the matrix *S*

$$S = \begin{pmatrix} s_1(\lambda_1) & \cdots & s_m(\lambda_1) \\ \vdots & \ddots & \vdots \\ s_1(\lambda_n) & \cdots & s_m(\lambda_n) \end{pmatrix}, \tag{2}$$

where *m* is the number of pixels in an image. After the spectral image is transformed to 2D-image, correlation matrix *R* is computed as follows

$$R = \frac{1}{m} SS^T. \tag{3}$$

Since the correlation matrix is formed, eigenvalues and eigenvectors (PCA components) can be computed by solving the following equation

$$R\Phi = \sigma\Phi, \tag{4}$$

where $\Phi$ is the matrix of eigenvectors as columns and $\sigma$ is the unity matrix with eigenvalues in diagonal. Calculated eigenvectors are sorted into decreasing order by using information from the eigenvalues. Eigenvector which corresponds to the highest eigenvalue corresponds to the mean vector of the spectral image. The second highest eigenvector is orthogonal to the first one and it describes the second best approximation to the spectral image and so on. These eigenvectors forms the base of the PCA components. A wanted number $\eta$ of the PCA components are selected from the base vectors *B*.

$$B = \begin{pmatrix} b_1(\lambda_1) & \cdots & b_1(\lambda_n) \\ \vdots & \ddots & \vdots \\ b_\eta(\lambda_1) & \cdots & b_\eta(\lambda_n) \end{pmatrix}. \tag{5}$$

Inner product images are calculated by using selected components and transformed 2D-spectral image with following equation

$$P = B^T S. \tag{6}$$

It is easy to see from these equations that there are a couple of parts in the PCA computation which can be computed parallel and efficiently by using GPU. First part is the correlation matrix and the second part is inner product image computation which is based on matrix multiplication.

## GPU Architecture and Programming Model

The efficiency of GPU computation in parallel tasks is based on different architecture compared to the CPU's architecture. In Figure 1, the architectures of CPU and GPU are represented. In the GPU, there are multiple streaming processor units and each unit contains various registries which are very fast to use. Each streaming processor uses the same global GPU memory and the same program can be executed with all the processors simultaneously.

Compared with the CPU architecture, it is clear that the GPU architecture is developed for highly parallel and multi threaded computational tasks. Therefore it is reasonable to use GPU for parallel computational tasks rather than CPU.

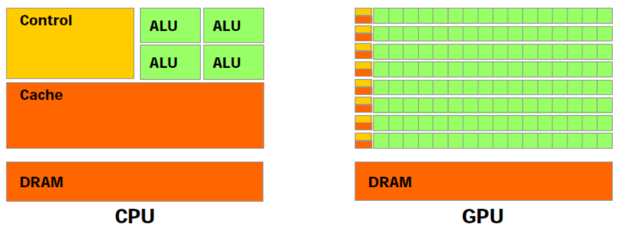

***Figure 1.*** *CPU and GPU architectures. [8]*

The programming model and the memory management in GPU is different than in CPU. In GPU, the data processing is divided into grids, blocks and multiple threads which are demonstrated in Figure 2. First the computation data is divided into the grid of blocks. Each block contains multiple threads which are executed parallel. Each thread has its own private memory and each block has its own memory which all threads in the block can use. Also all the threads can use the same global shared memory of the GPU card. [8]

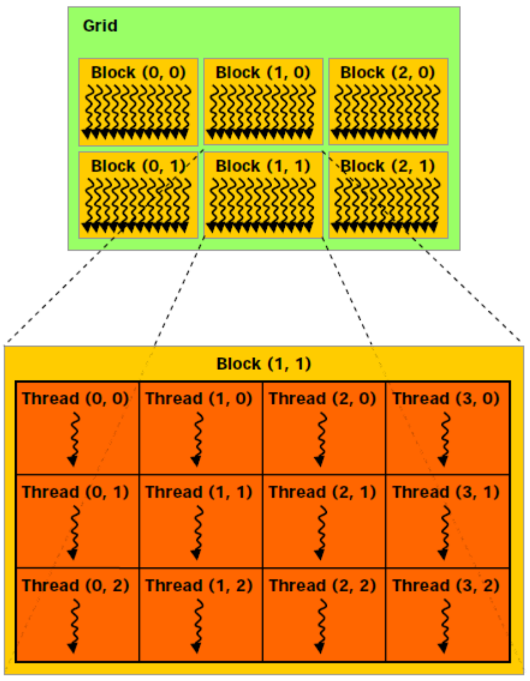

***Figure 2****. Grid with thread blocks [8]*

Because of the different GPU programming model, all computation algorithms should be developed and planned to work in suitable approach to achieve good efficiency. Figure 3 demonstrates the basic idea of matrix multiplication in GPU where matrices are computed by using $N$ x $N$ blocks. Each block with $N^2$ threads computes one sub block of the result matrix. Each thread computes only one value of one sub block. Therefore matrix multiplication can be parallelized very efficiently in GPU.

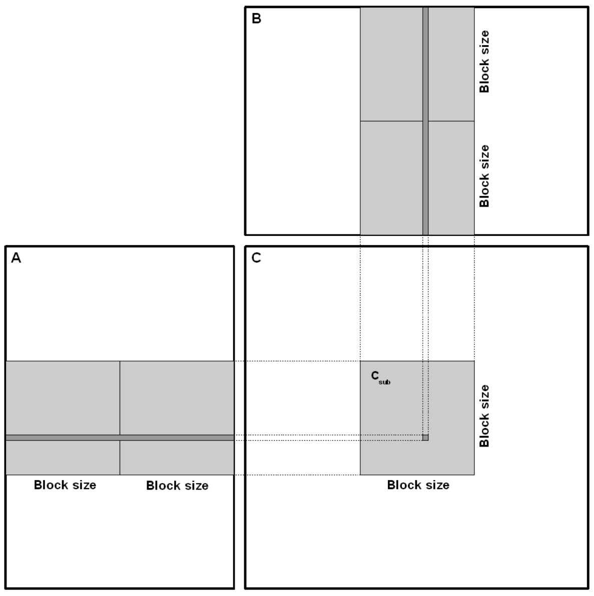

***Figure 3.*** *The basic method of matrix multiplication in GPU. [8]*

Matrix multiplication can be also computed by using highly optimized NVIDIA CUBLAS library [9]. This library is based on very commonly used and efficient BLAS (Basic Linear Algebra Subprograms) library [11]. The CUBLAS library includes many other useful mathematical functions which are optimized for the GPU computation.

## GPU Implementation of Principal Component Analysis

Because of the fast matrix multiplication, the calculation of the correlation matrix is highly efficiency. The same method is used in the calculation of inner product images. These parts are the most time consuming tasks of the PCA computation for large spectral images. In Figure 4, one example of PCA inner product image computation is explained with *m* x *n* x *32* dimensional spectral image where the count of inner product images is reduced to 16.

Some parts are calculated in the GPU and some parts in the CPU. First the spectral image is converted to the 2D-spectral image and it is loaded into the graphic card's memory. Correlation matrix R is calculated (Eq. 3) by using 16 x 16 blocks. After the computation of the correlation matrix, the eigenvectors and the eigenvalues are calculated with CPU by using highly optimized LAPACK library [10].

Efficiency for eigenvector computation between the GPU and the CPU was evaluated. With Quad-core Xeon 3 GHz CPU, eigenvectors and eigenvalues were calculated in 0.37 ms when it took 0.40 ms with Quadro FX 3700 GPU. In this low dimensional 32 x 32 matrix case, the CPU algorithm works little faster than GPU implementation. Therefore CPU computation was used. Computation time difference for smaller matrices would be even more because of the memory transfers between the graphics card and the CPU. If larger matrices are used, the GPU implementation for the eigenvectors and the eigenvalues can be taken in consideration.

### *Algorithm for PCA computation and data transfers between the CPU main memory and the GPU memory*

First the spectral image is loaded in the main memory from the hard disk or directly from the spectral camera and then the converted 2D spectral image is transferred to the GPU memory. Then the correlation matrix is computed in the GPU and the result matrix is transferred to the CPU where eigenvectors $\Phi$ and eigenvalues $\sigma$ are calculated by using LAPACK. A wanted number of the PCA components are transferred back to the GPU and the inner product images are calculated. After inner product image calculation, images can be returned to CPU or image processing can be continued in the GPU. Next part shows the memory transactions between the CPU and the GPU.

All variables with a subscript index are defined in the GPU memory. Otherwise the variable is defined in the main memory of the CPU.

| **Algorithm** | **Memory processing** |
|---|---|
| 1. S = spectral image | (CPU) |
| 2. $S_d$ = ConvertTo2D(S) | (CPU → GPU) |
| 3. $R_d = (1/m)S_dS_d^T$ | (GPU) |
| 4. $R = R_d$ | (GPU → CPU) |
| 5. $[\Phi, \sigma]$ = SolveEig(R) | (CPU) |
| 6. $C_d = \Phi(1{:}\eta)$ | (CPU → GPU) |
| 7. $P_d = C_d^TS_d$ | (GPU) |
| 8. PP = ConvertTo3D($P_d$) | (GPU → CPU) |
| 9. process result PP | (CPU/GPU) |

If steps 5 to 7 are made in GPU, memory transfers between CPU and GPU can be reduced. Still in the small dimensional case it is still better to calculate eigenvectors and eigenvalues in the CPU and do the data transfers between GPU and CPU.

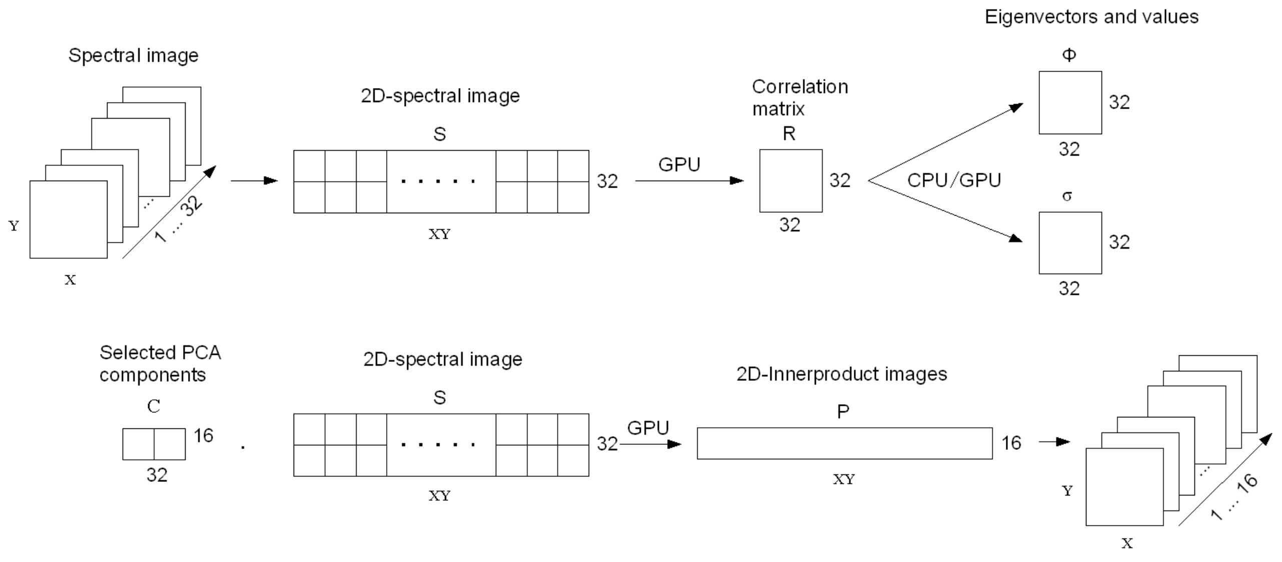

***Figure 4.*** *PCA inner product image calculation process for 32 dimensional spectral image*

## Results

The calculation program with C++ language was done by utilizing Visual Studio C++ 2005. To achieve optimized CPU code, source code was compiled by using efficient Intel C++ compiler which includes very good optimizations for the Intel processors. Actual GPU utilization was done by using NVIDIA CUDA 2.0 software library. GPU codes were compiled with NVIDIA compiler.

For the GPU computation, 16 x 16 block size with 256 threads was selected excluding the case where 8 PCA inner product images were computed. In that case, used block size was 8 x 8 with 64 threads. When using 16 x 16 block sizes in the GPU computation, the dimensions of the spectral image should be the factor of 16. Therefore we need to add a padding of zeros to the spatial and the spectral axis of the spectral image to achieve correct division if it is needed. This has no significant effect on the speed of our implementation.

PCA calculation was tested with different sizes of spectral images which were formed from the same spectral image. The formation of smaller spectral images was done by resizing the original image on spatial and spectral domains. The original size of the spectral image and its first six calculated PCA components are visualized in Figure 5. The spatial size of the spectral image was 800 x 800 pixels and the wavelength area was 31 dimensional from 420 nm to 720 nm by 10 nm steps. Therefore one extra wavelength channel was added to achieve 32 spectral dimensions which is needed for the efficient computation. First nine calculated inner product images from the spectral image are displayed in Figure 6.

The correlation between calculation times and the number of inner product images were measured. CPU and GPU calculation times were measured with Quad-core Intel Xeon 3GHz processor, NVIDIA Quadro FX 3700 and NVIDIA GeForce GTX280 graphics cards. Efficiency of the Matlab 2008a was also measured. NVIDIA Quadro FX 3700 uses 112 parallel processor cores with 512 MB of graphics memory with 51.2 GB/sec of memory bandwidth. The GeForce GTX280 uses 240 parallel processor cores with 1GB of graphics memory with 141.7 GB/sec of memory bandwidth which is almost three times faster than in Quadro FX 3700.

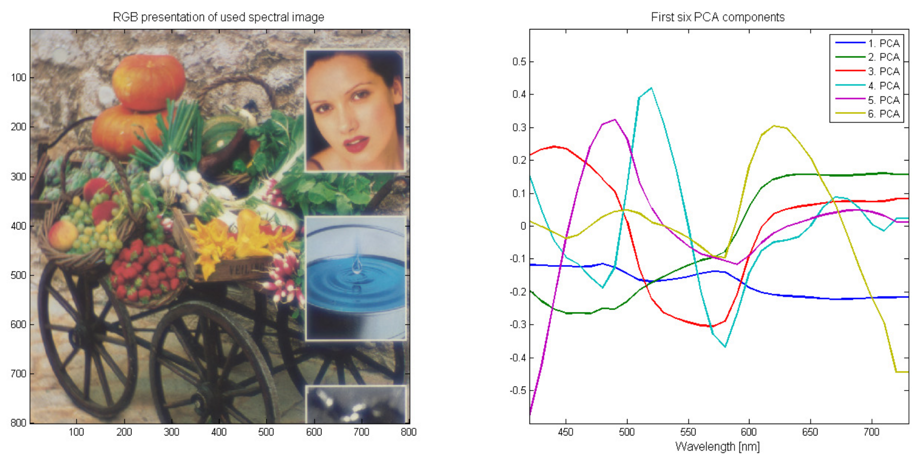

***Figure 5.*** *The sample spectral image and the first six calculated PCA components.*

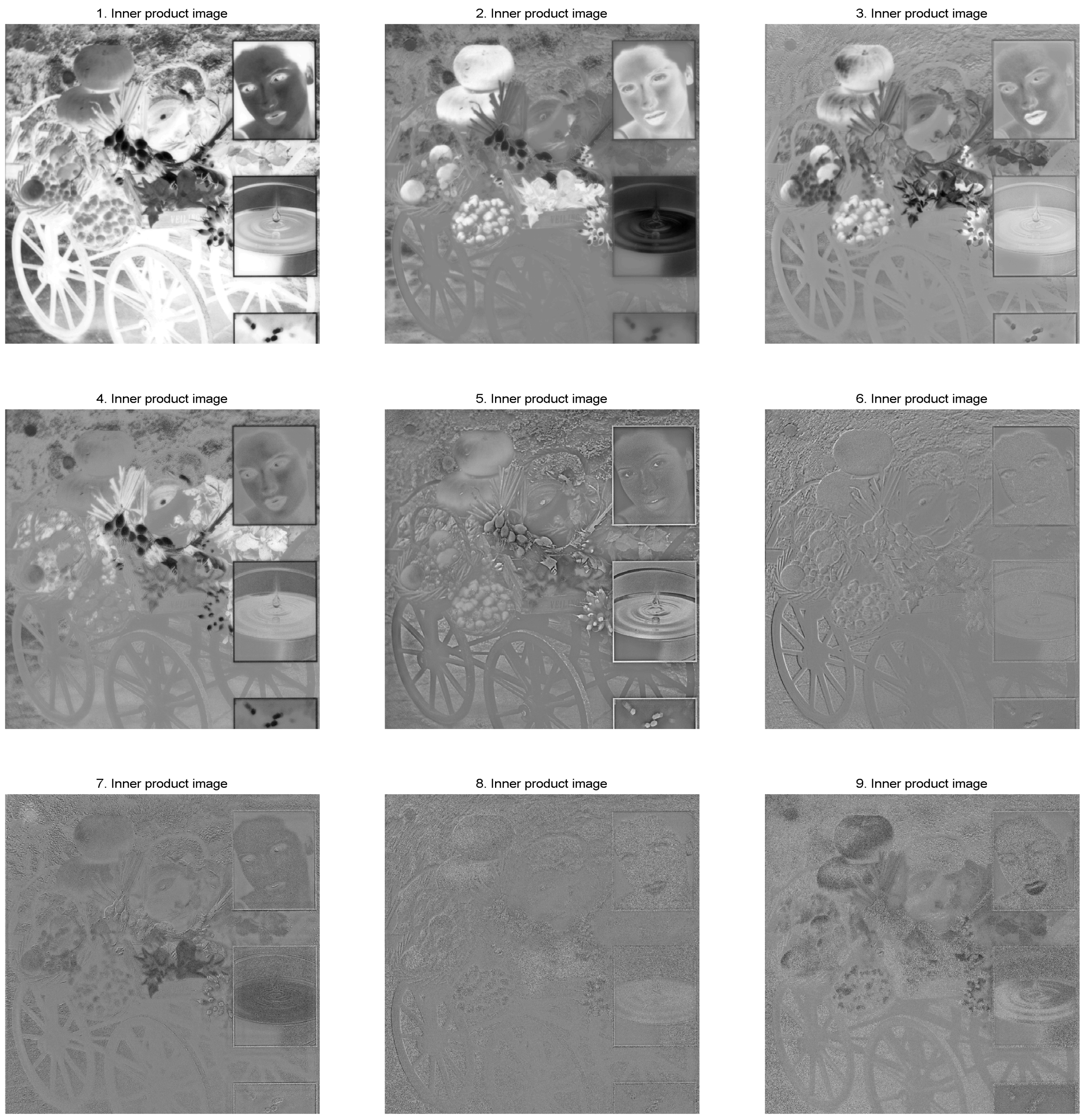

***Figure 6.*** *First nine PCA inner product images.*

From the Table 1, we can see that PCA inner product image computation in the GPU is very efficient in compared with highly optimized multi threaded C++- version and Matlab. This can be also seen from Figure 7, where computation time was measured from different sizes of spectral images with 8, 16 and 32 inner product images.

All computation times has been measured by using a timer which is provided by CUDA library and the timer was started after the spectral image was loaded in the main memory to minimize the effect of the hard disk.

**Table 1. Computation times of 32 PCA inner product images with different dimensional spectral images in milliseconds**

| Dimensions | Size in MB | Matlab | Xeon 3GHz (optimized) | Quadro FX 3700 | GeForce GTX280 |
|---|---|---|---|---|---|
| 80 x 80 x 32 | 0.78 | 18.8 | 10.87 | 4.0 | 2.6 |
| 160 x 160 x 32 | 3.13 | 64.1 | 24.2 | 7.5 | 5.4 |
| 240 x 240 x 32 | 7.03 | 143.8 | 53.0 | 14.6 | 8.6 |
| 320 x 320 x 32 | 12.50 | 251.6 | 94.3 | 23.2 | 14.9 |
| 400 x 400 x 32 | 19.53 | 395.3 | 147.7 | 35.7 | 22.7 |
| 480 x 480 x 32 | 28.13 | 567.2 | 215.1 | 50.1 | 33.0 |
| 560 x 560 x 32 | 38.28 | 768.8 | 294.1 | 67.0 | 43.5 |
| 640 x 640 x 32 | 50.00 | 1006.3 | 378.3 | 88.2 | 57.9 |
| 720 x 720 x 32 | 63.28 | 1273.4 | 480.5 | 109.6 | 71.4 |
| 800 x 800 x 32 | 78.13 | 1568.8 | 615.1 | 135.2 | 89.5 |

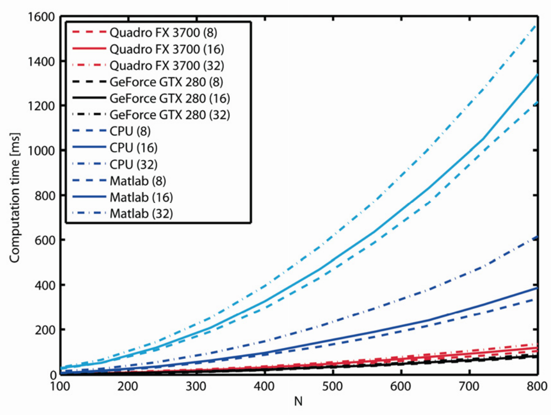

***Figure 7.*** *Computation times for the different number of PCA inner product images by using different computation devices.*

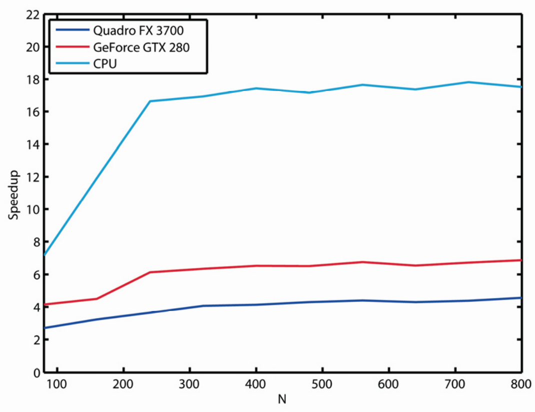

***Figure 8.*** *Computation speedups for different computation devices*

From Figure 7, we can see that computation times raise very high with Matlab and even with C-program for larger matrices. Computation time for the largest test image is about 7 times faster in GTX280 than in CPU which can be seen from the Figure 8. By using Matlab, difference to GTX280 is over 17 times faster. Differences between Quadro FX 3700 and GTX280 cards are not so massive.

The Results show that when the real-time (25fps) computation is needed, CPU can only process 160x160x32 dimensional spectral images, while GTX280 card can process 480x480x32 dimensional spectral images. The spatial size of the spectral image which CPU can compute on real-time is nine time smaller than the spectral image which GPU can process on real-time. This result is very good for this implementation.

## Discussion

We have developed and tested a very high performance implementation of PCA for spectral image analysis. Speed of the computation was tested with different dimensional spectral images. With GPU, we can achieve real-time computation for 480x480x32 dimensional spectral image when CPU can only achieve real-time performance for 160x160x32 dimensional spectral image. This is very good and an important result for this research and for our future work.

We have started to implement real-time PCA computation for microscopy surgery camera video which is used in brain surgeries. The video data can contain three or more channels. Therefore this study is relevant to the future studies. Also spectral image estimation from the RGB-video will be examined by utilizing the efficiency of the GPU.

## References

[1] J. P. S. Parkkinen, J. Hallikainen and T. Jaaskelainen, “Characteristic spectra of Munsell colors”, Journal of Optical Society of America, Vol. 6, No. 2/February 1989

[2] Julius Ohmer, Frederic Maire and Ross Brown, ”Implementation of Kernel Methods on the GPU”, in Proceedings of the Digital Image Computing on Techniques and Applications, 2005

[3] M. Andrecut, ”Parallel GPU Implementation of Iterative PCA Algorithms”, Journal of Computational Biology, 2008.

[4] Martinkauppi, B., Lehtonen, J. and Parkkinen, J.: “Near-infrared images of skin”, CGIV 2008/ MCS'08, 4th European Conference on Colour in Graphics, Imaging, and Vision, 10th International Symposium on Multispectral Colour Science, Terrassa-Barcelona, España, June 9-13, 2008, pp. 508-511, ISBN 978-0-89208-2626.

[5] Nathan Bell and Michael Garlandy, “Efficient Sparse Matrix-Vector Multiplication on CUDA”, NVIDIA Technical Report NVR- 2008-004, Dec. 2008.

[6] Youquan Liu; Xuehui Liu; Enhua Wu, “Real-time 3D fluid simulation on GPU with complex obstacles”, Computer Graphics and Applications, Proceedings of 12th Pacific Conference, 6-8 Oct. 2004 Page(s): 247 - 256, 2004.

[7] Ek, L. A., Vistnes, R., and Gundersen, O.E., “Animating physically based explosions in real- time”, In Proceedings of the 5th international Conference on Computer Graphics, Virtual Reality, Visualisation and interaction in Africa (Grahamstown, South Africa, October 29 - 31, 2007) S. N. Spencer, Ed. AFRIGRAPH '07. ACM, New York, NY, 61-69.

[8] NVIDIA CUDA. CUDA Programming Guide 2.3.1, August, 2009, NVIDIA Corporation, http://www.nvidia.com/

[9] Documentation for CUDA BLAS (CUBLAS) Library, March, 2008, NVIDIA Corporation, http://www.nvidia.com/

[10] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. LAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[11] J. Dongarra, Basic Linear Algebra Subprograms Technical Forum Standard, International Journal of High Performance Applications and Supercomputing, 16(1) (2002), pp. 1-111, and International Journal of High Performance Applications and Supercomputing, 16(2), pp. 115—199, 2002.

## Biography

*Jukka Antikainen received his M.Sc. degree in Computer Science in 2006 from the University of Kuopio, Finland. The subject of the master thesis was parallelization of matrix operations. Currently he is working with his doctoral thesis with the School of Computing at the University of Eastern Finland. His thesis is concerning spectral color research in industrial applications.*