GPU-BASED PARALLELIZATION ON PRINCIPAL COMPONENTS TRANSFORMATION

Victor J. D. Tsai and Chia-Wei Tsai

Department of Civil Engineering, National Chung Hsing University, Taiwan, jdtsai@nchu.edu.tw

ABSTRACT ... Techniques in band selection are usually used to select a subset of highly correlated data without losing their physical meaning for dimensionality reduction purpose. Among these techniques, principal components transformation (PCT) is the most commonly used in finding a new set of orthogonal bases (principal axes) that better captures spectral characteristics with the variance of transformed data in descending order. Considering the high parallel computing capability of current multi-core Graphics Processing Units (GPUs), this research aims on developing a GPU-based parallel processing approach for PCT using C/C++ and NAVIDIA's CUDA. A 191-band HYDICE hyperspectral image was used in the experiments for illustrating that the developed approach accelerates PCT in an overall speedup of 30× through the high performance computing capability of GPU.

KEY WORDS: Principal Components, Parallel Computing, GPU, CUDA, Hyperspectral Imagery

1. INTRODUCTION

There is always encountered with problems of high correlation among neighbouring spectral bands, which deliver similar spectral information, when using hyperspectral imagery with hundreds of spectral bands. Techniques in band selection are usually used to select a subset of these highly correlated data without losing their physical meaning for dimensionality reduction purpose (Yang *et al.*, 2011; Ifarraguerri, 2004; Bajcsy & Groves, 2004; Huang & He, 2005; Du & Yang, 2008). Among these techniques, principal components transformation (PCT) is the most commonly used in finding a new set of orthogonal bases (principal axes) that better captures spectral characteristics with the variance of transformed data in descending order (Richards & Jia, 1999; Schowengerdt, 1997). However, PCT demands intensive computations for the covariance matrix of the original data set and the solution for corresponding eigenvalues and eigenvectors, which are too time consuming in serial mode of central processing unit (CPU).

Recently, graphics processing units (GPUs) are of great interest to high-performance computing (HPC) community for strong parallel processing capability, multithread, many-core processors with tremendous computational speed, and extremely high memory bandwidth (Tao *et al.*, 2012; Nickolls & Daily, 2010). Although GPUs are originally specified for computer graphics, they are now popular for general-purpose computing and HPC for remote sensing (Yan *et al.*, 2011; McCool, 2007; Lee *et al.*, 2011; Plaza *et al.*, 2011). Meanwhile, with the NVIDIA's release of Compute Unified Device Architecture (CUDA) in 2006 (NVIDIA, 2011a, 2011b; Harris, 2007), image processing algorithms can be implemented in C/C++ language and executed in parallel to a large number of threads in GPU hardware without the use of graphics API (application program interface) such as OpenGL and Direct X. A CUDA compliant GPU exposes three novel hardware features: general load-store memory architecture, on-chip shared memory, and thread synchronization (Senguta *et al.*, 2007). Hence, CUDA provides highly data-parallel processing for GPUs with better performance than existing parallel approaches based on SIMD (single instruction multiple data) or MIMD (multiple instruction multiple data) stream structures.

This research aims to approach an efficient application of parallel computing algorithms for PCT using CUDA compliant GPUs. The remainder of this paper is organized as follows: Section 2 reviews the computing principles in PCT, while Section 3 provides an overall description on the parallelization of basic operations in PCT using CUDA. The experimental results from the PCT of a 191-band HYDICE hyperspectral imagery by parallel GPU approaches and serial CPU approaches are presented and discussed in Section 4. Finally, the conclusions are drawn in Section 5 with the suggestions on future work.

2. COMPUTING PRINCIPLES IN PCT

PCT is a feature space transformation designed to remove the high spectral redundancy in multispectral and hyperspectral image bands with high correlation due to material spectral correlation, topography, and sensor band overlap (Richards & Jia, 1999; Schowengerdt, 1997; Ready & Wintz, 1973). For a pixel $\boldsymbol{x} = [f_1 \quad f_2 \quad \cdots \quad f_L]^t$ (^{t}: transpose operation) in an L-band image f with $K = M \times N$ pixels in each band, PCT is a zero-correlation rotational transform of the following type with image-specific matrix $\boldsymbol{G}$,

$$\boldsymbol{y} = \boldsymbol{G}\,\boldsymbol{x} \qquad (1),$$

subject to the constraint that the covariance matrix of the transformed pixel data in the new $\boldsymbol{y}$ space is diagonal. It is desired to find the linear transformation matrix $\boldsymbol{G}$ in the following procedures (Schowengerdt, 1997; Richards & Jia, 1999):

A. Computing the mean vector $\boldsymbol{m}$ in the original $\boldsymbol{x}$ space:

$$m = \frac{1}{K}\sum_{i=1}^{K} x_i \quad (2),$$

B. Computing the covariance matrix $\boldsymbol{C_x}$:

$$C_x = \frac{1}{K-1}\sum_{i=1}^{K}(x_i - m)(x_i - m)^t \quad (3),$$

C. Solving for the eigenvalues and eigenvectors of $\boldsymbol{C_x}$:

$$|C_x - \lambda I| = 0 \quad (4),$$

where $\boldsymbol{I}$ is the (diagonal) identity matrix. It is noted that the eigenvalues are sorted in descending order, i.e., $\lambda_1 > \lambda_2 > \cdots > \lambda_L$, each eigenvalue is equal to the variance of the respective band in $\boldsymbol{y}$ space, and the sum of all the eigenvalues is equal to the trace of $\boldsymbol{C_x}$. Then, the eigenvector g_i for the i[th] eigenvalue λ_i can be determined by solving for the following equation:

$$(C_x - \lambda_i I)\cdot g_i = 0 \quad (5),$$

where $g_i = [g_{i1} \quad g_{i2} \quad \cdots \quad g_{iL}]^t$. However, the equation systems in Eq. (5) for each eigenvalue are not independent. The orthogonality constraint of the resulting matrix $\boldsymbol{G}$ adds the following characteristic equation that can be solved simultaneously with Eq. (5) for g_i,

$$g_{i1}^2 + g_{i2}^2 + \cdots + g_{iL}^2 = 1 \quad (6).$$

As results, the transformation matrix $\boldsymbol{G}$ is composed:

$$G = \begin{bmatrix} g_1^t \\ g_2^t \\ \vdots \\ g_L^t \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1L} \\ g_{21} & g_{22} & \cdots & g_{2L} \\ \vdots & \vdots & \vdots & \vdots \\ g_{L1} & g_{L2} & \cdots & g_{LL} \end{bmatrix} \quad (7).$$

D. Transform the original image using Eq. (1) with the first P ($\leqq L$) components. A linear contrast stretch may apply to scale the dynamic range of the transformed image into 8-bit data:

$$y_{out} = \frac{255}{y_{\max} - y_{\min}}(y - y_{\min}) \quad (8).$$

3. PARALLELIZATION OF PCT IN CUDA

To alleviate the computational burden of PCT, it is attractive and desirable to implement such algorithms in parallel using CUDA compliant GPUs. In CUDA programs, a large number of threads can be dynamically executed in parallel *Kernel* functions that run in scalable GPU cores through three layers of hierarchical architectures: thread, block, and grid (NVIDIA, 2011a, 2011b). CUDA programs thus provide heterogeneous CPU+GPU coprocessing for achieving optimal performance of applications that use the right core for a right job: a single-latency optimized CPU core for serial portions and throughput-optimized GPU cores for parallel portions of code (Nickolls & Daily, 2010).

Figure 1 describes the flowchart of our CPU+GPU PCT implementation using CUDA v4.1. The CPU serial codes are designed to load the L-band hyperspectral imagery, with $K = M \times N$ pixels in each band, into host (CPU) memory and solve for the L eigenvalues and eigenvectors of its corresponding covariance matrix $\boldsymbol{C_x}$. The GPU parallel codes are designed to map the imagery data onto the device (GPU) shared memory in an $L' \times K'$ matrix $\boldsymbol{X}$ for the computations of the mean vectors, covariance matrix, and final linear transformation, where $L' = Ceiling(L/16) \times 16$ and $K' = Ceiling(K/16) \times 16$ for optimal parallelization, with *Ceiling(r)* representing the smallest integer that is greater than or equal to r. Considering the examples in (NVIDIA, 2011a, 2011b), five *Kernels* of basic arithmetic and matrix operations in PCT are designed as described in the following.

A. Kernel 1: computation of the sum of a 1-D array

A block, consisting of 1024 threads, is scheduled for each band of the imagery. The tree reduction approach (Harris, 2007) for computing the sum of all elements in a 1-D array was applied for each band in *Ceiling*(K'/1024) iterations.

B. Kernel 2: division of all elements by a constant

The division of all elements in a 1-D array with K' elements by a constant D can be parallelized by using 1024 threads in *Ceiling*(K'/1024) iterations.

Therefore, the computation of the mean value for each band of the imagery is a combination of *Kernels 1* and *2*.

C. Kernel 3: subtraction of a constant for all elements

Similar to *Kernel 2*, subtraction of a constant D from all elements in a 1-D array with K' elements in GPU shared memory is also parallelized by using 1024 threads in *Ceiling*(K'/1024) iterations.

The mean value for the i[th] band of the imagery, i.e., m_i, is then subtracted from each element in the i[th] row of the $\boldsymbol{X}$ matrix using *Kernel 3*, resulting in the mean-shifted matrix $\boldsymbol{X'}$.

D. Kernel 4: transpose of a matrix

As described previously in Step B of PCT computing principles in Sec. **2**, the computation of the covariance matrix is then decomposed into three steps in the programs:

1. Transpose the $\boldsymbol{X'}$ matrix as $\boldsymbol{X'}^t$,
2. Multiply $\boldsymbol{X'}$ with $\boldsymbol{X'}^t$, resulting as $\boldsymbol{C_x}$,
3. Divide each element of $\boldsymbol{C_x}$ by K-1.

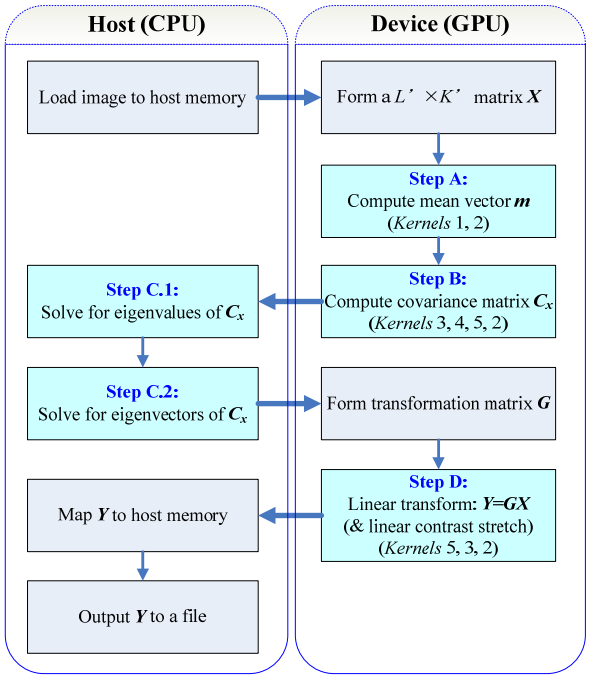

Figure 1. Flowchart of our CPU+GPU implementation.

Kernel 4 is designed to transpose the $\boldsymbol{X'}$ matrix in GPU's shared memory by using *Ceiling*(*L'*/16)× *Ceiling*(*K'*/16) blocks, each with *16×16* threads for the parallel swapping in symmetric elements about the diagonal (NVIDIA, 2011b).

E. Kernel 5: matrix multiplication

The multiplication of the $\boldsymbol{X'}$ and $\boldsymbol{X'^t}$ matrices is implemented in *Kernel 5* using *Ceiling*(*L'*/16)× *Ceiling*(*K'*/16) and *Ceiling*(*K'*/16)×*Ceiling*(*L'*/16) blocks, each with *16×16* threads respectively, following the computational notations in matrix multiplication. Figure 2 shows the schematic model for the multiplication of a matrix A (*64×128*) and its transpose matrix B (*128×64*) and the resulting matrix C (*64×64*), using sub-blocks of *16×16* threads in parallel computation.

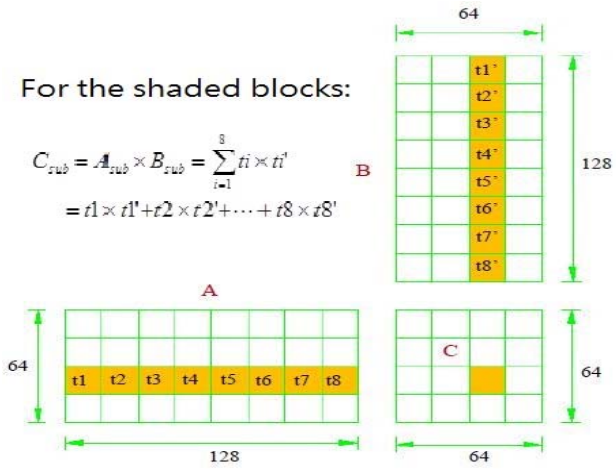

Figure 2. Scheme for parallel matrix multiplication.

The computation of the *L'×L'* covariance matrix $\boldsymbol{C_x}$ of the *L*-band imagery is then a combination of *Kernels 4, 5, and 3* (with constant value of *K*-1) in the GPU device. The solution for the eigenvalues of $\boldsymbol{C_x}$ is currently done in serial CPU model by applying QR decomposition (Francis, 1961; Burden & Faires, 2011), followed by a quicksort (Hoare, 1962) in descending order. The eigenvectors for the first *P* ($\leqq L$) bands are also solved in CPU host for compositing a *P×L'* transformation matrix $\boldsymbol{G}$. The selected *P*-band component of $\boldsymbol{G}$ matrix is then mapped onto the GPU device for linear transformation in Eq. (1) using *Kernel 5*, followed by a linear contrast stretch in Eq. (8) using *Kernels 3* and *2*.

4. EXPERIMENTAL RESULTS

The programs were developed using Microsoft Visual C++ and CUDA v4.1 under Microsoft Windows 7 64bits environment. In addition to the CPU+GPU coprocessing programs for PCT, the counterpart codes for CPU serial operations are also implemented for performance comparison in an ASUS ESC1000 computer with hardware specificity shown in Table 1. A 191-band, in the size of *1280×307*, 16-bit HYDICE image of Washington DC Mall from Purdue University (https://engineering.purdue.edu/~biehl/MultiSpec/) was used in the experiment.

The performances of our CPU+GPU parallelization and CPU serial process on PCT are shown in Table 2. The parallelization on the key steps A, B, and D can get speedup of up to 86×, 58×, and 94×, respectively, with overall speedup of 30×, compared to the CPU-based process. The transformed images from both CPU+GPU parallelization and CPU serial process were also examined with the root-mean-squared error, correlation coefficient, and mean absolute error, using the result from the CPU serial process as reference. It is shown that corresponding bands from both processes are identical due to unique eigenvectors solved in the same routines.

5. CONCLUSIONS

This research adapts NVIDIA's Compute Unified Device Architecture (CUDA) in designing CPU+GPU parallelized programs for principal components transformation (PCT), and compares the computational efficiency with the counterpart CPU single-thread programs. We focused on the parallelization of basic arithmetic and matrix operations in PCT. A 191-band *1280×307* HYDICE hyperspectral image was used in the experiments for illustrating that the CPU+GPU approach accelerates PCT operation in an overall speedup of 30× through the parallel computing capability of GPU. Individual parallelized steps may get speedup of up to 86×, 58×, and 94× for computing mean vectors, covariance matrix, and linear transformation, respectively. The concept in parallelism of basic operations is anticipated for near real-time high performance processing of volumetric image data in remote sensing applications.

Table 1. Hardware specification in the experiment.

Item	CPU	GPU
Brand/Model	Intel Xeon W3530	NVIDIA Tesla C0270
Number of Cores	6	448
Core Frequency	2.8 GHz	1.15 GHz
Memory	24576 MB DDR3	2688 MB GDDR5
Memory Bandwidth	25.6 GB/s	144 GB/s

Table 2. Performance comparison on HYDICE imagery.

Step	Computations	CPU (sec)	GPU (sec)	Speedup
A	Mean vector	1.732	0.020	86.60
B	Covariance matrix	72.097	1.225	58.86
C	Eigenvalues & eigenvectors	0.107	0.112*	-
D	Transformation of the first 16 PCs with contrast stretch	8.681	0.092	94.36
	Memory mapping & others	0.101	1.254	-
Overall		82.718	2.703	30.60

* : executed on CPU with time for memory mapping between host and device

ACKNOWLEDGEMENTS

This research was supported by the National Science Council, Executive Yuan, Taiwan, under the contract grant NSC 100-2221-E-005-075-MY2.

REFERENCES

Bajcsy, B. and P. Groves, 2004. Methodology for hyperspectral band selection. *Photogramm. Eng. Remote Sens.*, 70(7), pp. 793-802.

Burden, R. L. and J. D. Faires, 2011. *Numerical Analysis*, 9th ed., Brooks/Cole Cengage Learning.

Du, Q. and H. Yang, 2008. Similarity-based unsupervised band selection of hyperspectral image analysis. *IEEE Geosci. Remote Sens. Lett.*, 5(4), pp. 564-568.

Francis, J. G. F., 1961. The QR Transformation. *The Computer Journal*, 4(3), pp. 265-271.

Harris, M.,2007. *Parallel prefix sum (Scan) with CUDA*, NVIDIA.

Hoare, C. A. R., 1962. Quicksort. *The Computer Journal*, 5(1), pp. 10-16.

Huang, R. and M. He, 2005. Band selection based on feature weighting for classification of hyperspectral imagery. *IEEE Geosci. Remote Sens. Lett.*, 2(2), pp. 156-159.

Ifarraguerri, A., 2004. Visual method for spectral band selection. *IEEE Geosci. Remote Sens. Lett.*, 1(2), pp. 101-106.

Lee, C. A., S. D. Gasster, A. Plaza, C-I Chang, and B. Huang, 2011. Recent development in high performance computing for remote sensing: A review. *IEEE J. of Selected Topics in Applied Earth Observations and Remote Sens.*, 4(3), pp. 508-527.

McCool, M. D., 2007. Signal processing and general-purpose computing on GPUs. *IEEE Signal Process. Mag.*, 24(3), pp. 109-114.

Nickolls, J. and W. J. Daily, 2010. The GPU computing era. *IEEE Micro*, 30(2), pp. 56-69.

NVIDIA, 2011a. *NVIDIA CUDA C Programming Guide, Version 4.1*.

NVIDIA, 2011b. *NVIDIA GPU Computing SDK, Version 4.1*.

Plaza, A., Q. Du, Y-L Chang, and R. L. King, 2011. High performance computing for hyperspectral remote sensing. *IEEE J. of Selected Topics in Applied Earth Observations and Remote Sens.*, 4(3), pp. 528-544.

Ready, P. J. and P. A. Wintz, 1973. Information extraction, SNR improvement and data compression in multispectral imagery. *IEEE Trans. on Commun.*, COM-21(10), pp. 1123-1131.

Richards, J. A. and X. Jia, 1999. *Remote Sensing Digital Image Processing: An Introduction*, 3rd ed., Springer.

Schowengerdt, R. A., 1997. *Remote sensing: Models and methods for image processing*, Academic Press.

Senguta, S., M. Harris, Y. Zhang, and J. D. Owens, 2007. Scan primitives for GPU computing. In: *Graphics Hardware 2007*, San Diego, CA, USA.

Tao, Z., Y. Luo, K. Guo, and S. Zhao, 2012. Feature extraction of hyperspectral remote sensing in parallel computing research based on GPU. In: *The 4th Int. Conf. on Computational and Information Sciences (ICCIS 2012)*, Chongqing, China, pp. 570-573.

Yang, H., Q. Du, and G. Chen, 2011. Unsupervised Hyperspectral Band Selection Using Graphics Processing Units. *IEEE J. of Selected Topics in Applied Earth observation and Remote Sensing*, 4(3), pp. 660-668.