GPU IMPLEMENTATION OF SPATIAL PREPROCESSING FOR SPECTRAL UNMIXING OF HYPERSPECTRAL DATA

Jaime Delgado[*]*, Gabriel Martin*[**]*, Javier Plaza*[*]*, Luis Ignacio Jimenez*[*] *and Antonio Plaza*[*]

[*]Hyperspectral Computing Laboratory, University of Extremadura, Caceres, Spain.
[**]Instituto de Telecomunicações, Av Rovisco Pais 1, 1049-001, Lisbon, Portugal.

ABSTRACT

The integration of spatial information into spectral unmixing process has attracted much attention in recent years. Several approaches have been developed to incorporate spatial considerations into the endmember extraction/estimation procedure. Spatial preprocessing algorithms are one of the most commonly adopted techniques to guide endmember identification algorithms in terms of the spatial characteristics of the hyperspectral data. Particularly, spatial preprocessing algorithm (SPP) consists on a preprocessing technique that can be used prior to most of existing spectral-based endmember extraction process, thus promoting the selection of endmembers from the most spatially homogeneous regions of the data set. This paper presents a parallel implementation of SPP algorithm which is tested over two different graphic processing units (GPUs) architectures: NVidia^{TM}GeForce GTX 580 and NVidia^{TM}GeForce GTX 870M. Experimental validation using a hyperspectral data set collected by AVIRIS sensor shows that it is possible to achieve real-time performance.

Index Terms— Hyperspectral imaging, Spectral unmixing, Spatial preprocessing, GPU

1. INTRODUCTION

Spectral unmixing amounts at estimating the abundance of pure spectral signatures (called endmembers) in each mixed pixel of a hyperspectral image, where mixed pixels arise due to insufficient spatial resolution and other phenomena [1]. A challenging problem is how to automatically identify endmembers, as the presence of mixed pixels generally prevents the localization of pure spectral signatures in transition areas between different land-cover classes. A possible strategy to address this problem is to guide the endmember identification process to spatially homogeneous areas, expected to contain the purest signatures available in the scene [2, 3]. For this purpose, several spatial preprocessing methods have been used prior to endmember identification [4, 5, 6]. For instance, the spatial preprocessing (SPP) [4] introduces the spatial information in the endmember extraction process, so that the preprocessing can be combined with classic methods for endmember extraction. In this way, the endmembers can be obtained based on spatial and spectral features. However the inclusion of SPP in the unmixing chain increases the computational cost, and the overall processing time for a given scene [6]. In order to address this issue, several high performance computing architectures have been proposed in order to reduce the processing time of the hyperspectral linear spectral unmixing chain [7]. For instance, graphics processing units (GPUs) have been used in order to perform the linear spectral unmixing chain in real time [8, 9, 10]. However, no spatial preprocessing techniques have been developed using high performance computing architectures as of yet. The aim of this work is to develop several implementations of the SPP algorithm using GPUs to drastically reduce its processing time. The proposed implementation achieves real-time performance.

This work was supported by the Portuguese Science and Technology Foundation under Projects: UID/EEA/50008/2013 and SFRH/BPD/94160/2013.

2. SPATIAL PREPROCESSING (SPP)

The main idea behind the SPP framework is to estimate, for each input pixel vector, a scalar factor ρ which is intimately related to the spatial similarity between the pixel and its spatial neighbors, and then use this scalar factor to spatially weight the spectral information associated to the pixel [4]. Let $\mathbf{y}_{i,j}$ represents the pixel in spatial coordinates i, j. With this notation in mind, the scalar factor is calculated as follows:

$$\alpha(i,j) = \sum_{r=i-d}^{i+d} \sum_{s=j-d}^{j+d} \beta[r-i, s-j] \cdot \gamma[\mathbf{y}_{i,j}, \mathbf{y}_{r,s}], \quad (1)$$

where $\mathbf{y}_{i,j}$ is the pixel at spatial coordinates i, j for which we are calculating the scalar factor, and $\mathbf{y}_{r,s}$ are the spatial neighbours. Here d represents half of the window size, so that the window size $ws = 2 \cdot d + 1$. The γ function computes the spectral angle between the pixel and the neighbours and finally the β function computes a weight factor basing on the distance between the pixel and the neighbour. The spectral

angle is computed as follows:

$$\gamma[\mathbf{y}_{i,j}, \mathbf{y}_{r,s}] = \arccos \frac{< \mathbf{y}_{i,j}, \mathbf{y}_{r,s} >}{||\mathbf{y}_{i,j}|| \cdot ||\mathbf{y}_{r,s}||}, \quad (2)$$

where $< \cdot, \cdot >$ denotes the dot product between two vectors and $||\cdot||$ denotes de euclidean norm of a vector. As we can see in (3) the closest neighbours are given more relevance. Also the β function is normalized to sum to one as follows:

$$\beta(a, b) \propto \frac{1}{a^2 + b^2} \quad (3)$$

Once the scalar factor has been computed, every pixel is displaced to the simplex centroid depending on the scalar factor. Expressions (4) and (5) shows how to displace the image pixels depending on the scalar factor.

$$\rho(i, j) = (1 + \sqrt[2]{\alpha(i, j)})^2 \quad (4)$$

$$\mathbf{y}_{i,j}{}' = \frac{1}{\rho(i, j)}(\mathbf{y_{ij}} - \overline{\mathbf{c}}) + \overline{\mathbf{c}} \quad (5)$$

Here, $\overline{\mathbf{c}}$ is the simplex centroid, computed as the average of all the image pixels. $\mathbf{y}_{i,j}{}'$ is the new displaced pixel and $\mathbf{y}_{i,j}$ is the original pixel at the coordinates i, j. Finally, n_l and n_c are the number of lines and columns of the image, respectively.

3. GPU ARCHITECTURE

The architecture of a GPU can be seen as a set of multiprocessors (MPs). Each multiprocessor is characterized by a single instruction multiple data (SIMD) architecture, i.e., in each clock cycle each processor executes the same instruction but operating on multiple data streams. Each processor has access to a local shared memory and also to local cache memories in the multiprocessor, while the multiprocessors have access to the global GPU (device) memory. We can see a representation of the GPU physical model in Fig. 1. Unsurprisingly, the programming model for these devices is similar to the architecture lying underneath. GPUs can be abstracted in terms of a stream model, under which all data sets are represented as streams (i.e., ordered data sets). Algorithms are constructed by chaining socalled kernels which operate on entire streams and which are executed by a multiprocessor, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. The kernels can perform a kind of batch processing arranged in the form of a grid of blocks, where each block is composed by a group of threads that share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. We can see a representation of the GPU logical model in Fig. 2

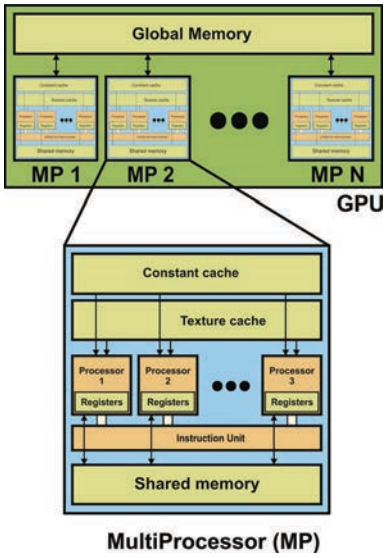

Fig. 1. Physical model of the GPU architecture.

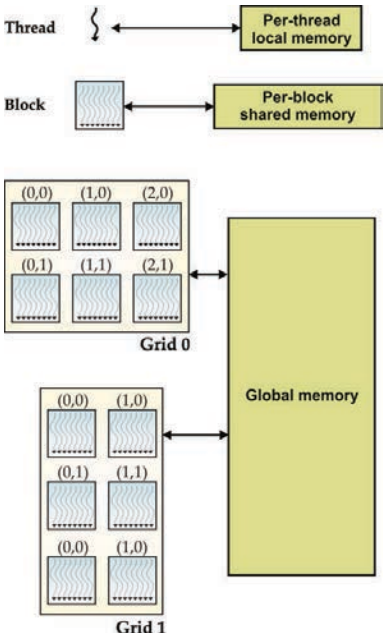

Fig. 2. Logial model of the CUDA GPU architecture.

4. GPU IMPLEMENTATIONS

In this section we describe the parallel implementations of SPP in the GPU. The parallel implementation of the algorithm is divided in four main kernels:

1. The first kernel computes the centroid of the simplex $\overline{\mathbf{c}}$. Here the number of blocks is equal to the number of bands and each block computes the average of each band using a reduction kernel.

2. The second kernel computes the euclidean norms of each image pixel that will be used to compute the γ function in (1). Here the number of threads T is set to the maximum value supported by the GPU. Each thread will compute the euclidean norm of a vector so the number of blocks B will be the number of image pixels divided by the number of threads: $B = \lceil (n_l \cdot n_c)/T \rceil$.

3. The third kernel computes the similarity factor $\alpha(i,j)$ for each pixel as given by the expression (1). In this kernel there are as many blocks as pixels: $B = n_l \cdot n_c$, and there are as many threads as the window size: $T = ws^2 = (2d+1)^2$. In this kernel each thread will compute the dot product between the central and the corresponding neighbour pixel in the window $< \mathbf{y}_{i,j}, \mathbf{y}_{r,s} >$, then computes $\gamma[\mathbf{y}_{i,j}, \mathbf{y}_{r,s}]$ as in (2). After that the kernel weights this value using the β function (precomputed using the CPU). Finally the kernel performs a reduction to sum all the values inside the window, as a result the kernel obtains $\alpha(i,j)$.

4. The fourth kernel computes the displacement to the centroid for each pixel as in (5). In this kernel each thread will compute the displacement of one pixel. The number of threads used is the maximum allowed by the GPU and the number of blocks is the number of image pixels divided by the number of threads $B = \lceil (n_l \cdot n_c)/T \rceil$.

5. EXPERIMENTAL RESULTS

The hyperspectral image scene used for experiments in this work was collected by the AVIRIS instrument, which was flown by NASAs Jet Propulsion Laboratory over the World Trade Center area in New York City on 16 September 2001, just 5 days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex [1]. The full data set selected for experiments consists of 614 $\times$ 512 pixels, 224 spectral bands, and a total size of (approximately) 140 Mbytes. The spatial resolution is 1.7 m/pixel. The leftmost part of Fig. 3 shows a false color composite of the data set selected for experiments using the 1.682, 1.107 and 655nm channels, displayed as red, green and blue, respectively.

[1] http://speclab.cr.usgs.gov/wtc/

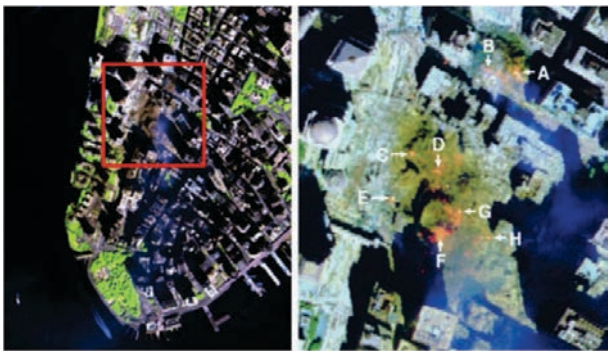

Fig. 3. False color composition of an AVIRIS hyperspectral image collected by NASA's Jet Propulsion Laboratory over lower Manhattan on 16 September 2001 *(left)*. Location of termal hot spot fires in World Trade Center area*(right)*.

The GPU implementation of P-SPP algorithm has been tested on two different computers, the first one was a desktop computer with a GPU NVidia^{TM}GTX 580, which features 512 processor cores operating at 1.54 GHz, with single precision floating point performance of 1581.1 Gflops, total dedicated memory of 1,536 MB, 2,004 MHz memory (with 384-bit GDDR5 interface) and memory bandwidth of 192.4 GB/s [2]. The GPU is connected to an Intel core i7 920 CPU at 2.67 GHz with eight cores, which uses a motherboard Asus P6T7 WS SuperComputer. The second computer is a laptop computer with a GPU NVidia^{TM}GTX 870M, which features 1344 processor cores operating at 967 MHz, with single precision floating point performance of 2599 Gflops, total dedicated memory of 3072 MB, 5000 MHz memory (with 192-bit GDDR5 interface) and memory bandwidth of 120 GB/s[3]. The GPU is connected to an Intel i7-4710MQ at 3.5 GHz with four cores.

It is important to emphasize that our GPU versions of P-SPP provide exactly the same results as the serial version of the SPP algorithm. Hence, the only difference between the serial and parallel algorithms is the time they need to complete their calculations. The serial algorithm was executed in one of the available cores of the desktop computer, and the parallel times were measured in the considered GPU platform. For each experiment, 10 runs were performed and the mean values are reported (these times were always very similar, with differences on the order of a few milliseconds only). Table I summarizes the obtained results by the C implementation and by the GPU implementation. An optimization has been considered for the CPU implementation, namely the inclusion of the –O3 optimization flag in the compiler.

At this point, it should be noted that the cross-track line scan time in AVIRIS, a push-broom instrument, is quite fast [11] (8.3 ms to collect 512 full pixel vectors). This introduces

[2] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580

[3] http://www.geforce.com/hardware/notebook-gpus/geforce-gtx-870m

Table 1. Mean execution times for the parallel and serial implementations of the SPP algorithm after 10 Monte-Carlo runs.

window size	3	5	7	9	11	13	15
SPP	7.86	18.63	38.87	77.72	141.80	211.27	286.97
P-SPP GTX580	0.69	0.71	0.78	0.87	1.06	1.32	1.62
Speedup	11.32	26.06	50.12	89.77	133.29	159.66	176.77
P-SPP GTX870M	0.98	1.03	1.06	1.29	1.60	2.07	2.47
Speedup	8.03	18.16	36.80	60.29	88.85	102.11	116.28

the need to process the considered scene (614×512 pixels and 224 spectral bands) in $<$5.09s to fully achieve real-time performance. As we can see in Table I, the execution times for the P-SPP algorithm is in real time with both GPUs for all the considered window sizes. The speedups increase when the windows size increases, which is expected due to the fact that the windows are processed in parallel and the bigger are the windows more computations are performed in parallel. It is important to note that in the best case the speedup is about 177 times faster than the normal version.

6. CONCLUSIONS

We have presented a GPU implementation of a spatial preprocessing (SPP) algorithm for including spatial information in spectral unmixing of hyperspectral data. The proposed implementation is scalable and highly efficient, achieving real-time results. Further work will be focused on embedding the spatial preprocessing into a full hyperspectral unmixing chain and the development of other implementations for the full chain on hardware devices such as FPGAs or multi-GPU systems.

7. REFERENCES

[1] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 5, no. 2, pp. 354–379, 2012.

[2] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, pp. 2025–2041, 2002.

[3] D. M. Rogge, B. Rivard, J. Zhang, A. Sanchez, J. Harris, and J. Feng, "Integration of spatial–spectral information for the improved extraction of endmembers," *Remote Sens. Environ.*, vol. 110, no. 3, pp. 287–303, 2007.

[4] M. Zortea and A. Plaza, "Spatial preprocessing for endmember extraction," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, pp. 2679–2693, 2009.

[5] G. Martin and A. Plaza, "Region-based spatial preprocessing for endmember extraction and spectral unmixing," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 4, pp. 745–749, 2011.

[6] G. Martin and A. Plaza., "Spatial-spectral preprocessing prior to endmember identification and unmixing of remotely sensed hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 5, no. 2, pp. 380–395, 2012.

[7] Sergio Bernabe, Sergio Sanchez, Antonio Plaza, Sebastián López, Jón Atli Benediktsson, and Roberto Sarmiento, "Hyperspectral unmixing on gpus and multicore processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, 2013.

[8] S. Sanchez, A. Paz, G. Martin, and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1538–1557, 2011.

[9] Sergio Sánchez, Rui Ramalho, Leonel Sousa, and Antonio Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on gpus," *Journal of Real-Time Image Processing*, pp. 1–15, 2012.

[10] José MP Nascimento, José M Bioucas-Dias, Jose M Rodriguez Alves, Vítor Silva, and Antonio Plaza, "Parallel hyperspectral unmixing on gpus," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 3, pp. 666–670, 2013.

[11] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis, M. R. Olah, and Orlesa Williams, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris)," *Remote Sens. Environ.*, vol. 65, pp. 227–248, 1998.