PARALLEL IMPLEMENTATION OF A HYPERSPECTRAL UNMIXING CHAIN: GRAPHIC PROCESSING UNITS VERSUS MULTI-CORE PROCESSORS

Sergio Bernabé[1], *Antonio Plaza*[1], *Sebastián López*[2] *and Roberto Sarmiento*[2]

[1]Hyperspectral Computing Laboratory, University of Extremadura, Caceres, Spain.
[2]Institute for Applied Microelectronic, University of Las Palmas de Gran Canaria, Spain.

ABSTRACT

Spectral unmixing is an important tast for remotely sensed hyperspectral imaging. Spectral unmixing algorithms can be computationally expensive. In this paper, we develop two efficient implementations of a full hyperspectral unmixing chain on two different kinds of high performance computing architectures: graphics processing units (GPUs) and multi-core processors. These platforms are inter-compared in the context of hyperspectral unmixing. Our experimental results reveal that both GPUs and multi-core processors can provide real-time unmixing performance when properly implemented in this kind of parallel platforms.

Index Terms— Hyperspectral imaging, spectral unmixing, high performance computing, GPUs, multi-core platforms.

1. INTRODUCTION

Hyperspectral imaging instruments are capable of collecting hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth. For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS), which is able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of reflected light in an area 2 to 12 kilometers wide and several kilometers long, using 224 spectral bands [1]. One of the main problems in the analysis of hyperspectral data cubes is the presence of mixed pixels [2], which arise when the spatial resolution of the sensor is not fine enough to separate spectrally distinct materials. In this case, several spectrally pure signatures (endmembers) are combined into the same (mixed) pixel. Spectral unmixing involves the separation of a pixel spectrum into its endmember spectra, and the estimation of the abundance value for each enmember. A popular approach for this purpose in the literature has been linear spectral unmixing, which assumes that the targets substances interact linearly within the field of view of the imaging instrument. In practice, the linear model is flexible and can be easily adapted to different analysis scenarios. Let $\mathbf{x}$ be a pixel vector given by a collection of values at different wavelengths. In the context of linear spectral unmixing, such vector can be modeled as:

$$\mathbf{x} \approx \mathbf{M}\boldsymbol{\alpha} + \mathbf{n} = \sum_{i=1}^{p} \mathbf{e}_i \alpha_i + \mathbf{n}, \quad (1)$$

where $\mathbf{M} = \{\mathbf{e}_i\}_{i=1}^{p}$ is a matrix containing p endmember signatures, $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_p]$ is a p-dimensional vector containing the abundance fractions for each of the p endmembers in $\mathbf{M}$, and $\mathbf{n}$ is a noise term. The spectral unmixing chain comprises two main steps: 1) endmember identification, in which the endmembers are often extracted from the image data; and 2) abundance estimation, in which the fractional coverage of each endmember is estimated for each pixel of the hyperspectral scene. The hyperspectral unmixing process is time consuming, but some applications require real-time responses for swift decisions which depend upon high computing performance, particularly for high-dimensional hyperspectral images. In this paper, we develop two efficient implementations of a full hyperspectral unmixing chain on two different kinds of high performance computing architectures: graphics processing units (GPUs) and multi-core processors [3]. These platforms are inter-compared in the context of hyperspectral unmixing. Our study reveals that both GPUs and multi-core processors can provide real-time unmixing performance.

2. UNMIXING CHAIN ALGORITHMS

2.1. Orthogonal subspace projection with Gram-Schmidt orthogonalization (OSP-GS) for endmember extraction

The orthogonal subspace projection (OSP) algorithm [4] was originally developed to find spectrally distinct signatures using orthogonal projections. In this work, we have developed an optimization of this algorithm which allows calculating the OSP without requiring the computation of the inverse of the matrix that contains the endmembers already identified in the

This work has been supported by the European Communitys Marie Curie Research Training Networks Programme under contract MRTN-CT-2006- 035927, Hyperspectral Imaging Network (HYPER-I-NET). Funding from the Spanish Ministry of Science and Innovation (CEOS-SPAIN project, reference AYA2011-29334-C02-02; DREAMS project, reference TEC2011-28666-C04-04) is also gratefully acknowledged.

image. This operation, which is difficult to implement in parallel, is accomplished using the Gram-Schmidt method for orthogonalization. This process selects a finite set of linearly independent vectors $\mathbf{A} = \{\mathbf{a}_1, \cdots, \mathbf{a}_p\}$ in the inner product space $\mathbf{R}^n$ in which the original hyperspectral image is defined, and generates an orthogonal set of vectors $\mathbf{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_p\}$ which spans the same *p-dimensional* subspace of $\mathbf{R}^n$ ($p \leq n$) as $\mathbf{A}$. In particular, $\mathbf{B}$ is obtained as follows:

$$\begin{array}{ll} \mathbf{b}_1=\mathbf{a}_1, & \mathbf{e}_1 = \frac{\mathbf{b}_1}{\|\mathbf{b}_1\|} \\ \mathbf{b}_2=\mathbf{a}_2 - proj_{\mathbf{b}_1}(\mathbf{a}_2), & \mathbf{e}_2 = \frac{\mathbf{b}_2}{\|\mathbf{b}_2\|} \\ \mathbf{b}_3=\mathbf{a}_3 - proj_{\mathbf{b}_1}(\mathbf{a}_3) - proj_{\mathbf{b}_2}(\mathbf{a}_3), & \mathbf{e}_3 = \frac{\mathbf{b}_3}{\|\mathbf{b}_3\|} \\ \mathbf{b}_4=\mathbf{a}_4 - proj_{\mathbf{b}_1}(\mathbf{a}_4) - proj_{\mathbf{b}_2}(\mathbf{a}_4) - proj_{\mathbf{b}_3}(\mathbf{a}_4), & \mathbf{e}_4 = \frac{\mathbf{b}_4}{\|\mathbf{b}_4\|} \\ \vdots & \vdots \\ \mathbf{b}_p=\mathbf{a}_p - \sum_{j=1}^{p-1} proj_{\mathbf{b}_j}(\mathbf{a}_p), & \mathbf{e}_p = \frac{\mathbf{b}_p}{\|\mathbf{b}_p\|}, \end{array} \quad (2)$$

where the projection operator is defined in Eq. (3), in which $< \mathbf{a}, \mathbf{b} >$ denotes the inner product of vectors $\mathbf{a}$ and $\mathbf{b}$.

$$proj_{\mathbf{b}}(\mathbf{a}) = \frac{< \mathbf{a}, \mathbf{b} >}{< \mathbf{b}, \mathbf{b} >}\mathbf{b}. \quad (3)$$

The sequence $\mathbf{b}_1, \cdots, \mathbf{b}_p$ in Eq. (2) represents the set of orthogonal vectors generated by the Gram-Schmidt method, and thus, the normalized vectors $\mathbf{e}_1, \cdots, \mathbf{e}_p$ in Eq. (2) form an orthonormal set. As far as $\mathbf{B}$ spans the same p-dimensional subspace of $\mathbf{R}^n$ as $\mathbf{A}$, an additional vector $\mathbf{b}_{p+1}$ computed by following the procedure stated at Eq. (2) is also orthogonal to all the vectors included in $\mathbf{A}$ and $\mathbf{B}$. This algebraic assertion constitutes the cornerstone of the proposed OSP method with Gram-Schmidt orthogonalization, referred to hereinafter as OSP-GS algorithm.

2.2. Unconstrained least-squares (UCLS) algorithm for abundance estimation

Once the set of endmembers $\mathbf{M} = \{\mathbf{e}_i\}_{i=1}^p$ have been identified, their correspondent abundance fractions $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_p]$ in a specific pixel vector $\mathbf{x}$ of the scene can be simply estimated (in least squares sense) by the following unconstrained expression:

$$\boldsymbol{\alpha} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{x}. \quad (4)$$

Two additional constrains can be imposed into the model described in Eq. (4), these are the abundance non-negativity constraint (ANC), i.e., $\alpha_i \geq 0$, and the abundance sum-to-one constraint (ASC), i.e., $\sum_{i=1}^p \alpha_i = 1$. However, in this work we focus on the unconstrained estimation as it is much faster and it has been shown in practice to provide satisfactory results if the model endmembers are properly selected.

3. GPU IMPLEMENTATION

GPUs can be abstracted in terms of a stream model, under which all data sets are represented as streams (i.e., ordered data sets). The architecture of a GPU can be seen as a set of multiprocesors, where 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. Algorithms are constructed by chaining so-called *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 which share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. With the above ideas in mind, our GPU implementation of the hyperspectral unmixing chain comprises two stages: 1) GPU implementation of OSP-GS; and 2) GPU implementation of UCLS. Our GPU implementation of OSP-GS can be summarized by the following steps:

1. Once the hyperspectral image is mapped onto the GPU memory, a structure is created in which the number of blocks equals the number of lines in the hyperspectral image and the number of threads equals the number of samples. A kernel is now used to calculate the brightest pixel in the hyperspectral image. This kernel computes (in parallel) the dot product between each pixel vector and its own transposed version, retaining the pixel that results in the maximum projection value and labeling it as the first endmember $\mathbf{e}_1$.

2. Once the brightest pixel in the hyperspectral image has been identified, the pixel is allocated as the first column in matrix $\mathbf{M}$. The algorithm now calculates the orthogonal vectors through the Gram-Schmidt method, this operation is performed in the CPU. A new kernel, in which the number of blocks equals the number of lines in the hyperspectral image and the number of threads equals the number of samples, is now applied to project the orthogonal vector onto each pixel in the image. The maximum of all projected pixels is calculated using a separate kernel which obtains the next endmember $\mathbf{e}_2$.

3. The algorithm now extends the endmember matrix as $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2]$ and repeats from step 2 until the desired number of endmembers (specified by the input parameter p) has been detected. The output of the algorithm is a set of endmembers $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_p]$.

On the other hand, our implementation of UCLS for GPUs can be summarized by the following steps:

1. The first step is to calculate the operation $(\mathbf{M}^T\mathbf{M})$, where $\mathbf{M} = \{\mathbf{e}_i\}_{i=1}^{p}$ is formed by the p endmembers extracted by the OSP-GS. The inverse of this operation is calculated in the CPU mainly due to two reasons: 1) its computation is relatively fast, and 2) the inverse operation remains the same throughout the whole execution of the code.

2. The result calculated in the previous step is now multiplied by the each pixel $\mathbf{x}$ in the hyperspectral image, thus obtaining a set of abundance vectors $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_p]$, each containing the fractional abundances of the p endmembers in $\mathbf{M}$. This is accomplished in the GPU by means of a specific kernel, which produces p abundance maps.

4. MULTI-CORE IMPLEMENTATION

A multi-core processor is a single CPU with two o more independent actual processors (called *cores*), which are the units that read and execute program instructions. The multiple cores can run multiple instructions at the same time, increasing the overall speed for programs amenable to parallel computing. In our implementation of the considered unmixing chain, we used `OpenMP`[1] which is an applications program interface (API) used to explicity address multi-threaded, shared-memory parallelism. In `OpenMP` the users specify the regions in the code that are suitable for parallel implementation. The user also specifies necessary synchronization operations, such as locks or barriers, to ensure correct execution of the parallel region. At runtime the threads are executed in different processors but sharing the same memory and address space. In the following, we briefly summarize the main techniques used in the multi-core implementation of the considered unmixing chain:

- One of the main techniques adopted in the `OpenMP` implementation of the considered unmixing chain is the use of locking routines. For instance, these routines are used in the OSP-GS algorithm to calculate the brighest pixel in the scene and the maximum projection operation.

- Another strategy adopted in the `OpenMP` implementation of the considered unmixing chain is the use of `parallel for` directives, which indicate the compiler that the structured block of code should be executed in parallel on multiple threads. Each thread will execute the same instruction stream, however not necessarily the same set of instructions. These directives are used with the locking routines and to implement the different steps for the UCLS algorithm in section 3.

[1]http://openmp.org

5. EXPERIMENTAL RESULTS

The hyperspectral image scene used in experiments is the well-known Cuprite scene, collected by the Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) [1] in the summer of 1997 and available online in reflectance units after atmospheric correction[2]. The portion used in experiments corresponds to a 350×350-pixels subset of the sector labeled as f970619t01p02_r02_sc03.a.rfl in the online data, which comprises 188 spectral bands in the range from 400 to 2500 nm and a total size of around 50 MB. Water absorption bands as well as bands with low signal-to-noise ratio (SNR) were removed prior to the analysis. The site is well understood mineralogically, and has several exposed minerals of interest, including *alunite*, *buddingtonite*, *calcite*, *kaolinite*, and *muscovite*. Reference ground signatures of the above minerals, available in the form of a USGS library[3] will be used in this work for evaluation purposes.

The number of endmembers to be detected was set to $p = 19$ after calculating the virtual dimensionality (VD) [5] of the AVIRIS Cuprite image. Table 1 shows the spectral angles (in degrees) between the most similar endmembers extracted by the OSP-GS and the reference USGS spectral signatures. The range of values for the spectral angle is [0°, 90°]. As shown by Table 1, the endmembers extracted by the OSP-GS algorithm are very similar, spectrally, to the USGS reference signatures, despite the potential variations (due to posible interferers still remaining after the atmospheric correction process) between the ground signatures and the airbone data. Since no reference information is available regarding the true abundance fractions of minerals in the AVIRIS Cuprite data, no quantitative experiments were conducted although the obtained mineral maps exhibit similar correlation with regards to previously published maps[4]. These results have been discussed in a previous work [6].

The proposed parallel implementations of the hyperspectral unmixing chain have been tested on a NVidia™ GeForce GTX 580 GPU[5], which features 512 processor cores operating at 1.544 GHz, with single precision floating point performance of 1,354 Gflops, double precision floating point performance of 198 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. The GPU is connected to an multi-core Intel i7 920 CPU at 2.67 GHz with 8 cores, which uses a motherboard Asus P6T7 WS SuperComputer. It is important to emphasize that our GPU and multi-core versions provide exactly the same results as the serial versions of the implemented algorithms, using the `gcc` (gnu compiler default) with optimation flags `-O3` and `-fopenmp` (flag used for the multi-core version) to exploit

[2]http://aviris.jpl.nasa.gov
[3]http://speclab.cr.usgs.gov
[4]http://speclab.cr.usgs.gov/cuprite.html
[5]http://www.nvidia.com/object/product-geforce-gtx-580-us.html

Table 1. Spectral angle values (in degrees) between the target pixels extracted by the OSP-GS algorithm and the reference USGS mineral signatures for the AVIRIS Cuprite scene.

Alunite	Buddingtonite	Calcite	Kaolinite	Muscovite	Average
5.48°	4.08°	5.87°	11.14°	5.68°	6.45°

Table 2. Processing times (in milliseconds) and speedsups achieved for the parallel unmixing chain in two different platforms: multi-core and GPU, tested with the AVIRIS Cuprite scene.

	Initialization	OSP-GS	UCLS	Writing of final results	Total
Serial time	36.158	1650.638	1423.173	10.335	3109.969
Parallel time (multi-core)	35.271	338.177	296.473	13.239	669.921
Parallel time GPU	38.866	12.426	19.491	10.207	81.010
Speedup (multi-core)	–	4.88	4.80	–	4.64
Speedup (GPU)	–	132.84	73.02	–	38.39

data locality and avoid redundant computations. Hence, the only difference between the serial and parallel algorithms is the time they need to complete their calculations. The serial algorithms were executed in one of the available cores, the GPU times were measured in the considered GPU platform, and the multi-core version were executed in all the available cores. For each experiment, ten 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 2 summarizes the obtained results. It should be noted that the cross-track line scan time in AVIRIS, a push-broom instrument [1], is quite fast (8.3 milliseconds to collect 512 full pixel vectors). This introduces the need to process the considered AVIRIS Cuprite scene (350×350 pixels and 188 spectral bands) in less than 1985 milliseconds to fully achieve real-time performance. As shown by Table 2, the two considered parallel implementations of the unmixing chain (GPU and multi-core) are strictly in real-time, including data transfers. This represents an important improvement with respect to previosly published implementations of this unmixing chain, which performs in real-time using only eight cores in the case of the multi-core version. The performance is significantly improved when the GPU is considered.

6. REFERENCES

[1] R. O. Green et al., "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment*, vol. 65, no. 3, pp. 227–248, 1998.

[2] A. Plaza, Q. Du, J. Bioucas-Dias, X. Jia, and F. Kruse, "Foreword to the special issue on spectral unmixing of remotely sensed data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 19, no. 11, pp. 4103–4110, 2011.

[3] A. Plaza, J. Plaza, A. Paz, and S. Sanchez, "Parallel hyperspectral image and signal processing," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 119–126, 2011.

[4] J. C. Harsanyi and C.-I Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 32, no. 4, pp. 779–785.

[5] C.-I Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*, Kluwer Academic: New York, 2003.

[6] 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 & Experience*, vol. 23, no. 13, pp. 1538–1557, 2011.