Proceedings of the International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2009
30 June, 1–3 July 2009.

Optimization of a Hyperspectral Image Processing Chain Using Heterogeneous and GPU-Based Parallel Computing Architectures

Antonio J. Plaza[1], Javier Plaza[1], Sergio Sánchez[1] and Abel Paz[1]

[1] *Department of Technology of Computers and Communications, University of Extremadura, Avda. de la Universidad s/n, E-10071 Cáceres, Spain*

emails: aplaza@unex.es, jplaza@unex.es, dante84@unex.es, apazga@unex.es

Abstract

Hyperspectral imaging is a new technique in remote sensing that generates hundreds of images, at different wavelength channels, for the same area on the surface of the Earth. In recent years, several efforts have been directed towards the incorporation of high-performance computing systems and architectures into remote sensing missions. With the aim of providing an overview of current and new trends in the design of parallel and distributed systems for remote sensing applications, this paper presents two solutions for efficient implementation of a hyperspectral image processing chain based on mixed pixel analysis. The first solution is intended for efficient exploitation of hyperspectral data after being transmitted to Earth, and is tested on a heterogeneous network of workstations at University of Maryland. The second solution is intended for on-board, real-time exploitation of hyperspectral data, and is tested on an NVidia graphics processing unit (GPU). Combined, the two discussed approaches integrate a system with different levels of priority in processing of the hyperspectral data, which can be tuned depending on the specific requirements of the application scenario. The proposed implementations are evaluated using hyperspectral data collected by the Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) operated by NASA/JPL.

Key words: hyperspectral imaging, high performance computing, heterogeneous parallel computing, commodity graphics hardware, mixed pixel analysis.

1 Introduction

Hyperspectral imaging is concerned with the measurement, analysis, and interpretation of spectra acquired from a given scene (or specific object) at a short, medium or long distance by an airborne or satellite sensor [1]. For instance, NASA is continuously gathering hyperspectral data with Earth-observing sensors such as JPL's Airborne

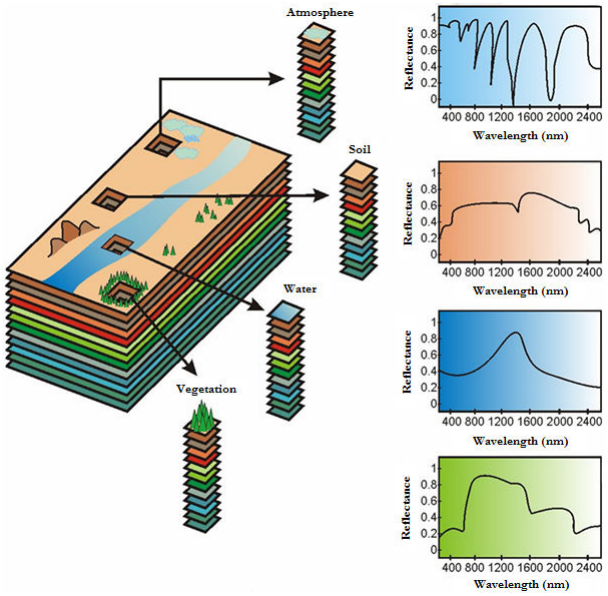

Figure 1: The concept of hyperspectral imaging.

Visible-Infrared Imaging Spectrometer (AVIRIS) [2], able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of the reflected light of an area 2 to 12 kilometers wide and several kilometers long using 224 spectral bands. The resulting hyperspectral data cube is a stack of images (see Fig. 1) in which each pixel (vector) has an associated spectral signature or *fingerprint* that uniquely characterizes the underlying objects, and the resulting data volume typically comprises several GBs per flight.

The wealth of spatial and spectral information provided by hyperspectral instruments has opened ground-breaking perspectives in many application domains, including environmental modeling and assessment, target detection for military and defense/security purposes, urban planning and management studies, risk/hazard prevention and response including wild land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination [3]. Most of the above-cited applications require analysis algorithms able to provide a response in (near) real-time, which is an ambitious goal since the price paid for the rich information available from hyperspectral sensors is the enormous amounts of data that they generate.

The utilization of high performance computing (HPC) systems has become more and more widespread in hyperspectral imaging applications [4]. Although most parallel techniques and systems for image information processing employed by NASA and other institutions during the last decade have chiefly been homogeneous in nature [5], a recent trend in the design of HPC systems for data-intensive problems is to utilize highly heterogeneous computing resources [6]. As shown in previous work [7, 8], networks of heterogeneous computing resources can realize a very high level of aggregate performance in hyperspectral imaging applications. Although remote sensing data pro-

cessing algorithms map nicely to heterogeneous networks of computers, these systems are generally expensive and difficult to adapt to on-board data processing scenarios, in which low-weight and low-power integrated components are essential to reduce mission payload and obtain analysis results in real-time, i.e., at the same time as the data is collected by the sensor. In this regard, an exciting new development in the field of commodity computing is the emergence of commodity graphic processing units (GPUs), which can bridge the gap towards on-board processing of remotely sensed data [9].

The main purpose of this paper is to describe the two components (ground and on-board) of an integrated parallel system for efficient processing of remotely sensed hyperspectral data. These two solutions provide different levels of priority in remote sensing data processing, with the ground segment being mainly oriented to information extraction from data sets already transmitted to Earth, while the on-board system can be applied for real-time data processing in time-critical applications. As a case study, we focus on efficient implementation of a standard hyperspectral image processing chain, available into Kodak's Research Systems ENVI software[1] which is one of the most widely used remote sensing software packages. The remainder of the paper is organized as follows. Section 2 describes the hyperspectral processing chain used as a representative case study in this work. Section 3 develops parallel implementations of the hyperspectral processing chain. Section 4 provides an experimental comparison of the proposed implementations using several architectures, including a heterogeneous network of workstations at University of Maryland and an NVidia GeForce 8800 GTX GPU. Section 5 concludes the paper with some remarks and hints future research lines.

2 Hyperspectral image processing chain

2.1 Problem formulation

Let us assume that a hyperspectral scene with N bands is denoted by $\mathbf{F}$, in which a pixel of the scene is represented by a vector $\mathbf{f}_i = [f_{i1}, f_{i2}, \cdots, f_{in}] \in \Re^N$, where $\Re$ denotes the set of real numbers in which the pixel's spectral response f_{ik} at sensor channels $k = 1, \ldots, N$ is included. Under the linear mixture model assumption, each pixel vector in the original scene can be modeled using the following expression [10]:

$$\mathbf{f}_i = \sum_{e=1}^{E} \mathbf{e}_e \cdot a_{\mathbf{e}_e} + \mathbf{n}, \tag{1}$$

where $\mathbf{e}_e$ denotes the spectral response of a pure spectral signature (*endmember* [11] in hyperspectral imaging terminology), $a_{\mathbf{e}_e}$ is a scalar value designating the fractional abundance of the endmember $\mathbf{e}_e$, E is the total number of endmembers, and $\mathbf{n}$ is a noise vector. The use of spectral endmembers allows one to deal with the problem of mixed pixels, which arise when the spatial resolution of the sensor is not high enough to separate different materials. For instance, it is likely that the pixel labeled as 'vegetation' in Fig. 1 actually comprises a mixture of vegetation and soil. In this case,

[1] ITT Visual Information Solutions, ENVI User's Guide. Online: http://www.ittvis.com.

the measured spectrum can be decomposed into a linear combination of pure spectral endmembers of soil and vegetation, weighted by abundance fractions that indicate the proportion of each endmember in the mixed pixel. The solution of the linear spectral mixture problem described in (1) relies on the correct determination of a set $\{\mathbf{e}_e\}_{e=1}^{E}$ of endmembers and their correspondent abundance fractions $\{a_{\mathbf{e}_e}\}_{e=1}^{E}$ at each pixel $\mathbf{f}_i$.

2.2 Spectral unmixing methodology

A standard approach to decompose mixed pixels in hyperspectral images is linear spectral unmixing, which comprises the following stages. Firstly, a set of spectral endmembers are extracted from the input data set. For this purpose, we have considered a standard algorithm in the literature: the pixel purity index (PPI) algorithm [12]. Then, the fractional abundances of such endmembers in each pixel of the scene is obtained using an inversion process [10]. The inputs to the hyperspectral processing chain considered in this work are a hyperspectral image cube $\mathbf{F}$ with N spectral bands and T pixel vectors; the number of endmembers to be extracted, E, a maximum number of projections, K; a cut-off threshold value, v_c, used to select as final endmembers only those pixels that have been selected as extreme pixels at least v_c times throughout the process; and a threshold angle, v_a, used to discard redundant endmembers. The chain can be summarized by the following steps:

1. *Skewer generation.* Produce a set of K randomly generated unit vectors, denoted by $\{\mathbf{skewer}_j\}_{j=1}^{K}$.

2. *Extreme projections.* For each $\mathbf{skewer}_j$, all sample pixel vectors $\mathbf{f}_i$ in the original data set $\mathbf{F}$ are projected onto $\mathbf{skewer}_j$ via products of $|\mathbf{f}_i \cdot \mathbf{skewer}_j|$ to find sample vectors at its extreme (maximum and minimum) projections, forming an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}(\mathbf{skewer}_j)$.

3. *Calculation of pixel purity scores.* Define an indicator function of a set S, denoted by $I_S(\mathbf{f}_i)$, to denote membership of an element $\mathbf{f}_i$ to that particular set as $I_S(\mathbf{f}_i) = 1$ if $\mathbf{f}_i \in S$. Using the indicator function above, calculate the number of times that given pixel has been selected as extreme using the following equation:

$$N_{times}(\mathbf{f}_i) = \sum_{j=1}^{K} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i) \tag{2}$$

4. *Endmember selection.* Find the pixels with value of $N_{times}(\mathbf{f}_i)$ above v_c and form a unique set of E endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the spectral angle distance (SAD) for all possible endmember pairs and discarding those which result in an angle value below v_a. SAD is invariant to multiplicative scalings that may arise due to differences in illumination and sensor observation angle [10]. The SAD between endmember $\mathbf{e}_i$ and endmember $\mathbf{e}_j$ is defined as follows:

$$\mathrm{SAD}(\mathbf{e}_i, \mathbf{e}_j) = cos^{-1} \frac{\mathbf{e}_i \cdot \mathbf{e}_j}{\|\mathbf{e}_i\| \cdot \|\mathbf{e}_j\|} \tag{3}$$

5. *Spectral unmixing.* For each sample pixel vector $\mathbf{f}_i$ in $\mathbf{F}$, a set of abundance fractions specified by $\{a_{\mathbf{e}_1}, a_{\mathbf{e}_2}, \cdots, a_{\mathbf{e}_E}\}$ is obtained using the set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$, so that each $\mathbf{f}_i$ can be expressed as a linear combination of endmembers as $\mathbf{f}_i = \mathbf{e}_1 \cdot a_{\mathbf{e}_1} + \mathbf{e}_2 \cdot a_{\mathbf{e}_2} + \cdots + \mathbf{e}_E \cdot a_{\mathbf{e}_E}$, thus solving the mixture problem.

3 Parallel implementations

3.1 Heterogeneous parallel implementation

Let us assume that a heterogeneous network can be modeled as a complete graph where each node models a computing resource p_i weighted by its relative cycle-time w_i [6]. Each edge in the graph models a communication link weighted by its relative capacity. In the following, we assume for simplicity that all communication links have the same capacity, thus relating our study to a partially heterogeneous network. With the above assumptions in mind, processor p_i should accomplish a share of $\alpha_i \times W$ of the total workload, denoted by W, to be performed by a certain algorithm, with $\alpha_i \geq 0$ for $1 \leq i \leq P$ and $\sum_{i=1}^{P} \alpha_i = 1$. With the above definitions in mind, the heterogeneous parallel implementation can be summarized as follows:

1. Obtain necessary information about the heterogeneous system, including the number of available processors P, each processor's identification number $\{p_i\}_{i=1}^{P}$, and processor cycle-times $\{w_i\}_{i=1}^{P}$.

2. Let V denote the total volume of data in the original hyperspectral image $\mathbf{F}$. Processor ps_i will be assigned a certain share $\alpha_i \times V$ of the input volume, where $\alpha_i \geq 0$ for $1 \leq i \leq P$ and $\sum_{i=1}^{P} \alpha_i = 1$. In order to obtain the value of α_i for processor p_i, calculate $\alpha_i = \frac{(1/w_i)}{\sum_{j=1}^{P}(1/w_j)}$.

3. Once the set $\{\alpha_i\}_{i=1}^{P}$ has been obtained, a further objective is to produce P spatial-domain partitions of the input hyperspectral data set. To do so, we first obtain a partitioning of the hyperspectral data set so that the number of entire pixel vectors allocated to each processor p_i is proportional to its associated value of α_i. Then, we refine the initial partitioning taking into account the local memory associated to each processor [8].

4. *Skewer generation.* Generate K random unit vectors $\{\mathbf{skewer}_j\}_{j=1}^{K}$ in parallel, and broadcast the entire set of skewers to all the workers.

5. *Extreme projections.* For each $\mathbf{skewer}_j$, project all the sample pixel vectors at each local partition p onto $\mathbf{skewer}_j$ to find sample vectors at its extreme projections, and form an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}^{(p)}(\mathbf{skewer}_j)$. Now calculate the number of times each pixel vector $\mathbf{f}_i^{(p)}$ in the local partition is selected as extreme using the following expression:

$$N_{times}^{(p)}(\mathbf{f}_i^{(l)}) = \sum_{j=1}^{K} I_{S_{extrema}^{(p)}(\mathbf{skewer}_j)}(\mathbf{f}_i^{(l)}) \quad (4)$$

6. *Candidate selection.* Each worker now sends the number of times each pixel vector in the local partition has been selected as extreme to the master, which forms a final matrix of pixel purity indices N_{times} by combining all the individual matrices $N_{times}^{(p)}$ provided by the workers.

7. *Endmember selection.* The master selects those pixels with $N_{times}(\mathbf{f}_i) > v_c$ and forms a unique set $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the SAD for all possible pixel vector pairs and discarding those pixels which result in angle values below v_a.

8. *Spectral unmixing.* For each sample pixel vector $\mathbf{f}_i$ in $\mathbf{F}$, obtain (in embarrasingly parallel fashion) the set of abundance fractions specified by $\{a_{\mathbf{e}_1}, a_{\mathbf{e}_2}, \cdots, a_{\mathbf{e}_E}\}$, so that each $\mathbf{f}_i$ can now be expressed as a linear combination of the set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$, weighted by their corresponding fractional abundances.

3.2 GPU-based parallel implementation

The first issue that needs to be addressed is how to map a hyperspectral image onto the memory of the GPU. Since the size of hyperspectral images usually exceeds the capacity of such memory, we split them into multiple spatial-domain partitions, so that each partition incorporates all the spectral information on a localized spatial region and is composed of spatially adjacent pixel vectors. In order to accommodate the partitions onto the GPU memory, each partition is further subdivided into 4-band tiles (called spatial-domain tiles), which are arranged in different areas of a 2-D texture. Such partitioning allows us to map four consecutive spectral bands onto the RGBA color channels of a texture (memory) element. Apart from the tiles, we also allocate additional memory to hold other information, such as the skewers and intermediate dot products, norms, and SAD-based distances.

Figure 2 shows a flowchart describing our GPU-based implementation. The *data partitioning* stage performs the spatial-domain decomposition of the original hyperspectral image $\mathbf{F}$. In the *stream uploading* stage, the spatial-domain partitions are uploaded as a set of tiles onto the GPU memory. The *skewer generation* stage provides K skewers, using NVidia's parallel implementation of the Mersenne twister pseudo-random number generator on the GPU [13]. The remaining stages comprise the following kernels:

- *Extreme projections.* The tiles are input streams to this stage, which obtains all the dot products necessary to compute the required projections. Since streams are actually tiles, the implementation of this stage is based on a multi-pass kernel that implements an element-wise multiply and add operation, thus producing four partial inner products stored in the RGBA channels of a texture element.

- *Candidate selection.* This kernel uses as inputs the projection values generated in the previous stage, and produces a stream for each pixel $\mathbf{f}_i$, containing the

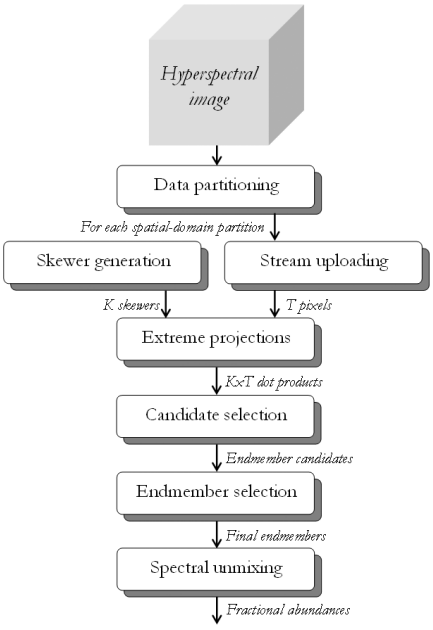

Figure 2: Flowchart of the proposed stream-based GPU implementation.

relative coordinates of the pixels with maximum and minimum distance after the projection onto each skewer. A complementary kernel is then used to identify those pixels which have been selected at least v_c times during the process.

- *Endmember selection.* For each endmember candidate, this kernel computes the cumulative SAD with all the other candidates. It is based on a single-pass kernel that computes the SAD between two pixel vectors using the dot products and norms produced by the previous stage. A complementary kernel is then used to discard those candidates with cumulative SAD scores below v_a, thus producing a final set of spectrally unique endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.

- *Spectral unmixing.* Finally, this kernel uses as inputs the final endmembers selected in the previous stage and produces the endmember fractional abundances for each pixel $\mathbf{f}_i$, thus solving the mixture problem. In order to achieve this, the kernel multiplies each $\mathbf{f}_i$ by $(\mathbf{M}^{\mathrm{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathrm{T}}$, where $\mathbf{M} = \{\mathbf{e}_e\}_{e=1}^{E}$ and the superscript "T" denotes a matrix transpose operation.

4 Experimental results

4.1 Parallel computing systems

Two parallel computing systems have been used in experiments. The first one is a heterogeneous network consisting of 16 different workstations which has been used to evaluate the proposed heterogeneous implementation. Table 1 shows the properties of

Table 1: Specifications of heterogeneous computing nodes.

Processor number	Architecture overview	Cycle-time (seconds/Mflop)	Memory (MB)	Cache (KB)
p_1	Intel Pentium 4	0.0058	2048	1024
p_2, p_5, p_8	Intel Xeon	0.0102	1024	512
p_3	AMD Athlon	0.0026	7748	512
p_4, p_6, p_7, p_9	Intel Xeon	0.0072	1024	1024
p_{10}	UltraSparc-5	0.0451	512	2048
$p_{11} - p_{16}$	AMD Athlon	0.0131	2048	1024

the 16 heterogeneous workstations, which are interconnected using the same homogeneous communication network with capacity $c = 26.64$ milliseconds. Although this is a simple architecture, it is also a quite typical and realistic one as well.

The GPU-based experiments were performed on a 2006-model HP xw8400 workstation based on dual Quad-Core Intel Xeon processor E5345 running at 2.33 GHz with 1.333 MHz bus speed and 3 GB RAM. The computer was equipped with an NVidia GeForce 8800 GTX with 16 multiprocessors, each composed of 8 SIMD processors operating at 1350 Mhz. Each multiprocessor has 8192 registers, a 16 KB parallel data cache of fast shared memory, and access to 768 MB of global memory. The processing chain was implemented using NVidia's Compute Unified Device Architecture (CUDA).

4.2 Hyperspectral data

A well-known hyperspectral data set collected over the Cuprite mining district in Nevada was used in experiments to evaluate the algorithms in the context of a real mineral mapping application. The data set[2] consists of 1939×677 pixels and 224 bands in the wavelength range 0.4–2.5 μm (574 MB in size). The Cuprite site has been extensively mapped by the U.S. Geological Survey (USGS), and there is extensive ground-truth information available, including a library of mineral signatures collected on the field[3]. Fig. 3(a) shows the spectral band at 587 nm wavelength of the AVIRIS scene. The spectra of USGS ground minerals are also displayed in Figs. 3(b-c).

4.3 Performance evaluation

Table 2 shows the SAD-based spectral similarity scores obtained after comparing the ten USGS library spectra with the corresponding endmembers extracted by the original chain in Kodak's Research Systems ENVI software, version 4.5, and the two parallel implementations of the processing chain (heterogeneous and GPU). It is important to emphasize that smaller SAD values indicate higher spectral similarity. As shown by Table 2, the spectral similarity scores with regards to the reference USGS signatures were very satisfactory. In all cases, we empirically set parameter v_c (threshold value) to the mean of N_{times} scores obtained after $K = 10^4$ iterations, while we set $v_a = 0.01$ (threshold angle to remove redundant endmembers) according to previous work [11].

[2]Available online from http://aviris.jpl.nasa.gov/html/aviris.freedata.html
[3]Available online from http://speclab.cr.usgs.gov/spectral-lib.html

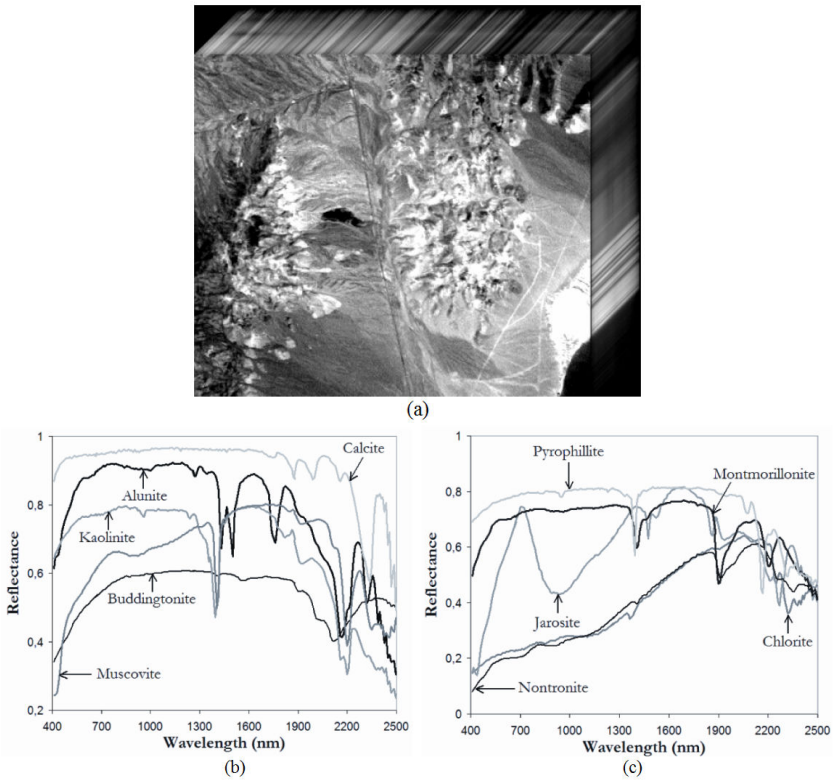

Figure 3: (a) A portion of the AVIRIS scene over Cuprite mining district. (b-c) Ground-truth mineral spectra provided by USGS.

Table 3 shows the performance gain of the heterogeneous implementation with regards to the sequential version of the processing chain as the number of processors was increased. Here, we assumed that processor p_3 (the fastest) was always the master and varied the number of slaves. The construction of speedup plots in heterogeneous environments is not straightforward, mainly because the workers do not have the same relative speed, and therefore the order in which they are added to plot the speedup curve needs to be further analyzed. We have tested three different ordering strategies:

1. *Strategy #1.* First, we used an ordering strategy in which new processors were added according to their numbering in Table 1, i.e., the first case study tested (labeled as "2 CPUs" in Table 3) consisted of using p_3 as the master and p_0 as the slave; the second case (labeled as "3 CPUs") consisted of using p_3 as the master and $\{p_0, p_1\}$ as slaves, and so on, until a final case (labeled as "15 CPUs") was tested, based on using p_3 as the master and all remaining 15 processors as slaves.

2. *Strategy #2.* Second, we used an ordering strategy based on the relative speed of processors in Table 1, i.e., the first case study tested (labeled as "2 CPUs" in Table 3) consisted of using processor p_3 as the master and processor p_{10} (i.e., the one with lowest relative speed) as the slave, and so on.

Table 2: SAD spectral similarity scores between the endmembers extracted by different implementations of the hyperspectral processing chain and USGS reference signatures.

USGS mineral	ENVI	Heterogeneous	GPU-based
Alunite	0.084	0.084	0.084
Buddingtonite	0.071	0.075	0.071
Calcite	0.099	0.099	0.091
Chlorite	0.065	0.065	0.065
Jarosite	0.091	0.091	0.093
Kaolinite	0.136	0.136	0.145
Montmorillonite	0.106	0.112	0.106
Muscovite	0.092	0.092	0.092
Nontronite	0.099	0.102	0.102
Pyrophillite	0.094	0.094	0.097

Table 3: Speedups achieved by the proposed parallel heterogeneous implementation on the heterogeneous network.

# CPUs	Strategy #1	Strategy #2	Strategy #3
2	1.93	1.90	1.87
3	2.91	2.92	2.88
4	3.88	3.89	3.67
5	4.83	4.89	4.72
6	5.84	5.81	5.74
7	6.75	6.83	6.55
8	7.63	7.76	7.61
9	8.81	8.74	7.65
10	9.57	9.68	9.53
11	10.62	10.65	10.44
12	11.43	11.55	11.41
13	12.25	12.42	12.36
14	13.16	13.32	13.29
15	14.22	14.25	14.22
16	15.19	15.22	15.16

3. *Strategy #3.* Finally, we also used a random ordering strategy, i.e., the first case study tested (labeled as "2 CPUs" in Table 3) consisted of using processor p_3 as the master and a different processor, selected randomly among the remaining processors, as the slave, until all remaining processors were exhausted.

As shown by Table 3, the three ordering strategies tested provided almost linear performance increase (regardless of the relative speed of the nodes). Although the proposed implementation scales well in heterogeneous environments, there are several restrictions in order to incorporate this type of platform for on-board processing in remote sensing missions. To address this issue, we compare the performance of a fully optimized CPU implementation and the GPU implementation by measuring the execution time as a function of the image size. Table 4 shows the execution times measured for different image sizes by the CPU and GPU-based implementations, respectively, where the largest image size in the table (574 MB) corresponds to the full hyperspectral scene (1939×677 pixels and 224 bands) whereas the others correspond to cropped portions of the same image. The C function *clock()* was used for timing the CPU implementation and the CUDA timer was used for the GPU implementation. The time

Table 4: Processing time (seconds) for the CPU and GPU-based implementations.

Size (MB)	Processing time (CPU)	Processing time (GPU)
68	81.53	2.88
136	162.75	5.93
205	244.22	8.25
273	325.21	10.90
410	489.69	16.24
574	685.45	22.62

measurement was started right after the hyperspectral image file was read to the CPU memory and stopped right after the results of the processing chain were obtained and stored in the CPU memory. From Table 4, it can be seen that the full AVIRIS data cube was processed in 22.62 seconds. This response is not strictly in real-time since the cross-track line scan time in AVIRIS, a push-broom instrument [2], is quite fast (8.3 msec). This introduces the need to process the full image cube (1939 lines) in about 16 seconds to achieve real-time performance. Although the proposed implementation can still be optimized, Table 4 indicates that the complexity of the implementation scales linearly with the problem size, i.e. doubling the image size doubles the execution time. The speedups achieved by the GPU implementation over the CPU one remained close to 30 for all considered image sizes, which doubles the best speedup result reported for the heterogeneous network in Table 3 at much lower cost (1 GPU vs 16 CPUs) and, most importantly, with less restrictions in terms of power consumption and size, which are very important when defining mission payload in remote sensing missions.

5 Conclusions and future research lines

Remote sensing missions require efficient processing systems able to cope with the extremely high dimensionality of the collected data without compromising mission payload. In this paper, two innovative parallel implementations of a remotely sensed hyperspectral image processing chain for mixed pixel characterization have been evaluated from the viewpoint of both algorithm accuracy and parallel performance. The considered solutions include a heterogeneity-aware parallel implementation, developed for effective information extraction from data sets already transmitted to Earth, and a GPU-based implementation for on-board processing. Although both solutions are complementary, the GPU-based one is more appealing for data analysis in time-critical missions. Future work will comprise optimizing the GPU-based implementation and pursuing implementations in other systems, such as NASA's Discover supercomputer.

Acknowledgements

This research has been developed in the framework of the network "High Performance Computing on Heterogeneous Parallel Architectures" (CAPAP-H), supported by the Spanish Ministry of Science and Innovation (TIN2007-29664-E). Funding from HYPERCOMP/EODIX project (AYA2008-05965-C04-02) is also gratefully acknowledged.

References

[1] A. F. H. Goetz, G. Vane, J. E. Solomon and B. N. Rock, *Imaging spectrometry for Earth remote sensing*, Science (1985) 1147–1153.

[2] R.O. Green et al., *Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)*, Remote Sensing of Environment (1998) 227–248.

[3] D. A. Landgrebe, *Signal theory methods in multispectral remote sensing*, Wiley, Hoboken, NJ, 2003.

[4] A. Plaza and C.-I Chang, *High performance computing in remote sensing*, CRC Press, Boca Raton, FL, 2007.

[5] A. Plaza and C.-I Chang, *Clusters versus FPGA for parallel processing of hyperspectral imagery*, International Journal of High Performance Computing Applications (2008) 366–385.

[6] A. Lastovetsky, *Parallel computing on heterogeneous networks*, Wiley, Hoboken, NJ, 2003.

[7] A. Plaza, J. Plaza and D. Valencia, *Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high–dimensional image data*, Journal of Supercomputing (2007) 81–107.

[8] A. Plaza, *Parallel techniques for information extraction from hyperspectral imagery using heterogeneous networks of workstations*, Journal of Parallel and Distributed Computing (2008) 93–111.

[9] J. Setoain, M. Prieto, C. Tenllado and F. Tirado, *GPU for parallel on-board hyperspectral image processing*, International Journal of High Performance Computing Applications (2008) 424–437.

[10] C.-I Chang, *Hyperspectral imaging: techniques for spectral detection and classification*, Kluwer, New York, 2003.

[11] A. Plaza, P. Martinez, R. Perez and J. Plaza, *A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data*, IEEE Transactions on Geoscience and Remote Sensing (2004) 650–663.

[12] J. W. Boardman, *Automating spectral unmixing of AVIRIS data using convex geometry concepts*, Proceedings of the Fourth Annual NASA/JPL Airborne Earth Science Workshop (1993) 11–14.

[13] M. Matsumoto and T. Nishimura, *Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator*, ACM Transactions on Modeling and Computer Simulation (1998) 3–30.