Recent Developments and Future Directions in Parallel Processing of Remotely Sensed Hyperspectral Images

Antonio J. Plaza
Department of Technology of Computers and Communications
Escuela Politécnica de Cáceres, University of Extremadura
Avda. de la Universidad s/n, E-10071 Cáceres, Spain
E-mail: aplaza@unex.es – URL: http://www.umbc.edu/rssipl/people/aplaza

Abstract

Remotely sensed hyperspectral imaging is a technique that generates hundreds of spectral bands at different wavelength channels for the same area on the surface of the Earth. Computationally effective processing of these image cubes can be greatly beneficial in many application domains, including environmental modeling, risk/hazard prevention and response, or defense/security. With the aim of providing an overview of recent developments and new trends in the design of parallel and distributed systems for hyperspectral image analysis, this paper discusses and inter-compares four different strategies for efficiently implementing a standard hyperspectral image processing chain: 1) commodity Beowulf-type clusters, 2) heterogeneous networks of workstations, 3) field programmable gate arrays (FPGAs), and 4) graphics processing units (GPUs). Combined, these parts deliver a snapshot of the state-of-the-art in those areas, and a thoughtful perspective on the potential and emerging challenges of adapting high performance computing systems to remote sensing problems.

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 [3]. For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) [4], 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 [2] is a stack of images (see Fig. 1) in which each pixel (vector) is represented by a spectral signature or 'fingerprint' that uniquely characterizes the underlying objects, and the resulting multidimensional data volume typically comprises several GBs per flight.

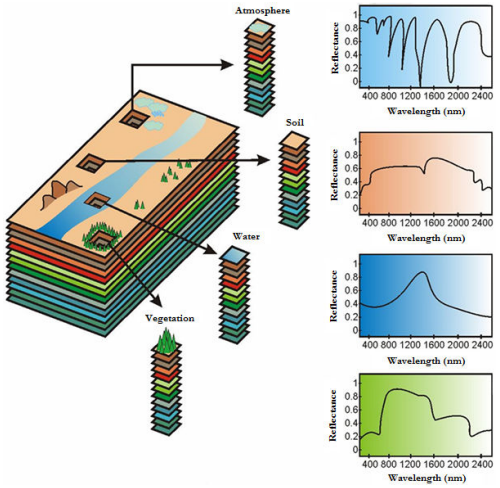

Figure 1. Concept of hyperspectral imaging.

The utilization of high performance computing (HPC) systems in hyperspectral imaging applications has become more widespread in recent years [6]. Although most parallel systems used for hyperspectral image processing have been chiefly homogeneous in nature (i.e., they are made up of identical processing units [10]), a recent trend in the design of HPC-based parallel algorithms for solving hyperspectral imaging problems has been to utilize highly heterogeneous computing resources [9, 13].

On the other hand, although these algorithms map nicely to clusters and networks of workstations, these systems are generally expensive and difficult to adapt to on-board data processing requirements introduced by several applications, such as wild land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination. In those cases, low-weight and low-power integrated components are essential to reduce mission payload and obtain analysis results quickly enough for practical use. In this regard, the emergence of specialized hardware devices such as field programmable gate arrays (FPGAs) [7] and graphic processing units (GPUs) [12] has helped in bridging the gap towards real-time analysis of remotely sensed hyperspectral data.

Authorized licensed use limited to: Universidad de Extremadura. Downloaded on November 16, 2009 at 08:18 from IEEE Xplore. Restrictions apply.

In this paper, we present our experience in efficient implementation of a standard hyperspectral image processing chain on four different types of parallel platforms: 1) commodity Beowulf-type clusters, 2) heterogeneous networks of workstations, 3) field programmable gate arrays (FPGAs), and 4) graphics processing units (GPUs). The description and comparison of several strategies for implementation of the same data processing approach is expected to provide a good perspective on recent developments and new perspectives in this active research area. The remainder of the paper is organized as follows. Section 2 describes the hyperspectral processing chain used in our experiments. Section 3 develops several high-performance implementations of the same chain in different parallel platforms. Section 4 describes the hyperspectral data sets and parallel platforms used, followed by a discussion based on several performance metrics. Section 5 concludes with some remarks.

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 wavelengths $k = 1, \ldots, N$ is included. Under the linear mixture model assumption, each pixel vector can be modeled using the following expression [2]:

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

where $\mathbf{e}_e$ denotes the spectral response of a pure spectral signature (*endmember* [8] 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. 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, 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 [1]. Then, the fractional abundances of such endmembers in each pixel of the scene are obtained using an inversion process [2]. This hyperspectral processing chain, available in commercial software packages such as Research Systems ENVI, receives as inputs 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 dot 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 function above, calculate the number of times that given pixel has been selected as extreme using the following equation:

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

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 [2].

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 [2] using the set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ by minimizing the noise term $\mathbf{n}$ in the following expression: $\mathbf{f}_i = \mathbf{e}_1 \cdot a_{\mathbf{e}_1} + \mathbf{e}_2 \cdot a_{\mathbf{e}_2} + \cdots + \mathbf{e}_E \cdot a_{\mathbf{e}_E} + \mathbf{n}$, thus solving the mixture problem.

3 Parallel implementations

3.1 Cluster-based implementation

To reduce code redundancy and enhance reusability, our goal was to reuse much of the code for the sequential algorithm in the different parallel implementations. For that purpose, we adopted a spatial-domain decomposition approach [11] that subdivides the image cube into multiple

Authorized licensed use limited to: Universidad de Extremadura. Downloaded on November 16, 2009 at 08:18 from IEEE Xplore. Restrictions apply.

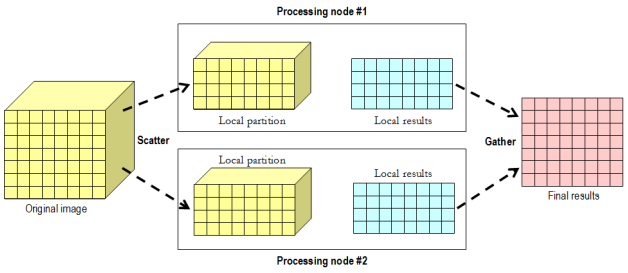

Figure 2. Spatial-domain decomposition.

blocks made up of entire pixel vectors, and assigns one or more blocks to each processing element (see Fig. 2).

It should be noted that the hyperspectral processing chain presented in Section 2 is mainly based on calculations in which pixels are always treated as entire spectral signatures. Therefore, a spectral-domain partitioning scheme (which subdivides the whole multi-band data into blocks made up of contiguous spectral bands or sub-volumes, and assigns one or more sub-volumes to each processing element) is not appropriate in our application [10]. This is because the latter approach breaks the spectral identity of the data because each pixel vector is split amongst several processing element.

A further reason that justifies the above decision is that, in spectral-domain partitioning, the calculations made for each hyperspectral pixel need to originate from several processing elements, and thus require intensive inter-processor communication. Therefore, in our proposed implementation, a master-slave spatial domain-based decomposition paradigm is adopted. As shown in Fig. 2, the master processor sends partial data to the workers and coordinates their actions. Then, the master gathers the partial results provided by the workers and produces a final result.

3.2 Heterogeneous implementation

In order to balance the load of the processors in a heterogeneous parallel environment, each processor should execute an amount of work that is proportional to its speed. Therefore, two major goals of data partitioning in heterogeneous networks should be: to obtain an appropriate set of workload fractions that best fit the heterogeneous environment, and to translate the chosen set of values into a suitable decomposition of the input data, taking into account the properties of the heterogeneous system. In order to accomplish the above goals, we have developed a workload estimation algorithm for heterogeneous networks that assumes that the workload of each processor must be directly proportional to its local memory and inversely proportional to its speed.

The parallel heterogeneous algorithm has been implemented using two strategies. The first one is based on using the C++ programming language with calls to standard MPI functions (also used in our cluster-based implementation). The second strategy was based on using HeteroMPI [13],

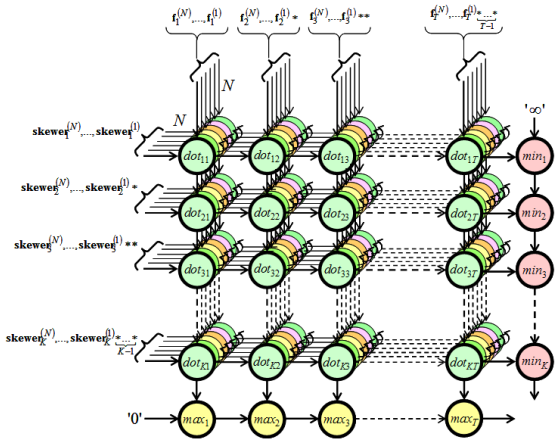

Figure 3. Systolic array design for the FPGA implementation.

a heterogeneous version of MPI which automatically optimizes the workload assigned to each heterogeneous processor. Experimentally, we tested that both implementations resulted in very similar results.

3.3 FPGA implementation

Our hardware-based parallel strategy for implementation of the hyperspectral data processing chain is aimed at enhancing replicability and reusability of slices in FPGA devices through the utilization of systolic array design [7]. Fig. 3 describes the systolic array design adopted in previous work. Here, local results remain static at each processing element, while pixel vectors are input to the systolic array from top to bottom and skewer vectors are fed to the systolic array from left to right. In Fig. 3, asterisks represent delays. The processing nodes labeled as *dot* in Fig. 3 perform the individual products for the skewer projections. On the other hand, the nodes labeled as *max* and *min* respectively compute the maxima and minima projections after the dot product calculations have been completed. In fact, the *max* and *min* nodes can be respectively seen as part of a 1-D systolic array which avoids broadcasting the pixel while simplifying the collection of the results. Based on the system design described above, which also allows implementation of the spectral unmixing stage, we have ported the implementation of the full hyperspectral image processing chain to a Xilinx FPGA using Handel-C, a prototyping language that uses a pseudo-C programming style.

3.4 GPU implementation

In order to accommodate the spatial-domain partitions onto the memory of an NVidia GPU, 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

Authorized licensed use limited to: Universidad de Extremadura. Downloaded on November 16, 2009 at 08:18 from IEEE Xplore. Restrictions apply.

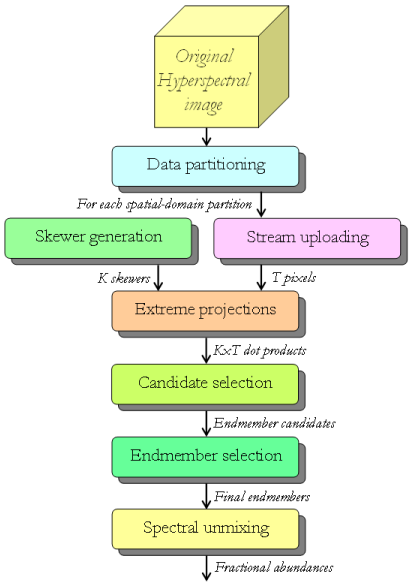

Figure 4. GPU implementation.

(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 4 shows a flowchart describing our GPU-based implementation, which has been developed using the compute unified device architecture (CUDA). The *data partitioning* stage performs the spatial-domain decomposition of the original hyperspectral image. 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 the skewers, using NVidia's parallel implementation of the Mersenne twister pseudo-random number generator on the GPU [5]. 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. 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 vector, containing the 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 repeatedly 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 a threshold angle.

- *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, thus solving the mixture problem.

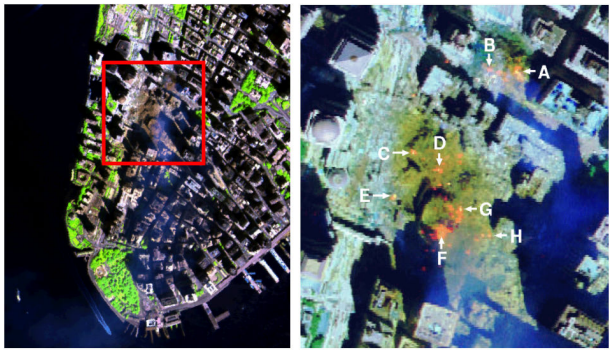

Figure 5. AVIRIS hyperspectral image (left). Location of fires in World Trade Center (right).

4 Experimental results

4.1 Hyperspectral data set

The image scene used for experiments in this work was collected by the AVIRIS instrument, flown over the World Trade Center (WTC) area in New York City on September 16, 2001, just five days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex. The data set consists of 614×512 pixels, 224 spectral bands and a total size of 140 MB. The spatial resolution is 1.7 meters per pixel. Fig. 5(left) shows a false color composite of the data set selected for experiments. Fig. 5(right) shows a thermal map centered at the region where the buildings collapsed. The map shows the target locations of the thermal hot spots, used in this work as ground-truth to validate the proposed parallel algorithms.

4.2 Parallel computing architectures

Four different parallel computing platforms have been used in experiments:

- A Beowulf cluster called Thunderhead at NASA's Goddard Space Flight Center in Maryland, a 512-processor homogeneous Beowulf cluster composed of 256 dual 2.4 GHz Intel Xeon nodes, each with 1 GB of memory and 80 GB of main memory, interconnected with 2 Ghz optical fibre Myrinet.

Authorized licensed use limited to: Universidad de Extremadura. Downloaded on November 16, 2009 at 08:18 from IEEE Xplore. Restrictions apply.

- A fully heterogeneous network at University of Maryland which consists of 16 different workstations and four different communication segments.
- A Xilinx Virtex-II XC2V6000-6 FPGA of Celoxica's ADMXRC2 board, with $33,792$ slices, 144 Select RAM Blocks and 144 multipliers of 18-bit $\times$ 18-bit.
- An NVidia GeForce 8800 GTX GPU 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. Both the GPU and the FPGA were connected to 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.

4.3 Sub-pixel accuracy

Table 1 evaluates the accuracy of the four parallel algorithms in estimating the sub-pixel abundance of fires in Fig. 5(right), taking advantage information about the area covered by each thermal hot spot available from U.S. Geological Survey. Since each pixel in the AVIRIS data has a size of 1.7 square meters, thermal hot spots are sub-pixel in nature and hence require a spectral unmixing step as the last one provided in our hyperspectral processing chain. Experiments in Table 1 demonstrate that the parallel algorithms can provide accurate estimations of the area covered by thermal hot spots, which can lead to good characterization results. In particular, the estimations for the thermal hot spots with higher temperature (labeled as "A" and "G" in the table) were almost perfect, and identical for the four considered implementations.

4.4 Parallel performance

Table 2 shows the total time spent by the parallel implementation in communications and computations in the Thunderhead Beowulf cluster (4 to 256 CPUs) and on the heterogeneous network (16 CPUs). Two types of computation times were analyzed, namely, sequential (those performed by the root node with no other parallel tasks active in the system), and parallel (the rest of computations, i.e. those performed by the root node and/or the workers in parallel). The latter includes the times in which the workers remain idle. In addition, Table 2 also displays the communication times, the total execution times, and the speedups. The total execution time measured for the sequential version on a Thunderhead processor was 1163.05 seconds, while the same code executed on the slowest processor of the heterogeneous network was 1098.63 seconds.

It can be seen from Table 2 that the times for sequential computations were always very low when compared to the time for parallel computations, which anticipates high parallel efficiency on both the cluster and the heterogeneous network, even for a high number of processors. On the other hand, it can also be seen that the impact of communications is not particularly significant.

Table 3 shows the execution times measured for different image sizes by the GPU-based implementation compared to execution in the dual-core CPU of the system in which the GPU is integrated. The largest image size in the table (140 MB) corresponds to the scene shown in Fig. 5, 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 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 3, it can be seen that the GPU can process a standard AVIRIS scene in 18.93 seconds. This response is not strictly in real-time since the cross-track scan line in AVIRIS, a pushbroom instrument [4], is quite fast (8.3 msec). This introduces the need to process a full image cube (614512 pixels with 224 bands) in no more than 5 seconds to achieve real-time performance. Although the proposed implementation can be optimized, Table 3 indicates that the complexity of the implementation scales linearly with 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 26 for all considered image sizes, which doubles the speedup reported for the heterogeneous network in Table 2, at much lower cost (1 GPU vs 16 CPUs) and with less restrictions in terms of power consumption, size and weight, which are important when defining mission payload in remote sensing missions.

Finally, Table 4 shows a summary of resource utilization by the proposed FPGA implementation, using different numbers of processors. Here, the maximum speedup measured was about 24, with a total processing time of about 20 seconds. Although our implementation is not fully optimized and better results can be achieved in terms of speedup and execution times by increasing resource utilization on the FPGA, it is quite important to leave room in the FPGA for additional algorithms so that dynamic algorithm selection can be performed on the fly (as shown by Table 4, when 400 processors are used, 36% of the total available resources in the FPGA are already consumed).

5 Conclusions

In this paper, we have explored different strategies to increase the performance of a standard hyperspectral image processing chain on different parallel platforms. Techniques discussed include a commodity cluster-based implementation, a heterogeneity-aware parallel implementation developed for distributed networks of workstations, an FPGA-based implementation, and a GPU-based implementation. Our study reveals that computational power offered by heterogeneous platforms is likely to introduce substantial changes in the (mostly homogeneous) parallel systems currently used for exploiting large volumes of data already

Authorized licensed use limited to: Universidad de Extremadura. Downloaded on November 16, 2009 at 08:18 from IEEE Xplore. Restrictions apply.

Table 1. Comparison of area estimation (in square meters) for each thermal hot spot by different parallel implementations of the hyperspectral processing chain (USGS reference values included).

Thermal hot spot	Latitude (North)	Longitude (West)	Temperature (Kelvin)	Area (USGS)	Area (Cluster)	Area (Heterogeneous)	Area (FPGA)	Area (GPU)
'A'	$40^o42'47.18"$	$74^o00'41.43"$	1000	0.56	0.53	0.53	0.53	0.53
'B'	$40^o42'47.14"$	$74^o00'43.53"$	830	0.08	0.06	0.10	0.06	0.06
'C'	$40^o42'42.89"$	$74^o00'48.88"$	900	0.80	0.78	0.78	0.78	0.78
'D'	$40^o42'41.99"$	$74^o00'46.94"$	790	0.80	0.81	0.81	0.83	0.83
'E'	$40^o42'40.58"$	$74^o00'50.15"$	710	0.40	0.55	0.59	0.57	0.57
'F'	$40^o42'38.74"$	$74^o00'46.70"$	700	0.40	0.36	0.31	0.36	0.38
'G'	$40^o42'39.94"$	$74^o00'45.37"$	1020	0.04	0.05	0.05	0.05	0.06
'H'	$40^o42'38.60"$	$74^o00'43.51"$	820	0.08	0.12	0.09	0.07	0.11

Table 2. Performance on Thunderhead (4 to 256 CPUs) and the heterogeneous network (16 CPUs).

	Thunderhead Beowulf cluster								Heterogeneous network
# CPUs	4	16	36	64	100	144	196	256	16
Sequential computations	1.63	1.26	1.12	1.19	1.06	0.84	0.91	0.58	1.69
Parallel computations	292.09	73.24	30.46	15.44	8.76	5.08	3.18	1.91	79.56
Communications	2.20	2.41	2.39	2.21	2.46	2.65	2.32	2.49	3.56
Total time	295.92	76.91	33.97	18.84	12.38	8.57	6.41	4.98	83.05
Speedup	3.93	15.12	34.23	61.73	93.89	135.67	181.34	233.45	13.23

Table 3. Processing time (seconds) for the dual-core CPU and GPU implementations.

Size (MB)	Processing time (CPU)	Processing time (GPU)
18	66.58	2.55
36	126.13	4.87
70	249.35	9.57
140	493.78	18.93

Table 4. Summary of resource utilization for the FPGA-based implementation.

Number of processors	Total gates	Total slices	% of total	Frequency (MHz)	Time (seconds)
100	97,443	1,185	3%	29,257	69.91
200	212,412	3,587	10%	21,782	35.36
400	526,944	12,418	36%	18,032	20.48

available in repositories. To address the time-critical constraints introduced by many remote sensing applications, we have also developed FPGA and GPU-based implementations of the hyperspectral processing chain for on-board analysis. A major goal in future missions will be to overcome the bottleneck introduced by the bandwidth of the downlink connection from the observatory platform. In this regard, both the reconfigurability of FPGA systems and the low cost and portability of GPU systems open innovative perspectives. Radiation-tolerance and power consumption issues for these devices should be explored in future work.

References

[1] J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping Target Signatures Via Partial Unmixing of Aviris Data", *Proc. JPL Airborne Earth Sci. Workshop*, 1995, pp. 23–26.

[2] C.-I. Chang, *Hyperspectral Data Exploitation: Theory and Applications*, John Wiley & Sons: New York, 2007.

[3] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for Earth remote sensing", *Science*, 228, 1985, pp. 1147–1153.

[4] R. O. Green et al., "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)", *Remote Sensing of Environment*, 65(3), 1998, pp. 227–248.

[5] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator", *ACM Transactions on Modeling and Computer Simulation*, 8(1), 1998, pp. 3–30.

[6] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*, CRC Press: Boca Raton, FL, 2007.

[7] A. Plaza and C.-I. Chang, "Clusters versus FPGA for parallel processing of hyperspectral imagery", *International Journal of High Performance Computing Applications*, 22(4), 2008, pp. 366–385.

[8] 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*, 42(3), 2004, pp. 650–663.

[9] A. Plaza, D. Valencia, and J. Plaza, "An experimental comparison of parallel algorithms for hyperspectral analysis using homogeneous and heterogeneous networks of workstations", *Parallel Computing*, 34(2), 2008, pp. 92–114.

[10] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral Imagery", *Journal of Parallel and Distributed Computing*, 66(3), 2006, pp. 345–358.

[11] F. Seinstra and D. Koelma, "User transparency: A fully sequential programming model for efficient data parallel image processing", *Concurrency and Computation: Practice & Experience*, 16(6), 2004, pp. 611–644.

[12] J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware", *IEEE Geoscience and Remote Sensing Letters*, 43(3), 2007, pp. 441–445.

[13] D. Valencia, A. Lastovetsky, M. O'Flynn, A. Plaza, and J. Plaza, "Parallel processing of remotely sensed hyperspectral images on heterogeneous networks of workstations using HeteroMPI", *International Journal of High Performance Computing Applications*, 22(4), 2008, pp. 386–407.

Authorized licensed use limited to: Universidad de Extremadura. Downloaded on November 16, 2009 at 08:18 from IEEE Xplore. Restrictions apply.