High Performance Computing for Hyperspectral Remote Sensing

Antonio Plaza, *Senior Member, IEEE*, Qian Du, *Senior Member, IEEE*, Yang-Lang Chang, *Senior Member, IEEE*, and Roger L. King, *Senior Member, IEEE*

***Abstract*—Advances in sensor and computer technology are revolutionizing the way remotely sensed data is collected, managed and analyzed. In particular, many current and future applications of remote sensing in Earth science, space science, and soon in exploration science will require real- or near real-time processing capabilities. In recent years, several efforts have been directed towards the incorporation of high-performance computing (HPC) models to remote sensing missions. A relevant example of a remote sensing application in which the use of HPC technologies (such as parallel and distributed computing) is becoming essential is hyperspectral remote sensing, in which an imaging spectrometer collects hundreds or even thousands of measurements (at multiple wavelength channels) for the same area on the surface of the Earth. In this paper, we review recent developments in the application of HPC techniques to hyperspectral imaging problems, with particular emphasis on commodity architectures such as clusters, heterogeneous networks of computers, and specialized hardware devices such as field programmable gate arrays (FPGAs) and commodity graphic processing units (GPUs). A quantitative comparison across these architectures is given by analyzing performance results of different parallel implementations of the same hyperspectral unmixing chain, delivering a snapshot of the state-of-the-art in this area and a thoughtful perspective on the potential and emerging challenges of applying HPC paradigms to hyperspectral remote sensing problems.**

***Index Terms*—Cluster computing, FPGAs, GPUs, hardware implementations, heterogeneous computing, high performance computing (HPC), hyperspectral remote sensing.**

I. Introduction

HYPERSPECTRAL remote sensing 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]. The wealth of spectral information available from latest-generation hyperspectral imaging instruments, which have substantially increased their spatial, spectral and temporal resolutions, has quickly introduced new challenges in the analysis and interpretation of hyperspectral data sets. For instance, the NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) [2] is now able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 μm) of the reflected light of an area 2 to 12 kilometers wide and several kilometers long using 224 spectral bands. The resulting data cube (see Fig. 1) is a stack of images 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 Gigabytes per flight. This often leads to the requirement of HPC infrastructure to accelerate hyperspectral-related computations, in particular, in analysis scenarios with real-time constraints [3]. It is expected that, in future years, hyperspectral sensors will continue increasing their spatial, spectral and temporal resolutions (imagers with thousands of spectral bands are currently in operation or under development [4]). Such wealth of information has opened groundbreaking perspectives in several applications [5] (many of which with real-time processing requirements) such as environmental modeling and assessment for Earth-based and atmospheric studies, risk/hazard prevention and response including wild land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination, target detection for military and defense/security purposes, urban planning and management studies, etc.

The utilization of HPC infrastructure in hyperspectral imaging applications has become more widespread in recent years [6]. The idea developed by the computer science community of using personal computers (PCs) clustered together to work as a computational 'team,' was a very attractive solution for remote sensing data processing from the beginning [7]. This strategy, often referred to as cluster computing [8], has already offered access to greatly increased computational power at a low cost (commensurate with falling commercial PC costs) in a number of hyperspectral imaging applications [9]–[14]. Although most parallel techniques and systems for hyperspectral image information processing employed during the last decade have chiefly been homogeneous in nature (i.e., they are made up of identical processing units, thus simplifying the design of parallel solutions adapted to those systems), a recent trend in the design of HPC systems for data-intensive problems such as those involved in hyperspectral image analysis is to utilize highly heterogeneous computing resources [15]–[17]. This heterogeneity is seldom planned, arising mainly as a result of technology evolution over time and computer market sales and trends. In this regard, networks of heterogeneous

Manuscript received August 31, 2010; revised November 05, 2010; accepted November 16, 2010. Date of publication January 06, 2011; date of current version August 26, 2011.

A. Plaza is with the Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain.

Q. Du is with the Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, MS 39762 USA.

Y.-L. Chang is with the Department of Electrical Engineering, National Taipei University of Technology, Taipei 10608, Taiwan.

R. L. King is with the Center for Advanced Vehicular Systems, Mississippi State University, Mississippi State, MS 39762 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTARS.2010.2095495

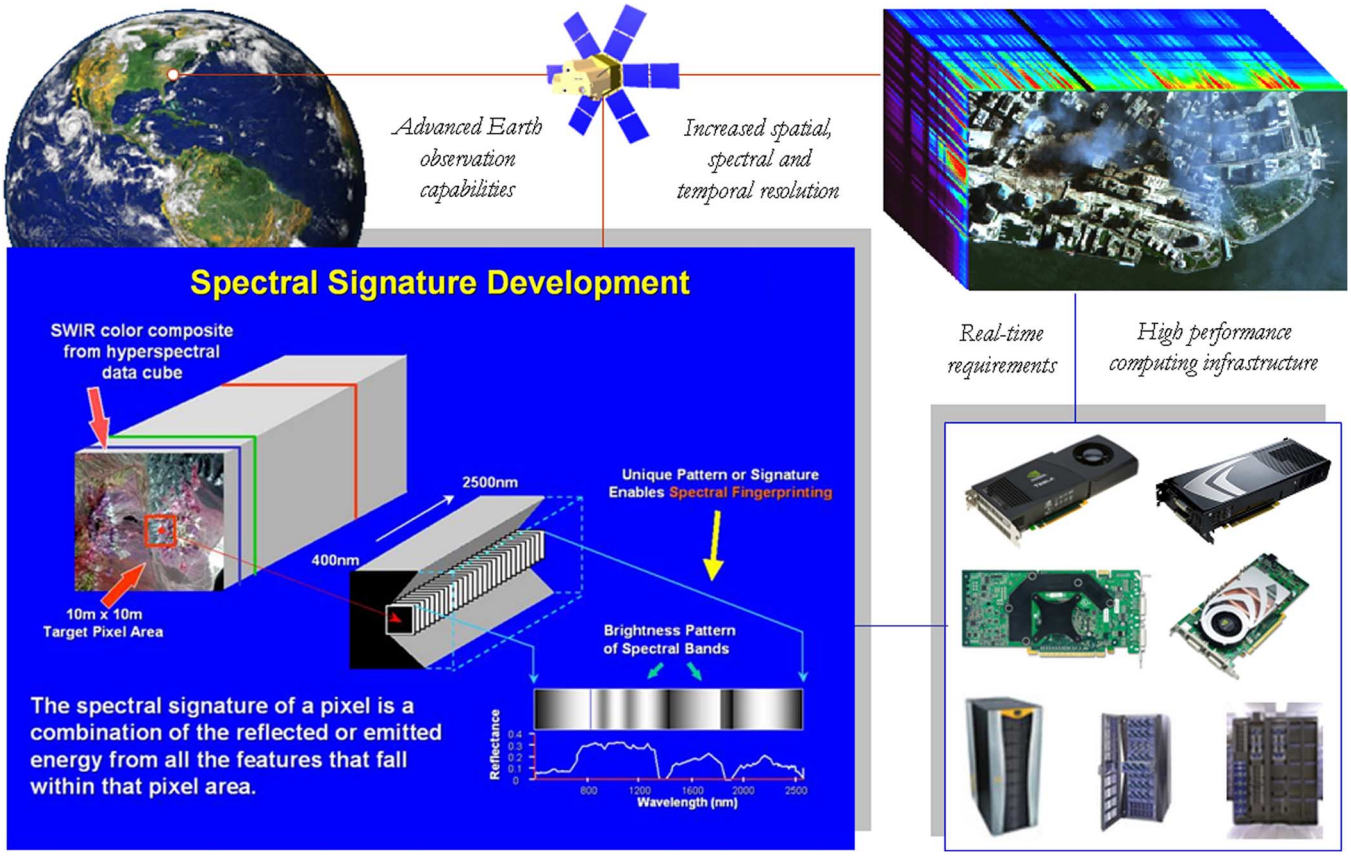

Fig. 1. An illustration of the processing demands introduced by the ever increasing dimensionality of remotely sensed hyperspectral imaging instruments.

PCs can realize a very high level of aggregate performance in hyperspectral imaging applications [18], and the pervasive availability of these resources resulted in the current notions of *grid* and *cloud* computing, which are yet to be fully exploited in hyperspectral imaging problems [19], [20].

Even though hyperspectral image processing algorithms generally map quite nicely to parallel systems made up of commodity PCs, 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 [3]. In this regard, an exciting new development in the field of commodity computing is the emergence of programmable hardware devices such as field programmable gate arrays (FPGAs) [21] and graphic processing units (GPUs) [22], which can bridge the gap towards on-board and real-time analysis of hyperspectral data [23], [24]. FPGAs are now fully reconfigurable [25], which allows one to adaptively select a data processing algorithm (out of a pool of available ones) to be applied on-board the sensor from a control station on Earth [26]. On the other hand, the emergence of GPUs (driven by the ever-growing demands of the video-game industry) has allowed these systems to evolve from expensive application-specific units into highly parallel and programmable commodity components [27]. For instance, the latest-generation GPU architectures from NVidia (*Tesla* and *Fermi* series) now offer cards able to deliver up to 515 Gigaflops of double-precision peak performance,[1] which is several times the performance of the fastest quad-core processor available. The ever-growing computational demands of remote sensing applications can fully benefit from compact hardware components and take advantage of the small size and relatively low cost of these units as compared to clusters or networks of computers [28], [29]. Last but not least, remote sensing applications can greatly benefit from the most recent developments in HPC and cluster computing, including the advent of GPU clusters (or clusters with other special *add-ons* and hardware accelerators, including FPGAs) and the gradual increase in the number of cores for each cluster node. Today, most newly installed cluster systems have such special-purpose extensions and/or many-core nodes (even up to 48 cores, such as AMD's Magny-Cours[2]). In the near future, these systems may introduce significant advances in the way hyperspectral data sets are processed, stored and managed.

In this paper, we present a review of available techniques for HPC applied to hyperspectral imaging problems and further describe our experience in efficient implementation of a standard hyperspectral unmixing chain on four different types of parallel platforms: 1) commodity clusters, 2) heterogeneous networks of workstations, 3) FPGAs, and 4) GPUs. The description and comparison of several strategies for implementation of the same data processing approach is expected to provide a

[1]http://www.nvidia.com/object/product_tesla_M2050_M2070_us.html.

[2]http://www.amd.com/us/products/server/processors/6000-series-platform/.

perspective on recent developments in this active research area. The remainder of the paper is organized as follows. Section II describes relevant previous efforts in the field, such as the evolution of cluster computing in remote sensing applications, the emergence of distributed networks of computers as a cost-effective means to solve hyperspectral remote sensing problems, and the exploitation of specialized hardware architectures in on-board processing scenarios. Section III describes hyperspectral processing tasks and an application case study given by a full processing chain based on spectral unmixing, a widely used technique to analyze hyperspectral data with sub-pixel precision. Section IV discusses and inter-compares several HPC implementations of the considered hyperspectral unmixing chain, including a cluster-based parallel version, a variation of this version specifically tuned for heterogeneous computing environments, an FPGA-based implementation, and a GPU implementation. Section V provides an experimental comparison of the proposed parallel implementations, using different HPC architectures, in the context of a real application case study focused on the analysis of hyperspectral data collected by the AVIRIS instrument over the World Trade Center area in New York, just five days after the terrorist attacks of September 11th. Section VI concludes with some remarks. Finally, Section VII provides hints at plausible future research directions.

II. Related Work

This section first provides an overview of the evolution of cluster computing architectures in the context of hyperspectral remote sensing applications, from the initial developments in parallel systems at NASA centers to the systems currently being employed for this purpose. Then, we give an overview of recent advances in the application of heterogeneous computing to hyperspectral imaging problems. The section concludes with an overview of hardware-based implementations intended for on-board processing of this type of high-dimensional data.

A. Cluster Computing in Hyperspectral Remote Sensing

Clusters [8] were originally developed with the purpose of creating a cost-effective parallel computing system able to satisfy specific computational requirements in the Earth and space sciences community. Initially, the need for large amounts of computation was identified for processing multi-spectral imagery with tens of bands [30]. As sensor instruments incorporated hyperspectral capabilities, it was soon recognized that computer mainframes and mini-computers could not provide sufficient power for effectively processing this kind of data [7].

In the midnineties, a team was put together at NASA's Goddard Space Flight Center (GSFC) to build a cluster consisting only of commodity hardware (PCs) running Linux, which resulted in the first cluster [7]. It consisted of 16 identical PCs with central processing units (CPUs) working at clock frequency of 100 MHz, connected with 2 hub-based Ethernet networks tied together with channel bonding software so that the 2 networks acted like one network running at twice the speed. A similar system, the 'Zoo'-cluster [31], was also installed the same year at Vrije Universiteit Amsterdam, which was at that time already the fourth-generation. The next year Beowulf-II, a 16-PC cluster based on 100 MHz Pentium PCs, was built and performed about 3 times faster, but also demonstrated a much higher reliability. In 1996, Pentium-Pro cluster at California Institute of Technology (Caltech) demonstrated a sustained performance of one Gigaflop on a remote sensing-based application. This was the first time a commodity cluster had shown high performance potential in this context. Up until 1997, clusters were in essence engineering prototypes, that is, they were built by those who were going to use them. However, in 1997 a project was started at GSFC to build a commodity cluster that was intended to be used by those who had not built it, the HIVE (highly parallel virtual environment) project [11]. The idea was to have workstations distributed among different locations and a large number of compute nodes (the compute core) concentrated in one area. The workstations would share the compute core as though it was apart of each. Although the original HIVE only had one workstation, many users were able to access it from their own workstations over the Internet. The HIVE was also the first commodity cluster to exceed a sustained peak performance of 10 Gigaflops on a remote sensing data processing. Later, an evolution of the HIVE was used at GSFC for hyperspectral remote sensing data processing calculations. The system, called Thunderhead, was a 512-processor homogeneous cluster composed of 256 dual 2.4 GHz Intel Xeon nodes, each with 1 Gigabyte of memory and 80 Gigabytes of main memory.[3] The total peak performance of the system was 2457.6 Gigaglops. Along with the 512-processor computer core, Thunderhead has several nodes attached to the core with 2 GHz Myrinet network connectivity. This system has been employed in several hyperspectral image analysis studies over the last few years [5], [13], [14], [32].

It is worth noting that NASA and the European Space Agency (ESA) are currently supporting additional massively parallel clusters for remote sensing applications, such as the Columbia supercomputer[4] at NASA Ames Research Center, a 10,240-CPU SGI Altix supercluster, with Intel Itanium 2 processors, 20 terabytes total memory and heterogeneous interconnects including InfiniBand network and 10 gigabit Ethernet. Another massively parallel system which has been exploited for hyperspectral imaging applications is MareNostrum,[5] an IBM cluster with 10,240 GPUs, 2.3 GHz Myrinet network connectivity and 20,480 GB of main memory available at Barcelona Supercomputing Center [33]. Finally, the High Performance Computing Collaboratory (HPC^2) at Mississippi State University[6] has several supercomputing facilities that have been used in different remote sensing studies. These machines are illustrative of how rapidly the computing environment is changing. The Maverick system[7] debuted in 2003 and is composed of 192×335 IBM nodes with each node containing a dual 3.06 GHz Xeon processor and 2.5 GB of RAM; Raptor came on-line in 2006 with a 2048 core cluster composed of 512 Sun Microsystems SunFire X2200 M2 servers, each with two dual-core AMD Opteron 2218 processors (2.6 GHz) and

[3]http://thunderhead.gsfc.nasa.gov.

[4]http://www.nas.nasa.gov/Resources/Systems/columbia.html.

[5]http://www.bsc.es/plantillaA.php?cat_id=5.

[6]http://www.hpc.msstate.edu.

[7]http://www.hpc.msstate.edu/computing/maverick.

8 GB of memory; and Talon in 2010 with a 3072 core cluster composed of 256 IBM iDataPlex nodes, each with two six-core Intel Westmere processors (2.8 GHz) and 24 GB of memory.

B. Heterogeneous Parallel Computing in Hyperspectral Remote Sensing

With the commercial availability of networking hardware at lower costs, it soon became obvious that networked groups of machines distributed among different locations could be used together by one single parallel remote sensing code as a distributed-memory machine [34]. Of course, such networks were originally designed and built to connect heterogeneous sets of machines. As a result, heterogeneous networks of computers have soon become a very popular tool for distributed computing with essentially unbounded sets of machines, in which the number and location of machines may not be explicitly known [12] as opposed to cluster computing, in which the number and location of nodes are known and relatively fixed. An evolution of the concept of distributed computing described above resulted in the notion of grid computing [19], in which the number and location of nodes is relatively dynamic and have to be discovered at run-time.

There are currently several ongoing research efforts aimed at efficient distributed processing of hyperspectral image data. Perhaps the most simple example is the use of heterogeneous versions of data processing algorithms developed for clusters, for instance, by resorting to heterogeneous-aware variations of homogeneous algorithms able to capture the inherent heterogeneity of a network of computers and to load-balance the computation among the available resources [15]–[17]. This framework allows one to easily port an existing parallel code developed for a homogeneous system to a fully heterogeneous environment. Another approach has been the adaptation of algorithms designed to be run in a single cluster to distributed multicluster environments made up of homogeneous clusters interconnected via wide-area network connectivity based on light paths [35]. Another example is the use of software components, which have been used in the framework of a plug-and-play environment for the construction of advanced processing algorithms (e.g., for performing atmospheric correction of hyperspectral data [20]) through a set of software components that conform to standardized interfaces. Such components encapsulate much of the complexity of the data processing algorithms inside a black box and expose only well-defined interfaces (e.g., input and output parameters) to other components. Finally, there have been several efforts in the recent literature oriented towards adapting the notion of grid computing to hyperspectral image processing, with the ultimate idea of providing wide accessibility (with fault tolerance achieved through careful *middleware* design) and efficient distribution and sharing of large hyperspectral data collections among different research centers [19], [20], [34].

C. Specialized Hardware for On-Board Hyperspectral Image Processing

Over the last few years, several research efforts have been directed towards the incorporation of specialized hardware for accelerating hyperspectral imaging calculations on-board airborne and satellite sensor platforms [3]. Enabling on-board data processing introduces many advantages, such as the possibility to reduce the data downlink bandwidth requirements by both pre-processing data and selecting data to be transmitted based upon some predetermined content-based criteria [24]. For instance, previous work has explored the possibility of compressing the hyperspectral data before it is transmitted to Earth, using both lossless and lossy compression strategies [36]. It should be noted that this kind of on-board processing also has the potential to reduce the cost and the complexity of ground processing systems so that they can be affordable to a larger community. Among existing hyperspectral imaging applications that can benefit from this strategy, we can list future planetary exploration missions (including those involving the analysis of hyperspectral data collected over the surface of Mars [37], for which autonomous decisions may be taken on-board, without the need for costly and slow communications with the ground segment.

The appealing perspectives introduced by specialized data processing components, such as FPGAs (on-the-fly reconfigurability [29]) and GPUs (very high performance at low cost [28]) introduce significant advantages with regards to traditional cluster-based systems for on-board hyperspectral data exploitation. First and foremost, a cluster occupies much more space than an FPGA or a GPU. This aspect significantly limits the exploitation of cluster-based systems in on-board processing scenarios, in which the weight (and the power consumption) of processing hardware must be limited in order to satisfy mission payload requirements [3]. If the cluster system is distributed across different locations, the space requirements increase. On the other hand, the maintenance of a large cluster represents a major investment in terms of time and finance. Although a cluster is a relatively inexpensive parallel architecture, the cost of maintaining a cluster can increase significantly with the number of nodes [13]. In turn, FPGAs and GPUs are characterized by their low weight and size, and by their capacity to provide similar computing performance at lower costs (recent studies demonstrate that the performance increase obtained by FPGAs [14], [23], [26], [29] and GPUs [27], [28], [38] in the context of hyperspectral imaging applications is in the same order of magnitude with respect to that achieved by a compute cluster with a high number of nodes. In addition, FPGAs offer the appealing possibility of adaptively selecting a hyperspectral processing algorithm to be applied (out of a pool of available algorithms) from a control station on Earth. This feature is possible thanks to the inherent re-configurability of FPGA devices [21], which are generally more expensive than GPU devices [28]. In this regard, the adaptivity of FPGA systems for on-board operation, as well as the low cost and portability of GPU systems, open many innovative perspectives.

To conclude this section we emphasize that, despite their appealing properties, both FPGAs and GPUs are still prone to further developments in order to fully achieve their incorporation to future Earth observation missions. For instance, the very fine granularity of FPGAs is still not efficient, with extreme situations in which only a small percentage of available resources are available for logic while the rest is mostly used for interconnect and configuration. This usually results in a

penalty in terms of speed and power, which is particularly cumbersome in radiation-hardened FPGAs (needed for spaceborne operation) which generally have two orders of magnitude fewer equivalent gates than commercial FPGAs [29]. Regarding GPUs, their high power consumption as compared to FPGAs introduces some considerations from the viewpoint of mission payload, although *green* GPUs with lower energy consumption are already available.[8] Further advances in the design of radiation-hardened GPUs are also expected in future years.

III. Hyperspectral Image Processing Tasks

The number and variety of image processing tasks in hyperspectral remote sensing is enormous [5]. However, the majority of algorithms can be organized according to the following specific tasks [39]:

- *Dimensionality reduction* consists of reducing the dimensionality of the input hyperspectral scene in order to facilitate subsequent processing tasks.
- *Target and anomaly detection* consist of searching the pixels of a hyperspectral data cube for "rare" (either known or unknown) spectral signatures.
- *Change detection* consists of finding the "significant" (i.e., important to the user) changes between two hyperspectral scenes of the same geographic region.
- *Classification* consists of assigning a label (class) to each pixel of a hyperspectral data cube.
- *Unmixing* consists of estimating the fraction of the pixel area covered by each material present in the scene.

In this paper, we focus on hyperspectral unmixing as a case study to demonstrate HPC implementations. No matter the spatial resolution, the spectral signatures collected in natural environments are invariably a mixture of the signatures of the various materials found within the spatial extent of the ground instantaneous field view of the imaging instrument [40], [41]. The availability of hyperspectral instruments with a number of spectral bands that exceeds the number of spectral mixture components allow us to approach this problem as follows. Given a set of spectral vectors acquired from a given area, spectral unmixing aims at inferring the pure spectral signatures, called *endmembers* [42], [43], and the material fractions, called *fractional abundances* [44], at each pixel. 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 $\mathbf{f}_i = [f_{i1}, f_{i2}, \cdots, f_{\mathrm{in}}] \in \Re^N$, 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 [40], each pixel vector can be modeled using the following expression [45]:

$$\mathbf{f}_i = \sum_{j=1}^{p} \mathbf{e}_j \cdot \Phi_j + \mathbf{n} \tag{1}$$

where $\mathbf{e}_j$ denotes the spectral response of an endmember, Φ_j is a scalar value designating the fractional abundance of the endmember $\mathbf{e}_j$, p is the total number of endmembers, and $\mathbf{n}$ is a noise vector. The solution of the linear spectral mixture problem described in (1) relies on the correct determination of a set $\{\mathbf{e}_j\}_{j=1}^{p}$ of endmembers and their correspondent abundance fractions $\{\Phi_j\}_{j=1}^{p}$ at each pixel $\mathbf{f}_i$. Two physical constrains are generally imposed into the model described in (1), these are the abundance non-negativity constraint (ANC), i.e., $\Phi_j \geq 0$, and the abundance sum-to-one constraint (ASC), i.e., $\sum_{j=1}^{p} \Phi_j = 1$ [44].

In this work we have considered a standard hyperspectral unmixing chain which is available in commercial software packages such as ITTVis Environment for Visualizing Images (ENVI).[9] The unmixing chain consists of two main steps: 1) endmember extraction using the pixel purity index (PPI) algorithm [46], and 2) fully constrained (i.e. ASC-constrained and ANC-constrained) abundance estimation [44]. The inputs to the unmixing chain are a hyperspectral image cube $\mathbf{F}$ with N spectral bands; 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 during the process. The output is a set of fractional abundances $\{\Phi_j\}_{j=1}^{p}$ for each pixel $\mathbf{f}_i$ of the hyperspectral image $\mathbf{F}$. 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:

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

4) *Endmember selection*. Find the pixels with value of $N_{times}(\mathbf{f}_i)$ above v_c and form a unique set of p endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$ by calculating the spectral angle (SA) [41], [47] for all possible endmember pairs and discarding those which result in an angle value below v_a. The SA is invariant to multiplicative scalings that may arise due to differences in illumination and sensor observation angle [45].
5) *Fully constrained spectral unmixing*. Once a set of endmembers $\mathbf{E} = \{\mathbf{e}_j\}_{j=1}^{p}$ has been obtained, an unconstrained abundance estimate in a specific pixel vector $\mathbf{f}_i$ can be obtained (in least squares sense) by the following expression [47]:

$$\hat{\Phi}_j^{\mathrm{UC}} = (\mathbf{E}^T\mathbf{E})^{-1}\mathbf{E}^T\mathbf{f}_i. \tag{3}$$

It should be noted that the abundance estimation in (3) does

[8] http://www.gputech.com.

[9] http://www.ittvis.com.

not satisfy the ANC and the ASC constraints usually imposed in the linear mixture model. An estimate satisfying the ASC constraint can be obtained by solving the following optimization problem:

$$\min_{\Phi_j \in \Delta} \left\{ (\mathbf{f}_i - \Phi_j \cdot \mathbf{E})^T (\mathbf{f}_i - \Phi_j \cdot \mathbf{E}) \right\}, \quad \text{subject to: } \Delta = \left\{ \Phi_j \left| \sum_{j=1}^{p} \Phi_j = 1 \right. \right\}. \quad (4)$$

Similarly, imposing the ANC constraint results in the following optimization problem:

$$\min_{\Phi_j \in \Delta} \left\{ (\mathbf{f}_i - \Phi_j \cdot \mathbf{E})^T (\mathbf{f}_i - \Phi_j \cdot \mathbf{E}) \right\}, \quad \text{subject to: } \Delta = \{ \Phi_j | \Phi_j \geq 0 \text{ for all } j \}. \quad (5)$$

As indicated in [48], a non-negative constrained least squares (NCLS) algorithm can be used to obtain a solution to the ANC-constrained problem described in (5) in iterative fashion [49]. In order to take care of the ASC constraint, a new endmember signature matrix, denoted by $\mathbf{E}'$, and a modified version of the abundance vector Φ_j, denoted by Φ'_j, are introduced as follows:

$$\mathbf{E}' = \begin{bmatrix} \delta \mathbf{E} \\ \mathbf{1}^T \end{bmatrix}, \quad \Phi'_j = \begin{bmatrix} \delta \Phi_j \\ 1 \end{bmatrix} \quad (6)$$

where $\mathbf{1} = (\underbrace{1, 1, \cdots, 1}_{p})^T$ and δ controls the impact of the ASC constraint. Using the two expressions in (6), a fully constrained estimate can be directly obtained from the NCLS algorithm by replacing $\mathbf{E}$ and Φ_j used in the NCLS algorithm with $\mathbf{E}'$ and Φ'_j, respectively.

IV. Parallel Implementations of the Hyperspectral Unmixing Chain

This section first develops a parallel implementation of the hyperspectral unmixing chain described in Section III which has been specifically designed to be run on massively parallel, homogeneous clusters. Then, the parallel version is transformed into a heterogeneity-aware implementation by introducing an adaptive data partitioning algorithm specifically developed to capture the specificities of the underlying heterogeneous networks of distributed workstations. Finally, both FPGA and GPU implementations are also described.

A. Cluster-Based Implementation

To reduce code redundancy and enhance reusability, our goal when designing the cluster-based implementation was to reuse much of the code for the sequential algorithm. For that purpose, we adopted a spatial-domain decomposition approach [50] that subdivides the image cube into multiple blocks made up of entire pixel vectors, and assigns each block to a different processing element in the cluster. It should be noted that our hyperspectral unmixing chain is mainly based on calculations in which pixels are always treated as entire spectral signatures. Therefore, a spectral-domain partitioning scheme (i.e. subdividing the multi-band data into sub-volumes across the spectral dimension) is not appropriate in our application domain because the calculations made for each hyperspectral pixel would need to originate from several processing elements, thus requiring intensive inter-processor communication [13]. Therefore, in our implementation a simple master-slave approach is implemented in which the master processor distributes spatial-domain partitions of the data to the workers and coordinates their actions. Then, the master gathers the partial results provided by the workers and produces a global result. The parallel algorithm is given by the following steps:

1) *Data partitioning*. The master processor produces a set of W equally-sized spatial-domain partitions of the hyperspectral image $\mathbf{F}$ and scatters all partitions by indicating all partial data structure elements which are to be accessed and sent to each of the workers.
2) *Skewer generation*. The master generates K random unit vectors $\{\mathbf{skewer}_j\}_{j=1}^{K}$, and broadcasts the entire set of skewers to all the workers.
3) *Extreme projections*. For each $\mathbf{skewer}_j$, project all the sample pixel vectors at each local partition w 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 $S^{(w)}_{extrema}(\mathbf{skewer}_j)$. Now calculate the number of times each pixel vector $\mathbf{f}_i^{(w)}$ in the local partition w is selected as extreme using the following expression:
$$N^{(w)}_{times}\left(\mathbf{f}_i^{(w)}\right) = \sum_{j=1}^{K} I_{S^{(w)}_{extrema}(\mathbf{skewer}_j)}\left(\mathbf{f}_i^{(w)}\right). \quad (7)$$
4) *Candidate selection*. Each worker now sends the number of times that each pixel vector in the local partition has been selected as extreme to the master, which forms a final matrix of pixel purity indexes N_{times} by combining all the individual matrices $N^{(w)}_{times}$ provided by the workers.
5) *Endmember selection*. The master selects those pixels with $N_{times}(\mathbf{f}_i) > v_c$ and forms a unique set of p endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$ by calculating the SA for all possible pixel vector pairs and discarding those pixels which result in angle values below v_a.
6) *Fully constrained spectral unmixing*. The master broadcasts the set of endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$ to all the workers, and each worker locally computes a fully constrained abundance estimation for each pixel $\mathbf{f}_i^{(w)}$ in its local partition w. After the workers have computed their estimations locally, the master simply gathers the individual abundance estimation results.

To conclude this subsection, it is important to emphasize that several steps of this algorithm are purely sequential. This means that the master node performs some steps of the algorithm on its own. Nevertheless, the execution time of these purely sequential steps is insignificant in comparison to the total execution time (i.e. less than 1%). Moreover, as shown by the algorithm description, some communication steps between master and workers are required. However, the impact of communications was not particularly significant in our application, while most of the computations involved for endmember extraction and abundance estimation can be performed independently at each worker without additional memory management. In turn, other applications of cluster computing in remote sensing applications may have different results depending upon such issues

as degree of parallelization, amount of communication overhead in algorithms, and load balancing strategies used.

B. 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 [17]. Therefore, the parallel algorithm in Section IV-A needs to be adapted for efficient execution on heterogeneous computing environments. Two major goals of data partitioning in heterogeneous networks are [18]: 1) to obtain an appropriate set of workload fractions $\{\alpha_i\}_{i=1}^{W}$ that best fit the heterogeneous environment, and 2) to translate the chosen set of values into a suitable decomposition of total workload L, taking into account the properties of the heterogeneous system. In order to accomplish the above goals, we have developed a workload estimation algorithm (WEA) 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. Below, we provide a description of WEA, which replaces the *data partitioning* step in the parallel algorithm described in Section IV-A. Steps 2–6 of the parallel algorithm in Section IV-A would be executed immediately after WEA. The input to WEA is a hyperspectral data cube $\mathbf{F}$, and the output is a set of W spatial-domain heterogeneous partitions of $\mathbf{F}$:

1) Obtain necessary information about the heterogeneous system, including the number of workers W, each processor's identification number $\{w_i\}_{i=1}^{W}$, and processor cycle-times $\{t_i\}_{i=1}^{W}$.
2) Set $\alpha_i = \lfloor (1/t_i)/\sum_{i=1}^{W}(1/t_i) \rfloor$ for all $i \in \{1, \cdots, W\}$. In other words, this step first approximates the $\{\alpha_i\}_{i=1}^{W}$ so that the amount of work assigned to each processing node is proportional to its speed and $\alpha_i \cdot t_i \approx const$ for all processors.
3) Iteratively increment some α_i until the set of $\{\alpha_i\}_{i=1}^{W}$ best approximates the total workload L to be completed, i.e., for $m = \sum_{i=1}^{P} \alpha_i$ to L, find $k \in \{1, \cdots, W\}$ so that $t_k \cdot (\alpha_k + 1) = min\{t_i \cdot (\alpha_i + 1)\}_{i=1}^{W}$, and then set $\alpha_k = \alpha_k + 1$.
4) Produce W partitions of the input hyperspectral data set, so that the spectral channels corresponding to the same pixel vector are never stored in different partitions. In order to achieve this goal, we have adopted a methodology which consists of three main steps:
   a) The hyperspectral data set is first partitioned, using spatial-domain decomposition, into a set of vertical slabs which retain the full spectral information at the same partition. The number of rows in each slab is set to be proportional to the estimated values of $\{\alpha_i\}_{i=1}^{W}$, and assuming that no upper bound exist on the number of pixel vectors that can be stored by the local memory at the considered node.
   b) For each processor w_i, check if the number of pixel vectors assigned to it is greater than the upper bound. For all the processors whose upper bounds are exceeded, assign them a number of pixels equal to their upper bounds. Now, we solve the partitioning problem of a set with remaining pixel vectors over the remaining processors. We recursively apply this procedure until all pixel vectors in the input data have been assigned, thus obtaining an initial workload distribution for each w_i. It should be noted that, with the proposed algorithm description, it is possible that all processors exceed their upper bounds. This situation was never observed in our experiments. However, if the considered network includes processing units with low memory capacity, this situation could be handled by allocating an amount of data equal to the upper bound to those processors, and then processing the remaining data as an offset in a second algorithm iteration.
   c) Iteratively recalculate the workload assigned to each processor using the following expression:

$$L_i^k = L_i^{k-1} - \sum_{j \in N(i)} c_{ij} \left(\frac{L_i^{k-1}}{t_i} - \frac{L_j^{k-1}}{t_j} \right) \tag{8}$$

   where $N(i)$ denotes the set of neighbors of processing node w_i, and L_i^k denotes the workload of w_i (i.e., the number of pixel vectors assigned to this processor) after the k-th iteration. This scheme has been demonstrated in previous work to converge to an average workload $\overline{L_i} := (\sum_{j=1}^{W} L_j / \sum_{j=1}^{W} t_j) t_i$ [51].

The parallel heterogeneous algorithm has been implemented using the C++ programming language with calls to standard message passing interface (MPI)[10] library functions (also used in our cluster-based implementation).

C. FPGA Implementation

Our strategy for implementation of the hyperspectral unmixing chain in reconfigurable hardware is aimed at enhancing replicability and reusability of slices in FPGA devices through the utilization of systolic array design [14]. Fig. 2 describes our systolic architecture. Here, local results remain static at each processing element, while a total of T pixel vectors with N dimensions are input to the systolic array from top to bottom. Similarly, K skewers with N dimensions are fed to the systolic array from left to right. In Fig. 2, asterisks represent delays. The processing nodes labeled as dot in Fig. 2 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 avoid broadcasting the pixel while simplifying the collection of the results.

Based on the systolic array described above (which also allows implementation of the fully constrained spectral unmixing stage) we have implemented the full hyperspectral unmixing chain using the very high speed integrated circuit hardware description language (VHDL)[11] for the specification of the systolic array. Further, we have used the Xilinx ISE environment and the Embedded Development Kit (EDK) environment[12] to

[10]http://www.mcs.anl.gov/mpi.

[11]http://www.vhdl.org.

[12]http://www.xilinx.com/ise/embedded/edk_pstudio.htm.

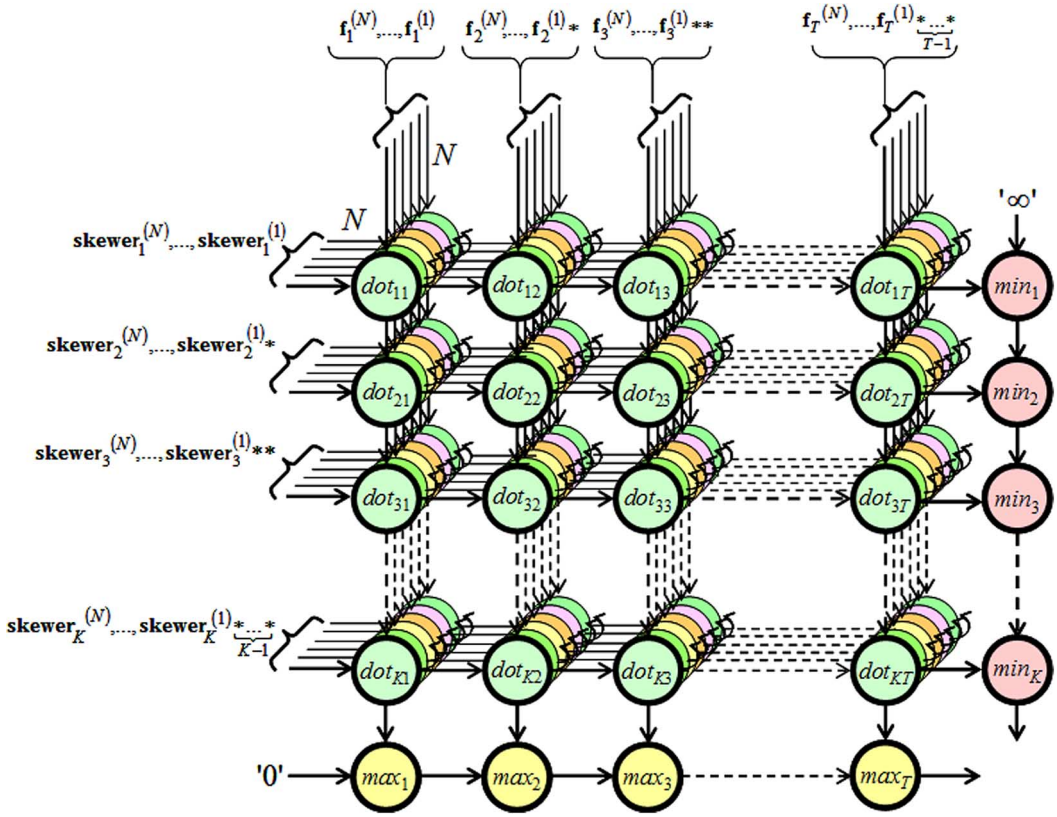

Fig. 2. Systolic array architecture used in our FPGA hardware implementation.

specify the complete system. The full system has been ported to a low-cost reconfigurable board of XUPV2P type with a single Virtex-II PRO xc2vp30 FPGA component. For data input we use the FPGA memory slot (of DDR2 SDRAM type) which holds up to 2 Gigabytes, and a direct memory access (DMA) module (controlled by a PowerPC) with a write queue to store the pixel data. A read queue and a transmitter are also used to send the endmembers to the FPGA via a RS232 port. A control unit, the systolic array, and a module for random generation of skewers are also implemented. Additional details about the considered hardware architecture are given in [29].

D. GPU Implementation

GPUs can be abstracted in terms of a *stream model* [22], under which all data sets are represented as streams (i.e., ordered data sets). Algorithms are constructed by chaining so-called *kernels*, which operate on entire streams, 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, as displayed in Fig. 3(a), 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. There is a maximum number of threads that a block can contain (512 in our considered architecture, the NVidia Tesla C1060 GPU) but the number of threads that can be concurrently executed is much larger (several blocks executed by the same kernel can be managed concurrently, at the expense of reducing the cooperation between threads since the threads in different blocks of the same grid cannot synchronize with the other threads). Fig. 3(a) displays how each kernel is executed as a grid of blocks of threads. On the other hand, Fig. 3(b) shows the execution model in the GPU, which can be seen as a set of multiprocesors (16 in our considered architecture). Each multiprocessor is characterized by a single instruction multiple data (SIMD) architecture, i.e., in each clock cycle each processor of the multiprocessor executes the same instruction but operating on multiple data streams. The use of a SIMD strategy in each multiprocessor of the GPU is in contrast with the more general multiple instruction multiple data (MIMD) strategy exploited in all other parallel implementations discussed in this paper. 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.

In order to implement the considered hyperspectral unmixing chain in a GPU, the first issue that needs to be addressed is how to map a hyperspectral image onto the global GPU memory. Since the size of hyperspectral images may exceed the capacity of the GPU memory, in that case we split the image into multiple spatial-domain partitions [13] made up of entire pixel vectors. In our considered architecture, the size of the global memory is 4 GB (more than enough to store a hyperspectral image of standard size). Fig. 4 shows a flowchart describing the kernels that comprise our GPU-based implementation, which has been developed using the compute unified device

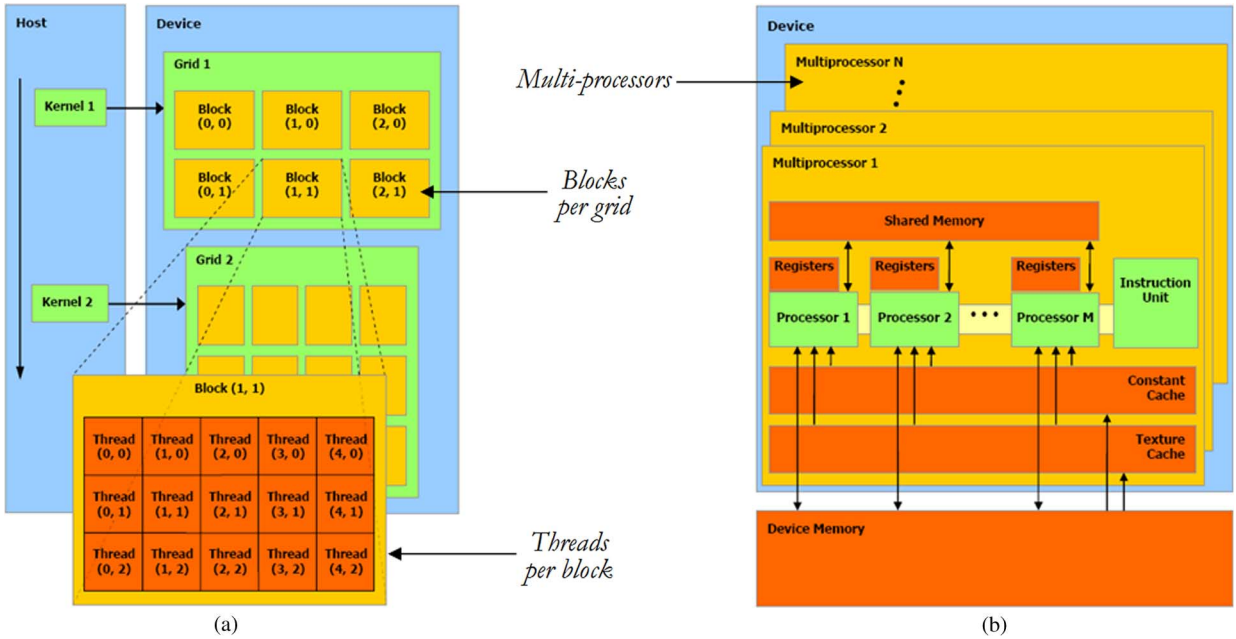

Fig. 3. Schematic overview of a GPU architecture. (a) Threads, blocks and grids. (b) Execution model in the GPU.

architecture (CUDA)[13]. The *data partitioning* kernel performs the spatial-domain decomposition of the original hyperspectral image. In the *stream uploading* kernel, the spatial-domain partitions are uploaded as a set of tiles onto the GPU memory. The *skewer generation* kernel provides the skewers, using NVidia's parallel implementation of the Mersenne twister pseudo-random number generator on the GPU [52]. The remaining stages comprise the following kernels:

- *Extreme projections*. In this kernel, each computing thread calculates the projection of all pixel vectors in the original data set onto each randomly generated skewer.
- *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 SA with all the other candidates. It is based on a single-pass kernel that computes the SA 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 SA scores below a threshold angle.
- *Spectral unmixing*. Finally, this kernel uses as inputs the final endmembers selected in the previous stage and produces fully constrained endmember fractional abundances for each pixel.

The most time-consuming kernel in our GPU implementation is the *extreme projections* kernel, denoted by `PPI` and displayed in Fig. 5 for illustrative purposes. The first parameter of the `PPI` kernel, (`d_image`), is the original hyperspectral image.

[13]http://www.nvidia.com/object/cuda_home_new.html.

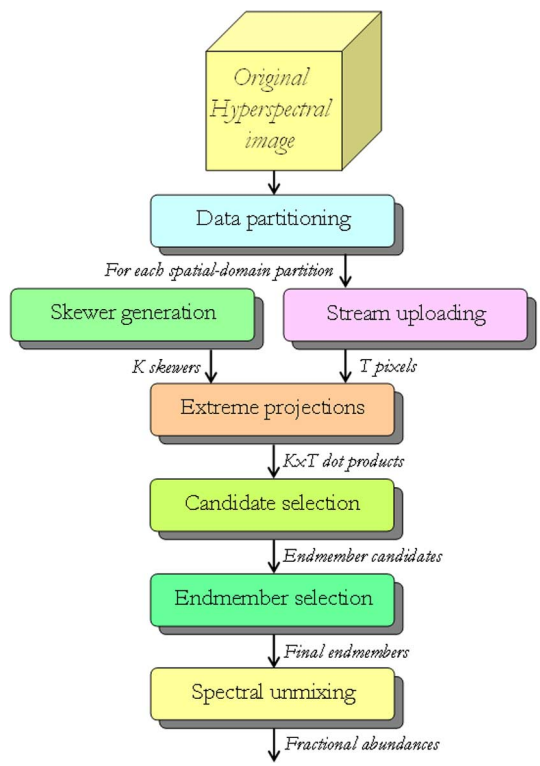

Fig. 4. Flowchart summarizing the kernels involved in our GPU implementation.

The second parameter is a structure that contains the randomly generated skewers. The third parameter (`d_res_partial`) is the structure in which the output of the kernel will be stored. The kernel also receives as input parameters the dimensions of the hyperspectral image, i.e., `num_lines`, `num_samples` and `num_bands`. The structure `l_rand` (local to each thread) is used to store a skewer through a processing cycle. It should be noted

```
__global__ void PPI (float *d_image, float *d_random, int *d_res_partial,
int num_lines, int num_samples, int num_bands)
{

int idx = blockDim.x * blockIdx.x+threadIdx.x;
float pemax; // Maximum value of dot product
float pemin; // Minimum value of dot product
float pe;    // Scalar product

int v,d; int imax = 0; int imin = 0; pemax = MIN_INT; pemin = MAX_INT;

__shared__ float s_pixels[Tam_Vector]; float l_rand[224];

//Copy a skewer from GPU global memory to GPU registers
for (int k=0; k < num_bands; k++){
      l_rand[k] = d_random[idx*num_bands+k];
}

for (int it = 0; it < num_lines*num_samples/N_Pixels; it++){

      //Copy N_Pixels pixels to shared memory
      if (threadIdx.x < N_Pixels){
      for (int j=0; j<num_bands; j++){
            s_pixels[threadIdx.x+N_Pixels*j] =
            d_imagen[(it*N_Pixels+threadIdx.x)+(num_lines*num_samples*j)];
      }
}

__syncthreads();

//For each pixel
for (v=0; v < N_Pixels; v++){

      //Calculate dot product
      pe = 0;
      for (d = 0; d < num_bands; d++){
            pe = pe + l_rand[d]*s_pixels[v+N_Pixels*d];
      }

      //Calculate extreme values
      if (pe > pemax){
            imax = it*N_Pixels+v; pemax = pe;
      }

      if (pe < pemin){
            imin = it*N_Pixels+v; pemin = pe;
      }
}

//Update d_res_partial structure
d_res_partial[idx*2] = imax;
d_res_partial[idx*2+1] = imin;
}
```

Fig. 5. CUDA Kernel `PPI` developed to implement the extreme projections step of the hyperspectral unmixing chain on the GPU.

that each thread works with a different skewer and the values of the skewer are continuously used by the thread in each iteration, hence it is reasonable to store skewers in the local memories associated to each thread since these memories are hundreds of times faster than the global GPU memory [see Fig. 3(b)]. The second structure used by the kernel is `s_pixels`, which is shared by all threads in the same block [see Fig. 3(a)]. Since each thread needs to perform the dot product using the same image pixels, it is reasonable to make use of a shared structure which can be accessed by all threads, thus avoiding that such threads access the global GPU memory separately. In our implementation, such shared structure is stored in the local shared memory of each multiprocessor, as displayed in Fig. 3(b). Since the `s_pixels` structure is also stored in local memory, the accesses are also much faster. It should be noted that the local memories in a GPU are usually quite small in order to guarantee very fast accesses, therefore the `s_pixels` structure can only accommodate a blocks of b pixels. In our implementation, this value has been set to $b = 10$ (the maximum number of pixels that can be allocated in a local GPU memory in the NVidia Tesla C1060 GPU [22]), hence in the following we assume that the hyperspectral image is processed in blocks of $b = 10$ pixels.

TABLE I
SPECIFICATIONS OF PROCESSORS IN A HETEROGENEOUS NETWORK OF COMPUTERS

Processor number	Architecture Specification	Cycle-time (seconds/megaflop)	Memory (MB)	Cache (KB)
w_1	Free BSD – i386 Intel Pentium 4	0.0058	2048	1024
w_2, w_5, w_8	Linux – Intel Xeon	0.0102	1024	512
w_3	Linux – AMD Athlon	0.0026	7748	512
w_4, w_6, w_7, p_9	Linux – Intel Xeon	0.0072	1024	1024
w_{10}	SunOS – SUNW UltraSparc-5	0.0451	512	2048
$w_{11} - w_{16}$	Linux – AMD Athlon	0.0131	2048	1024

Once a skewer and a group of $b = 10$ pixels are stored in the local memory associated to a thread, the next step is to perform the dot product between each pixel vector and the skewer. For this purpose, each thread uses two variables `pemin` and `pemax` which store the minima and maxima projection values, respectively, and other two variables `imin` and `imax` which store the relative index of the pixels resulting in the maxima and minima projection values. Once the projection process is finalized for a group of $b = 10$ pixels, another group is loaded. The process finalizes when all hyperspectral image pixels have been projected onto the skewer. Finally, the `d_res_partial` structure is updated with the minima and maxima projection values. This structure is reshaped into a matrix of pixel purity indexes by simply counting the number of times that each pixel was selected as extreme during the process and updating its associated score with such number, this producing a final structure `d_res_total` that stores the final values of $N_{times}(\mathbf{f}_i)$ for a given pixel $\mathbf{f}_i$.

V. EXPERIMENTAL RESULTS

This section is organized as follows. In Section V-A we describe the parallel computing architectures used for experiments. Section V-B describes the hyperspectral data set that will be used for demonstration purposes. Section V-C compares several different implementations of the considered hyperspectral unmixing chain in terms of sub-pixel analysis accuracy, using the hyperspectral data set described in Section V-B. Section V-D provides a technique comparison in terms of computational performance using the same hyperspectral data set. Finally, Section V-E provides an overall discussion in terms of both the achieved parallel efficiency and the time and effort spent in the creation of HPC solutions for each of the presented platforms.

A. Parallel Computing Architectures

Four different parallel computing platforms have been used in experiments:

- A cluster called Thunderhead at NASA's Goddard Space Flight Center in Maryland, 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.
- A heterogeneous network which consists of 16 different workstations and four different communication segments.

TABLE II
CAPACITY OF COMMUNICATION LINKS (IN TIME IN MILLISECONDS TO TRANSFER A ONE-MEGABIT MESSAGE) IN A HETEROGENEOUS NETWORK OF COMPUTERS

Processor	$w_1 - w_4$	$w_5 - w_8$	$w_9 - w_{10}$	$w_{11} - w_{16}$
$w_1 - w_4$	19.26	48.31	96.62	154.76
$w_5 - w_8$	48.31	17.65	48.31	106.45
$w_9 - w_{10}$	96.62	48.31	16.38	58.14
$w_{11} - w_{16}$	154.76	106.45	58.14	14.05

Table I shows the properties of the 16 heterogeneous workstations, where processors $\{w_i\}_{i=1}^{4}$ are attached to communication segment s_1, processors $\{w_i\}_{i=5}^{8}$ communicate through s_2, processors $\{w_i\}_{i=9}^{10}$ are interconnected via s_3, and processors $\{w_i\}_{i=11}^{16}$ share the communication segment s_4. For illustrative purposes, Table II also shows the capacity of all point-to-point communications in the heterogeneous network, expressed as the time in milliseconds to transfer a one-megabit message between each processor pair (w_i, w_j) in the heterogeneous system.

- A low-cost reconfigurable board with a single Virtex-II PRO xc2vp30 FPGA component, a DDR SDRAM DIMM memory slot with 2 Gigabytes of main memory, a RS232 port, and some additional components not used by our implementation.
- An NVidia Tesla C1060 GPU, which features 240 processor cores operating at 1.296 GHz, with single precision floating point performance of 933 Gigaflops, double precision floating point performance of 78 Gflops, total dedicated memory of 4 GB, 800 MHz memory (with 512-bit GDDR3 interface) and memory bandwidth of 102 GB/sec[14]. The GPU is connected to an Intel core i7 920 CPU at 2.67 GHz with 8 cores, which uses a motherboard Asus P6T7 WS SuperComputer.

B. Hyperspectral Data Set

The image scene used for experiments in this work was collected by the AVIRIS instrument, which was flown by NASA's Jet Propulsion Laboratory 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

[14]http://www.nvidia.com/object/product_tesla_c1060_us.html.

Fig. 6. AVIRIS hyperspectral image collected over the World Trade Center (left). Location of thermal hot spots in the World Trade Center complex (right).

TABLE III
COMPARISON OF AREA ESTIMATION (IN SQUARE METERS) FOR EACH THERMAL HOT SPOT BY DIFFERENT PARALLEL IMPLEMENTATIONS OF THE HYPERSPECTRAL UNMIXING CHAIN (USGS REFERENCE VALUES AND THE RESULTS OBTAINED USING ENVI SOFTWARE ARE ALSO INCLUDED)

Thermal hot spot	Latitude (North)	Longitude (West)	Temperature (Kelvin)	Area (USGS)	Area (ENVI)	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	0.53
'B'	$40^o42'47.14"$	$74^o00'43.53"$	830	0.08	0.06	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	0.78
'D'	$40^o42'41.99"$	$74^o00'46.94"$	790	0.80	0.81	0.81	0.81	0.83	0.83
'E'	$40^o42'40.58"$	$74^o00'50.15"$	710	0.40	0.55	0.55	0.59	0.57	0.57
'F'	$40^o42'38.74"$	$74^o00'46.70"$	700	0.40	0.36	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	0.05	0.05
'H'	$40^o42'38.60"$	$74^o00'43.51"$	820	0.08	0.12	0.12	0.09	0.07	0.11

other buildings in the WTC complex. The full data set selected for experiments consists of 614×512 pixels, 224 spectral bands and a total size of (approximately) 140 MB. The spatial resolution is 1.7 meters per pixel. The leftmost part of Fig. 6 shows a false color composite of the data set selected for experiments using the 1682, 1107 and 655 nm channels, displayed as red, green and blue, respectively. Vegetated areas appear green in the leftmost part of Fig. 6, while burned areas appear dark gray. Smoke coming from the WTC area (in the red rectangle) and going down to south Manhattan appears bright blue due to high spectral reflectance in the 655 nm channel.

Extensive reference information, collected by U.S. Geological Survey (USGS), is available for the WTC scene[15]. In this work, we use a USGS thermal map[16] which shows the target locations of the thermal hot spots at the WTC area, displayed as bright red, orange and yellow spots at the rightmost part of Fig. 6. The map is centered at the region where the towers collapsed. Further information available from USGS about the targets (including location, estimated size, and temperature) is reported in Table III. As shown by Table III, all the targets are sub-pixel in size since the spatial resolution of a single pixel is 1.7 square meters.

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

[16]http://pubs.usgs.gov/of/2001/ofr-01-0429/hotspot.key.tgif.gif.

C. Comparison of Techniques in Terms of Sub-Pixel Accuracy

Before empirically investigating the parallel performance of the proposed algorithms, we first evaluate their endmember extraction and spectral unmixing accuracy in the context of the considered application. Prior to a full examination and discussion of results, it is important to outline parameter values used for the considered unmixing chain. In all our considered implementations, the number of endmembers to be extracted was set to $p = 30$ after estimating the dimensionality of the data using the *virtual dimensionality* concept [48]. In addition, the number of skewers was set to $K = 10^4$ (although values of $K = 10^3$ and $K = 10^5$ were also tested, we experimentally observed that the use of $K = 10^3$ resulted in the loss of important endmembers, while the endmembers obtained using $K = 10^5$ were essentially the same as those found using $K = 10^4$). Finally, the threshold angle parameter was set to $v_a = 5^o$, which is a reasonable limit of tolerance for this metric, while the cut-off threshold value parameter v_c was set to the mean of N_{times} scores obtained after $K = 10^4$ iterations. These parameter values are in agreement with those used before in the literature [42].

Table III evaluates the accuracy of the four considered parallel implementations in the task of estimating the sub-pixel abundance of fires in the WTC scene, taking advantage of reference information about the area covered by each thermal hot

TABLE IV
TIMES (SECONDS) MEASURED AFTER PROCESSING THE AVIRIS WORLD TRACE CENTER HYPERSPECTRAL SCENE ON THE THUNDERHEAD CLUSTER AND ON THE HETEROGENEOUS NETWORK OF COMPUTERS

	Thunderhead cluster									Heterogeneous network
# CPUs	**1**	**4**	**16**	**36**	**64**	**100**	**144**	**196**	**256**	**16**
Sequential computations	1163.05	1.63	1.26	1.12	1.19	1.06	0.84	0.91	0.58	1.69
Parallel computations	0	292.09	73.24	30.46	15.44	8.76	5.08	3.18	1.91	79.56
Communications	0	2.20	2.41	2.39	2.21	2.46	2.65	2.32	2.49	3.56
Total time‡	1163.05	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

‡ The total time consumed by our FPGA implementation was 31.23 seconds, while our GPU implementation took 17.59 seconds

spot available from USGS. 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 unmixing chain. Experiments in Table III demonstrate that the parallel algorithms can provide accurate estimations of the area covered by thermal hot spots, which can lead to good characterization results (similar to those obtained using the unmixing chain available in ENVI software). 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.

It is important to note that the output produced by all parallel methods in Table III was verified using not only our own serial implementations, but the implementation of the considered hyperspectral unmixing chain available in the commercial version 4.5 of ENVI software available from ITTVis (using the same parameters in both cases). The endmembers found by our parallel implementations were exactly the same as the ones found by our serial implementations of the respective original algorithms. To arrive to this conclusion, we made sure that the same set of random skewers was used to guarantee that both the serial and parallel versions were exactly the same. It should be noted, however, that our parallel implementations produced slightly different endmembers than those found by ENVI's implementation. However, we experimentally tested that the spectral angle scores between the endmembers that were different between the original and parallel algorithms were always very low (below 0.85°), a fact that reveals that the final endmembers sets were almost identical in spectral sense.

D. Comparison of Techniques in Terms of Parallel Performance

Table IV shows the total time spent by the parallel implementation in communications and computations in the Thunderhead cluster and on the heterogeneous network. The time taken by an optimized serial implementation of the processing chain on a single Thunderhead node is also reported. 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 IV also displays the communication times, the total execution times, and the speedups (number of times that the parallel implementation was faster than the serial implementation [14]). The total execution time measured for the serial version on a single Thunderhead processor was 1163.05 seconds, while the same code executed on the fastest processor of the heterogeneous network was 1098.63 seconds.

It can be seen from Table IV that the times for sequential computations were always very low when compared to the time for parallel computations, which indicates high parallel efficiency of the developed implementations on both the cluster and the heterogeneous network, even for a high number of processors. On the other hand, it can also be seen from Table IV that the impact of communications was not particularly significant. Interestingly, the speedup achieved in the heterogeneous network (16 CPUs) is similar to that achieved in the cluster with the same number of CPUs. To further explore the issue of load balance (which is particularly crucial in heterogeneous platforms), we have calculated the imbalance scores achieved by our heterogeneous implementation based on the WEA algorithm in the considered heterogeneous network. The imbalance is simply defined as $D = R_{max}/R_{min}$, where R_{max} and R_{min} are respectively the maxima and minima processor run times measured across the set of heterogeneous processors. Therefore, perfect balance is achieved when $D = 1$. In our study, we measured the imbalance considering all processors, D_{all}, and also considering all processors but the master, D_{minus}. The values measured were, respectively, $D_{all} = 1.19$ and $D_{minus} = 1.05$. These values indicate that our implementation is well balanced, and also that the workload assigned to the master node is balanced with regards to that assigned to the workers. Similar values ($D_{all} = 1.04$ and $D_{minus} = 1.01$) were measured on the homogeneous cluster for 256 processors. In this case, the total processing time for the full unmixing chain was 4.98 seconds. It should be noted that this result is in real time since the cross-track line scan time in AVIRIS, a push-broom instrument, is quite fast (8.3 milliseconds to collect 512 full pixel vectors). This introduces the need to process the considered scene (614×512 pixels) in approximately 5.09 seconds to fully achieve real time performance. Both types of parallel computing architectures (cluster and heterogeneous network) are very appealing for information extraction from hyperspectral data already transmitted to Earth; it is estimated that a significant portion of hyperspectral image data sets collected by airborne/satellite instruments are never processed

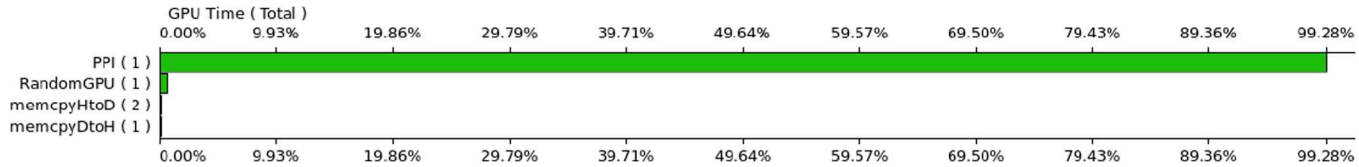

Fig. 7. Summary plot describing the percentage of the total time consumed by different kernels after processing the AVIRIS World Trade Center hyperspectral scene in the NVidia Tesla C1060 GPU.

but directly stored in databases, hence these systems offer an unprecedented opportunity for efficient information mining from large data repositories. However, these systems cannot be adapted to on-board processing scenarios in which low-weight, compact hardware devices are needed. In contrast, specialized hardware devices such as FPGAs and GPUs are more appealing for on-board processing.

Our implementation of the hyperspectral unmixing chain on the considered FPGA board was able to achieve a total processing time for the considered AVIRIS scene of 31.23 seconds (approximately the same as the one achieved using 36 Thunderhead processors) utilizing approximately 76% of the available hardware resources in the considered board. This FPGA has a total of 13696 slices (out of which 10423 were used by our implementation). An interesting feature of the considered design is that it can be scaled without significantly increasing the delay of the critical path (the clock cycle remained constant at 187 MHz). It should also be noted that in our implementation he have paid special attention to the impact of communications. In previous designs [14], [25], the module for random generation of skewers was situated in an external processor. Hence, frequent communications between the host (CPU) and the device (FPGA) were needed. However, in our implementation the hardware random generation module is implemented internally in the FPGA board. This approach significantly reduced the communications, leading to increased parallel performance.

Finally, the execution time for the parallel hyperspectral unmixing chain implemented on the NVidia C1060 Tesla GPU was 17.59 seconds (approximately the same as the one achieved using 64 Thunderhead processors and closer to real-time performance than the FPGA implementation). In our experiments on the Intel Core i7 920 CPU, the hyperspectral unmixing chain took 1078.03 seconds to process the considered AVIRIS scene (using just one of the available cores). This means that the speedup achieved by our GPU implementation with regards to the serial implementation in one core was approximately 61.28. For illustrative purposes, Fig. 7 shows the percentage of the total execution time consumed by the `PPI` kernel described in Fig. 5, which implements the *extreme projections* step of the unmixing chain, and by the `RandomGPU` kernel, which implements the *skewer generation* step in the chain. Fig. 7 also displays the number of times that each kernel was invoked (in the parentheses). These values were obtained after profiling the implementation using the CUDA Visual Profiler tool[17]. In the figure, the percentage of time for data movements from host (CPU) to device (GPU) and from device to host are also displayed. It should be noted that two data movements from host to device are needed to transfer the original hyperspectral image and the skewers to the GPU, while only one movement from device to host (final result) is needed. As shown by Fig. 7, the `PPI` kernel consumes about 99% of the total GPU time, while the `RandomGPU` kernel compraratively occupies a much smaller fraction. Finally, the data movement operations are not significant, which indicates that most of the GPU processing time is invested in the most time-consuming operation, i.e., the calculation of skewer projections and identification of maxima and minima projection values leading to endmember identification.

To conclude this section, we emphasize that our implementation of the unmixing chain has been fully optimized for the considered GPU platform. This statement is based on our utilization of NVidia CUDA's occupancy calculator,[18] a programmer tool that allows one to compute the multiprocessor occupancy of a GPU by a given CUDA kernel in terms of the number of active threads per multiprocessor, the number of active thread blocks per multiprocessor, and the total occupancy of each multiprocessor. The configuration adopted for our GPU implementation resulted in 100% occupancy according to this tool. This result was also confirmed by run-time experiments using NVidia CUDA's visual profiler tool[19].

E. Discussion

Through the detailed analysis of a full hyperspectral unmixing chain, we have illustrated different parallel systems and strategies to increase computational performance of hyperspectral imaging algorithms. Two of the considered techniques, i.e., commodity cluster-based parallel computing and heterogeneous parallel computing, seem particularly appropriate for efficient information extraction from very large hyperspectral data archives. In this regard, we have provided a discussion on the scalability of the unmixing chain on a NASA cluster and also studied the performance on a heterogeneous system made up of different workstations and communication links. Our study reveals that the combination of the (readily available) computational power offered by clusters and heterogeneous networks with the recent advances in sensor technology is ready to provide advanced exploration and mining capabilities for exploiting the sheer volume of Earth and planetary remotely sensed data which is already available in data repositories. These data, often never processed but simply stored in a database, can provide parameters and indicators useful to analyses related with other areas and/or analysis scenarios.

[17]http://developer.nvidia.com/object/cuda_3_1_downloads.html.

[18]http://news.developer.nvidia.com/2007/03/cuda_occupancy_.html.

[19]http://developer.nvidia.com/object/cuda_3_2_toolkit_rc.html.

To fully address the time-critical constraints introduced by many remote sensing applications, we have also developed an FPGA and a GPU-based implementation of the hyperspectral unmixing chain intended for on-board analysis (before the hyperspectral data is transmitted to Earth). A major goal is to overcome an existing limitation in many remote sensing and observatory systems: the bottleneck introduced by the bandwidth of the downlink connection from the observatory platform. Experimental results demonstrate that our hardware implementations make appropriate use of computing resources in the considered FPGA and GPU architectures, and further provide a response in (near) real-time which is believed to be acceptable in most remote sensing applications. The reconfigurability of FPGA systems on the one hand, and the low cost of GPU systems on the other, open many innovative perspectives from an application point of view, ranging from the appealing possibility of being able to adaptively select one out of a pool of available data processing algorithms (which could be applied on the fly aboard the airborne/satellite platform, or even from a control station on Earth), to the possibility of providing a response in applications with real-time constraints. Although the experimental results presented for the considered unmixing chain are encouraging, further work is still needed to arrive to optimal parallel design and implementations for other more sophisticated hyperspectral processing algorithms such as advanced supervised classifiers (e.g., support vector machines) which do not exhibit regular patterns of computation and communication [5].

Finally it is important to discuss the design effort needed for the creation of HPC solutions for each of the considered platforms. Compared to a C++ code developed for conventional CPUs, all solutions require additional design efforts. Specially since designers must learn different design paradigm and development environments, and also take into account several implementation low-level details. After comparing the design effort complexity of the options that we have implemented, we believe that the simplest implementation is the one developed for clusters, since the MPI calls can be integrated in the C++ code in a relatively straightforward manner. The adaptation of such codes for heterogeneous platforms involves the design of an additional module for ensuring load balance (WEA algorithm). In turn, the effort needed for designing the specialized hardware implementations is higher due to the closer relationship between the software and the hardware (i.e. the software designs needs to be tailored to the specific hardware architecture). With this in mind, probably the FPGA design is a little bit more complex than the GPU design due to two main reasons. On the one hand, in the FPGA implementation the platform needs to be designed whereas in the GPU implementation the platform just needs to be used. On the other hand, for large FPGA designs hardware debugging is a complex issue. However, we believe that, in both cases, the performance achievements are more significant than the increase in the design complexity.

VI. Conclusions

In this paper, we have reviewed the state-of-the-art in the application of HPC techniques and practices to hyperspectral remote sensing problems, and further explored different strategies to increase the performance of a hyperspectral unmixing chain on different parallel computing architectures. Techniques discussed include a commodity cluster-based implementation, a heterogeneity-aware parallel implementation developed for heterogeneous networks of workstations, an FPGA-based implementation, and a GPU-based implementation. Our study reveals that computational power offered by clusters and heterogeneous networks is ready to introduce substantial benefits from the viewpoint of exploiting large volumes of data already available in repositories, thus making a better use of hyperspectral data sets which have never been processed after being collected. These data exhibit the potential to provide relevant parameters that may be useful in other domains. To address the time-critical constraints introduced by several hyperspectral remote sensing applications, we have also developed FPGA and GPU-based implementations of the considered hyperspectral unmixing chain intended for on-board analysis.

VII. Future Directions

The presented work is part of a much larger strive to bring the benefits of high performance and distributed computing to the hyperspectral imaging community. Future work should include the development of techniques 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 many innovative perspectives. Radiation-tolerance and power consumption issues for these hardware devices should be explored in future developments. Further, additional hyperspectral imaging techniques should be considered for detailed analysis of parallelization strategies. For example, some more advanced endmember extraction algorithms, such as Winter's N-FINDR [53], are very popular and their parallel implementations will be investigated. In turn, the algorithms used in this paper to perform spectral unmixing are pixel-based and do not take into account the inter-pixel correlation. In this case, the parallelization framework is very simple and the data can be easily divided into a number of subsets whose number is equal to the number of available processors. An endmember extraction algorithm using both spectral and spatial information (e.g., [54]) presents more complex patterns of computation and communication and therefore bring new challenges for efficient parallel implementation. Other techniques of this kind are kernel-based methods, e.g., kernel-based anomaly detection using the RX algorithm [55] needs additional inter-processor communications in order to produce the inverse of the covariance matrix required for the overall computation. Future work should also comprise an investigation of SIMD versus MIMD strategies for parallel implementation of hyperspectral imaging algorithms.

References

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

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

[3] A. Plaza, "Special issue on architectures and techniques for real-time processing of remotely sensed images," *J. Real-Time Image Process.*, vol. 4, pp. 191–193, 2009.

[4] L. Zhou, M. Goldberg, C. Barnet, Z. Cheng, F. Chung, W. Wolf, T. King, X. Liu, H. Sun, and M. Divakarla, "Regression of surface spectral emissivity from hyperspectral instruments," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, pp. 328–333, 2008.

[5] A. Plaza, J. A. Benediktsson, J. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, J. Gualtieri, M. Marconcini, J. C. Tilton, and G. Trianni, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, pp. 110–122, 2009.

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

[7] J. Dorband, J. Palencia, and U. Ranawake, "Commodity computing clusters at goddard space flight center," *J. Space Commun.*, vol. 3, p. 1, 2003.

[8] R. Brightwell, L. Fisk, D. Greenberg, T. Hudson, M. Levenhagen, A. Maccabe, and R. Riesen, "Massively parallel computing using commodity components," *Parallel Comput.*, vol. 26, p. 243266, 2000.

[9] K. Itoh, "Massively parallel fourier-transform spectral imaging and hyperspectral image processing," *Opt. Laser Technol.*, vol. 25, p. 202, 1993.

[10] S. Kalluri, Z. Zhang, J. JaJa, S. Liang, and J. Townshend, "Characterizing land surface anisotropy from AVHRR data at a global scale using high performance computing," *Int. J. Remote Sens.*, vol. 22, pp. 2171–2191, 2001.

[11] T. El-Ghazawi, S. Kaewpijit, and J. L. Moigne, "Parallel and adaptive reduction of hyperspectral data to intrinsic dimensionality," *Cluster Comput.*, vol. 1, pp. 102–110, 2001.

[12] S. Tehranian, Y. Zhao, T. Harvey, A. Swaroop, and K. Mckenzie, "A robust framework for real-time distributed processing of satellite data," *J. Parallel Distrib. Comput.*, vol. 66, pp. 403–418, 2006.

[13] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *J. Parallel Distrib. Comput.*, vol. 66, no. 3, pp. 345–358, 2006.

[14] A. Plaza and C.-I Chang, "Clusters versus FPGA for parallel processing of hyperspectral imagery," *Int. J. High Perform. Comput. Applicat.*, vol. 22, no. 4, pp. 366–385, 2008.

[15] 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," *J. Supercomput.*, vol. 40, no. 1, pp. 81–107, 2007.

[16] A. Plaza, D. Valencia, and J. Plaza, "An experimental comparison of parallel algorithms for hyperspectral analysis using homogeneous and heterogeneous networks of workstations," *Parallel Comput.*, vol. 34, no. 2, pp. 92–114, 2008.

[17] A. Lastovetsky and J. Dongarra, *High-Performance Heterogeneous Computing*. New York: Wiley, 2009.

[18] 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," *Int. J. High Perform. Comput. Applicat.*, vol. 22, no. 4, pp. 386–407, 2008.

[19] W. Rivera, C. Carvajal, and W. Lugo, "Service oriented architecture grid based environment for hyperspectral imaging analysis," *Int. J. Inf. Technol.*, vol. 11, no. 4, pp. 104–111, 2005.

[20] J. Brazile, R. A. Neville, K. Staenz, D. Schlaepfer, L. Sun, and K. I. Itten, "Cluster versus grid for operation generation of ATCOR's MODTRAN-based look up table," *Parallel Comput.*, vol. 34, pp. 32–46, 2008.

[21] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proc. IEEE*, vol. 86, pp. 615–638, 1998.

[22] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, pp. 39–55, 2008.

[23] U. Thomas, D. Rosenbaum, F. Kurz, S. Suri, and P. Reinartz, "A new software/hardware architecture for real time image processing of wide area airborne camera images," *J. Real-Time Image Process.*, vol. 5, pp. 229–244, 2009.

[24] Q. Du and R. Nekovei, "Fast real-time onboard processing of hyperspectral imagery for detection and classification," *J. Real-Time Image Process.*, vol. 22, pp. 438–448, 2009.

[25] M. Hsueh and C.-I Chang, "Field programmable gate arrays (FPGA) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery," *Int. J. High Performance Comput. Applicat.*, vol. 22, pp. 408–423, 2008.

[26] E. El-Araby, T. El-Ghazawi, J. L. Moigne, and R. Irish, "Reconfigurable processing for satellite on-board automatic cloud cover assessment," *J. Real-Time Image Process.*, vol. 5, pp. 245–259, 2009.

[27] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado, "GPU for parallel on-board hyperspectral image processing," *Int. J. High Performance Comput. Applicat.*, vol. 22, no. 4, pp. 424–437, 2008.

[28] A. Paz and A. Plaza, "Clusters versus GPUs for parallel automatic target detection in remotely sensed hyperspectral images," *EURASIP J. Advances in Signal Process.*, vol. 915639, pp. 1–18, 2010.

[29] C. Gonzalez, J. Resano, D. Mozos, A. Plaza, and D. Valencia, "FPGA implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis," *EURASIP J. Advances in Signal Process.*, vol. 969806, pp. 1–13, 2010.

[30] D. A. Landgrebe, *Signal Theory Methods in Multispectral Remote Sensing*. New York: Wiley, 2003.

[31] A. S. Tanenbaum, *Structured Computer Organization*. Englewood Cliffs, NJ: Prentice-Hall, 2001.

[32] J. C. Tilton, W. T. Lawrence, and A. Plaza, "Utilizing hierarchical segmentation to generate water and snow masks to facilitate monitoring of change with remotely sensed image data," *GISci. Remote Sens.*, vol. 43, no. 1, pp. 39–66, Mar. 2006.

[33] J. Plaza, A. Plaza, D. Valencia, and A. Paz, "Massively parallel processing of hyperspectral images," in *Proc. SPIE*, 2009, vol. 7455, pp. 1–11.

[34] A. Plaza, J. Plaza, and A. Paz, "Parallel heterogeneous CBIR system for efficient hyperspectral image retrieval using spectral mixture analysis," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 9, pp. 1138–1159, 2010.

[35] F. Liu, F. Seinstra, and A. Plaza, "Parallel hyperspectral image processing on multi-cluster systems," *IEEE Geosci. Remote Sens. Lett.*, 2010, submitted for publication.

[36] Q. Du and J. E. Fowler, "Low-complexity principal component analysis for hyperspectral image compression," *Int. J. High Performance Comput. Applicat.*, vol. 22, pp. 273–286, 2009.

[37] M. Parente, J. L. Bishop, and J. F. Bell, "Spectral unmixing for mineral identification in pancam images of soils in Gusev crater, Mars," *Icarus*, vol. 203, pp. 421–436, 2009.

[38] Y. Tarabalka, T. V. Haavardsholm, I. Kasen, and T. Skauli, "Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and gpu processing," *J. Real-Time Image Process.*, vol. 4, pp. 1–14, 2009.

[39] G. Shaw and D. Manolakis, "Signal processing for hyperspectral image exploitation," *IEEE Signal Process. Mag.*, vol. 19, pp. 12–16, 2002.

[40] J. B. Adams, M. O. Smith, and P. E. Johnson, "Spectral mixture modeling: A new analysis of rock and soil types at the Viking Lander 1 site," *J. Geophys. Res.*, vol. 91, pp. 8098–8112, 1986.

[41] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 44–57, 2002.

[42] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 650–663, 2004.

[43] Q. Du, N. Raksuntorn, N. H. Younan, and R. L. King, "End-member extraction for hyperspectral image analysis," *Appl. Opt.*, vol. 47, pp. 77–84, 2008.

[44] D. Heinz and C.-I Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, pp. 529–545, 2000.

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

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

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

[48] Q. Du and C.-I Chang, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 608–619, 2004.

[49] C.-I Chang and D. Heinz, "Constrained subpixel target detection for remotely sensed imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 38, pp. 1144–1159, 2000.

[50] F. Seinstra and D. Koelma, "User transparency: A fully sequential programming model for efficient data parallel image processing," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 6, pp. 611–644, 2004.

[51] R. Elsasser, B. Monien, and R. Preis, "Diffusion schemes for load balancing on heterogeneous networks," *Theory of Computing Systems*, vol. 35, pp. 305–320, 2002.
[52] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator," *ACM Trans. Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.
[53] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data," in *Proc. SPIE Image Spectrometry V*, 2003, vol. 3753, pp. 266–277.
[54] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral end-member extraction by multidimensional morphological operations," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 9, pp. 2025–2041, 2002.
[55] I. Reed and X. Yu, "Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distrihution," *IEEE Trans. Acoustics, Speech Signal Process.*, vol. 38, pp. 1760–1770, 1990.

Antonio Plaza (M'05–SM'07) received the M.S. and Ph.D. degrees in computer engineering from the University of Extremadura, Caceres, Spain.

He was a Visiting Researcher with the Remote Sensing Signal and Image Processing Laboratory, University of Maryland Baltimore County, Baltimore, with the Applied Information Sciences Branch, Goddard Space Flight Center, Greenbelt, MD, and with the AVIRIS Data Facility, Jet Propulsion Laboratory, Pasadena, CA. Since 2000, he has been an Associate Professor with the Department of Technology of Computers and Communications, University of Extremadura, Caceres, Spain, where he is the Head of the Hyperspectral Computing Laboratory (HYPERCOMP). He is the Coordinator of the Hyperspectral Imaging Network (Hyper-I-Net), which is a European project designed to build an interdisciplinary research community focused on hyperspectral imaging activities. He has been a Proposal Reviewer with the European Commission, the European Space Agency, and the Spanish Government. He is the author or coauthor of more than 250 publications on remotely sensed hyperspectral imaging, including more than 40 Journal Citation Report papers, book chapters, and conference proceeding papers. His research interests include remotely sensed hyperspectral imaging, pattern recognition, signal and image processing, and efficient implementation of large-scale scientific problems on parallel and distributed computer architectures.

Dr. Plaza has co-edited a book on high-performance computing in remote sensing and guest-edited four special issues on remotely sensed hyperspectral imaging for different journals, including the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING (for which he currently serves as Associate Editor on hyperspectral image analysis and signal processing), the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING, the *International Journal of High Performance Computing Applications*, and the *Journal of Real-Time Image Processing*. He has served as a reviewer for more than 230 manuscripts submitted to more than 40 different journals. He was a recipient of the recognition of Best Reviewers of the IEEE *Geoscience and Remote Sensing Letters* in 2009.

Qian Du (S'98–M'00–SM'05) received the Ph.D. degree in electrical engineering from the University of Maryland Baltimore County in 2000.

She was with the Department of Electrical Engineering and Computer Science, Texas A&M University, Kingsville, from 2000 to 2004. She joined the Department of Electrical and Computer Engineering at Mississippi State University in Fall 2004, where she is currently an Associate Professor. Her research interests include remote sensing image analysis, pattern classification, data compression, and neural networks.

Dr. Du currently serves as Co-Chair for the Data Fusion Technical Committee of IEEE Geoscience and Remote Sensing Society. She also served as Guest Editor for the special issue on Spectral Unmixing of Remotely Sensed Data in IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, and Guest Editor for the special issue on High Performance Computing in Earth Observation and Remote Sensing in IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). Dr. Du is a member of SPIE, ASPRS, and ASEE.

Yang-Lang Chang (M'05–SM'08) received the B.S. degree in electrical engineering from Chung Yuan Christian University, Taiwan, in 1987, the M.S. degree in computer engineering from Syracuse University, New York, in 1993, and the Ph.D. degree in computer science and information engineering from the National Central University, Taiwan, in 2003.

He started his career with NDC IBM Taiwan as a Hardware Design Engineer before joining Alcatel as a Software Development Engineer. He presently is an Associate Professor in the Department of Electrical Engineering, National Taipei University of Technology. His research interests are in remote sensing, high performance computing and hyperspectral image analysis.

Dr. Chang is a member of SPIE, the Phi Tau Phi Scholastic Honor Society, the Chinese Society of Photogrammetry and Remote Sensing, and the Chinese Society of Image Processing and Pattern Recognition. He has been a Conference Program Committee member and Session Chair for several international conferences. He is a member of the Editorial Advisory Board of *Open Remote Sensing Journal*. Currently, he serves as a General Secretary of the Taipei Chapter of the IEEE Geoscience and Remote Sensing Society.

Roger L. King (M'73–SM'95) received the B.S. degree from West Virginia University, Morgantown, WV, in 1973, the M.S. degree from the University of Pittsburgh, Pittsburgh, PA, in 1978 in electrical engineering. He received the Ph.D. in engineering from the University of Wales, Cardiff, U.K., in 1988.

He began his career with Westinghouse Electric Corporation, and soon moved to the U.S. Bureau of Mines Pittsburgh Mining and Safety Research Center. Upon receiving the Ph.D. in 1988, he accepted a position in the Department of Electrical and Computer Engineering at Mississippi State University where he now holds the position of Giles Distinguished Professor. At Mississippi State University, he presently serves as the Director of the Center for Advanced Vehicular Systems.

Dr. King has received numerous awards for his leadership in research including the Department of Interior's Meritorious Service Medal and serves as an Honorary Professor at the Cardiff University in the United Kingdom. He served as the Co-Technical Chair for IGARSS '09 in Cape Town, South Africa. He is currently serving as an Associate Editor for the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. He has published over 250 journal and conference publications and holds four patents. Over the last 30 years, he has served in a variety of leadership roles with the IEEE Industry Applications Society, Power Engineering Society, and Geosciences and Remote Sensing Society.