Clusters Versus FPGA for Parallel Processing of Hyperspectral Imagery

Antonio Plaza and Chein-I Chang

International Journal of High Performance Computing Applications 2008; 22; 366
DOI: 10.1177/1094342007088376

CLUSTERS VERSUS FPGA FOR PARALLEL PROCESSING OF HYPERSPECTRAL IMAGERY

Antonio Plaza[1]
Chein-I Chang[2]

Abstract

Hyperspectral imaging is a new technique in remote sensing that generates images with hundreds of spectral bands, at different wavelength channels, for the same area on the surface of the Earth. Although in recent years several efforts have been directed toward the incorporation of parallel and distributed computing in hyperspectral image analysis, there are no standardized architectures for this purpose in remote sensing missions. To address this issue, this paper develops two highly innovative implementations of a standard hyperspectral data processing chain utilized, among others, in commercial software tools such as Kodak's Research Systems ENVI software package (one of the most popular tools currently available for processing remotely sensed data). It should be noted that the full hyperspectral processing chain has never been implemented in parallel in the past. Analytical and experimental results are presented in the context of a real application, using hyperspectral data collected by NASA's Jet Propulsion Laboratory over the World Trade Center area in New York City, shortly after the terrorist attacks of September 11th 2001. The parallel implementations are tested in two different platforms, including Thunderhead, a massively parallel Beowulf cluster at NASA's Goddard Space Flight Center, and a Xilinx Virtex-II field programmable gate array (FPGA) device. Combined, these platforms deliver an excellent snapshot of the state-of-the-art in those areas, and offer a thoughtful perspective on the potential and emerging challenges of incorporating parallel processing systems into realistic hyperspectral imaging problems.

Key words: hyperspectral imaging, commodity clusters, reconfigurable computing, hyperspectral data processing chain, pixel purity index

1 Introduction

The development of computationally efficient techniques for transforming the massive amount of remote sensing data into scientific understanding is critical for Earth observation and planetary exploration (Schowengerdt 2007). In particular, the wealth of spatial and spectral information provided by latest generation remote sensing has opened ground-breaking perspectives in many applications, including environmental modeling and assessment (Patrino 2005) or hazard prevention and response (Clark et al. 2006) including fire detection and 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, and so forth. (Chang 2007). Many of the above-mentioned applications require analysis algorithms able to provide a response in real-time (Winter et al. 2002), which is quite an ambitious goal because the price paid for the rich information available from latest generation sensors is the enormous amounts of data that they generate (Aloisio and Cafaro 2003; Chen, Fujishiro, and Nakajima 2003; Hawick, Coddington, and James 2003).

Hyperspectral imaging (Chang 2003) is a new technique in remote sensing, in which an image spectrometer collects hundreds or even thousands of measurements (at multiple wavelength channels) for the same area on the surface of the Earth. The images provided by such sensors are often called *image cubes* to denote the extremely high dimensionality of the data. The NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS; Green et al. 1998) is the most advanced airborne-based Earth observation hyperspectral instrument currently available, and routinely records 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 km wide and several kilometers long using 224 spectral bands (see Figure 1). The resulting image cube 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. The extremely high computational requirements already introduced by hyperspectral imaging applications

[1] DEPARTMENT OF TECHNOLOGY OF COMPUTERS AND COMMUNICATIONS, TECHNICAL SCHOOL OF CÁCERES, UNIVERSITY OF EXTREMADURA, AVDA. DE LA UNIVERSIDAD S/N, E-10071 CÁCERES, SPAIN, (APLAZA@UNEX.ES)

[2] REMOTE SENSING SIGNAL AND IMAGE PROCESSING LABORATORY (RSSIPL), DEPARTMENT OF COMPUTER SCIENCE AND ELECTRICAL ENGINEERING, UNIVERSITY OF MARYLAND BALTIMORE COUNTY (UMBC), 1000 HILLTOP CIRCLE, BALTIMORE MD-20250

The International Journal of High Performance Computing Applications, Volume 22, No. 4, Winter 2008, pp. 366–385
DOI: 10.1177/1094342007088376

Figures 1, 7, 8 appear in color online: http://hpc.sagepub.com

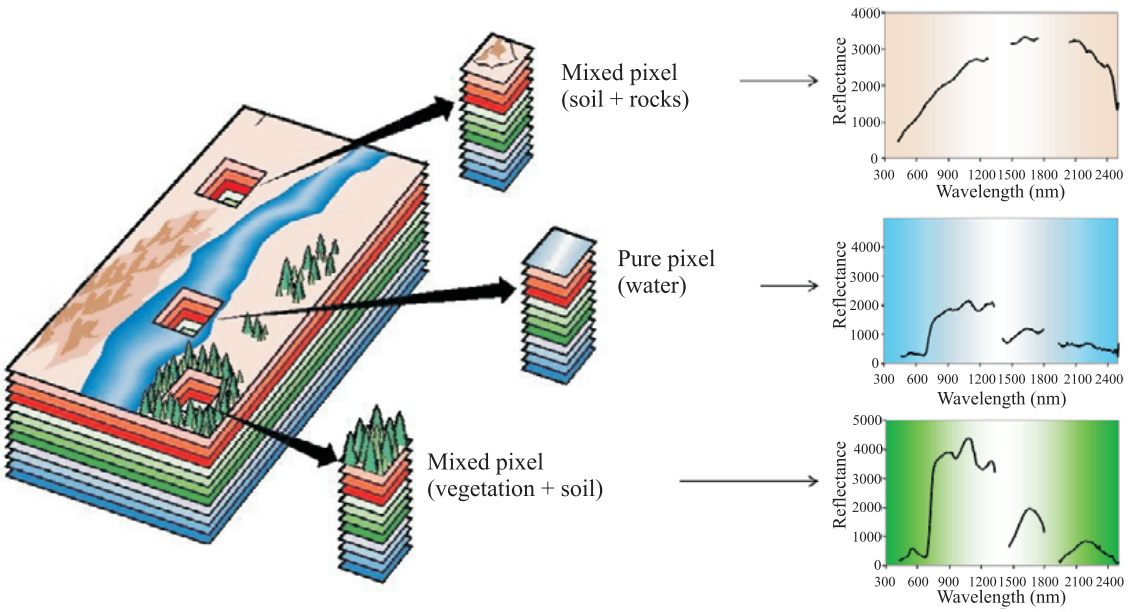

Fig. 1 The concept of hyperspectral imaging using NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) system.

(and the fact that these systems will continue to increase their spatial and spectral resolutions in the near future) make them an excellent case study to illustrate the need for high performance computing (HPC) systems in remote sensing applications.

Specifically, the utilization of HPC systems in hyperspectral imaging applications has become increasingly widespread in recent years. From a computational perspective, hyperspectral image processing algorithms exhibit inherent parallelism at multiple levels: across pixel vectors (coarse grained pixel-level parallelism), across spectral information (fine grained spectral-level parallelism) and even across tasks (task-level parallelism). Because data accesses in these algorithms are regular and predictable, they can be implemented using vector programming techniques, thus taking advantage of the SIMD (single-instruction, multiple-data) capabilities of vector processors. The introduction of MMX technology in 1997 and other architecture extensions in subsequent years, especially SSE and SSE2 in Intel processors (Skoglund and Felsberg 2005), and AltiVec in PowerPC (Diefendorff et al. 2000), allowed for the development of improved hyperspectral image processing algorithms (Wang, Rucker, and Fowler 2004).

Another approach to tackling the extremely high data volumes collected by Earth observation instruments in a cost-effective way was to utilize commodity Beowulf clusters (Sterling 2002). The idea of using commercial off-the-shelf (COTS) computer equipment clustered together to work as a computational "team" was originally developed to create a parallel computing system from commodity components to satisfy specific requirements for the Earth and space sciences community (Dorband, Palencia, and Ranawake 2003). This strategy has already offered access to greatly increased computational power, but at a low cost (commensurate with falling commercial PC costs) in a number of remote sensing applications (Dhodhi et al. 1999; Kalluri et al. 2001; Le Moigne, Campbell, and Cromp 2002; Wang et al. 2002; Achalakul and Taylor 2003; Tilton 2005; Tehranian et al. 2006). In theory, the combination of commercial forces driving down cost and positive hardware trends (e.g. CPU peak power doubling every 18–24 months, storage capacity doubling every 12–18 months and networking bandwidth doubling every 9–12 months) offers supercomputing performance that can now be applied to a wide range of hyperspectral problems.

Although hyperspectral image processing algorithms map nicely to parallel systems made up of commodity CPUs (Plaza et al. 2006), these systems are generally expensive and difficult to adapt to onboard remote sensing data processing scenarios. In onboard processing, low-weight and low-power integrated components are essential to reduce mission payload and obtain analysis results in real-time. In pushbroom instruments such as AVIRIS, a main goal of onboard real-time processing is

to be able to use a coprocessor system to analyze a full hyperspectral *cube frame* (consisting of 614 × 512 lines and 224 spectral bands, with about 140 MB in size because of the integer format used to store each radiance value) before the next frame is collected. Since the cross-track scan line in AVIRIS is quite fast (8.3 ms), this introduces the need to process a full cube frame in no more than 5 s to fully achieve real-time performance.

An exciting new development in the field of specialized commodity computing is the emergence of hardware devices such as field programmable gate arrays (FPGAs), which can bridge the gap toward onboard and real-time analysis of remote sensing data (Fry and Hauck 2002; El-Araby et al. 2004; Valencia and Plaza 2006; Vladimirova and Wu 2006). FPGAs are now fully reconfigurable, which allows one to adaptively select a data processing algorithm (out of a pool of available ones) to be applied onboard the sensor from a control station on Earth. The ever-growing computational demands of remote sensing applications can fully benefit from compact, reconfigurable hardware components and take advantage of the small size and relatively low cost of these units as compared to clusters or networks of computers.

In this paper, our main goal is to discuss the role of several HPC-based architectures in hyperspectral imaging, with the ultimate goal of analyzing the possibility of obtaining (near) real-time processing results in time-critical applications. To achieve this goal, we develop and compare several new parallel versions of a well-known hyperspectral data processing chain adopted in standard remote sensing software. In particular, our goal is to provide an application-oriented case study in which the performance of different architectures and techniques for efficient processing of hyperspectral image data is discussed in the context of a real problem in which an efficient utilization of hyperspectral data clearly depends on high computing performance of algorithm analysis. The paper is organized as follows. Section 2 describes a hyperspectral data processing chain that will serve as our case study throughout the paper. This framework has been widely used to analyze hyperspectral images and is available in Kodak's Research Systems ENVI, a very popular commercial tool for remote sensing image processing. Section 3 provides several HPC-based implementations of the data processing chain, including a cluster-based parallel version and an FPGA-based implementation. Section 4 provides an experimental comparison of the proposed parallel implementations using several HPC architectures. Specifically, we use Thunderhead, a massively parallel Beowulf cluster at NASA's Goddard Space Flight Center, and a Xilinx Virtex-II FPGA device. The results obtained in these two platforms are thoroughly discussed in the context of the considered application (analysis of hyperspectral data collected after the World Trade Center terrorist attacks in September 2001). Finally, Section 5 concludes with some remarks and plausible future research lines.

2 Hyperspectral Data Processing Chain

This section describes a commonly accepted hyperspectral data processing chain that will be used as a case study for the development of parallel algorithms (Chang 2003). It consists of the following stages. First, a set of pure spectral signatures (often called *endmembers* in hyperspectral analysis) are extracted from the input data set. The goal of using endmembers is to deal with the problem of mixed pixels, which arise when the spatial resolution of the sensor is not high enough to separate different materials. For instance, the pixel vector labeled as "vegetation + soil" in Figure 1 actually comprises a mixture of vegetation and soil. As a result, the spectral signature associated to each pixel vector measures the response of multiple underlying materials at the imaged site. To deal with this problem, spectral unmixing (Keshava and Mustard 2002) has been used to decompose the measured spectrum of a mixed pixel into a linear combination of endmembers weighted by a set of abundance fractions that indicate the proportion of each endmember (pure) pixel present in the mixture. A hyperspectral image is often a combination of the two situations, where a few sites in a scene are pure materials, but many others are mixtures of materials.

One of the most successful algorithms for automatic endmember extraction in the literature has been the PPI algorithm, originally developed by Boardman, Kruse, and Green (1993), and soon incorporated into Kodak's Research Systems ENVI software package. The algorithm proceeds by generating a large number of random, N-dimensional unit vectors called "skewers" through the dataset. Every data point is projected onto each skewer, and the data points that correspond to extrema in the direction of a skewer are identified and placed on a list (see Figure 2). As more skewers are generated, the list grows, and the number of times a given pixel is placed on this list is also tallied. The pixels with the highest tallies are considered the final endmembers.

The inputs to the algorithm are a hyperspectral data cube $\mathbf{F}$ with N dimensions; a maximum number of endmembers to be extracted, E; the number of random skewers to be generated during the process, K; a cut-off threshold value, t_v, used to select as final endmembers only those pixels that have been selected as extreme pixels at least t_v times throughout the PPI process; and a threshold angle, t_a, used to discard redundant endmembers during the process. The output of the algorithm is a set of E final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$. The algorithm can be summarized by the following steps:

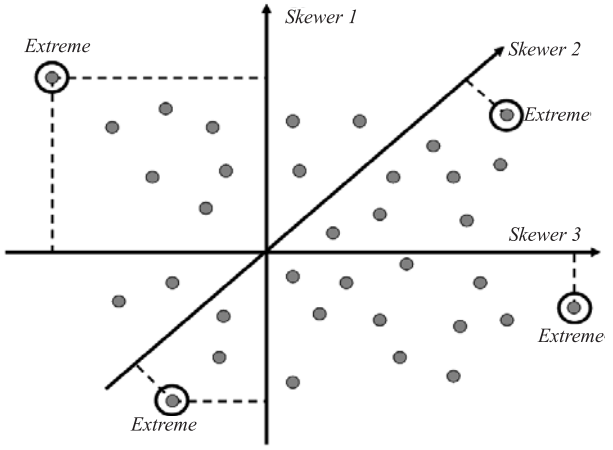

Fig. 2 Toy example illustrating the performance of the endmember extraction algorithm in a 2-dimensional space.

1. *Skewer generation*. Produce a set of K randomly generated unit vectors $\{\mathbf{skewer}_j\}_{j=1}^{K}$.
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 to find sample vectors at its extreme (maximum and minimum) projections, thus forming an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}(\mathbf{skewer}_j)$. The dot products are calculated as follows:

$$\left|\mathbf{f}_i \cdot \mathbf{skewer}_j\right| = \frac{\left|\mathbf{f}_i \cdot \mathbf{skewer}_j\right|}{\left\|\mathbf{f}_i\right\| \cdot \left\|\mathbf{skewer}_j\right\|} \tag{1}$$

   Despite the fact that a different $\mathbf{skewer}_j$ would generate a different extrema set $S_{extrema}(\mathbf{skewer}_j)$, it is very likely that some sample vectors may appear in more than one extrema set. In order to deal with this situation, we define an indicator function of a set S, denoted by $I_S(\mathbf{x})$, to denote membership of an element $\mathbf{x}$ to that particular set as follows:

$$I_S(\mathbf{f}_i) = \left\{ \begin{array}{l} 1 \text{ if } \mathbf{x} \in S \\ 0 \text{ if } \mathbf{x} \notin S \end{array} \right\} \tag{2}$$

3. *Calculation of PPI scores*. Using the indicator function above, we calculate the PPI score associated to the sample pixel vector $\mathbf{f}_i$ (i.e. the number of times that given pixel has been selected as extreme in step 2) using the following equation:

$$N_{PPI}(\mathbf{f}_i) = \sum_{j=1}^{K} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i) \tag{3}$$

4. *Endmember selection*. Find the pixels with value of $N_{PPI}(\mathbf{f}_i)$ above t_v, and form a unique set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the spectral angle mapper (SAM; Chang 2003) for all possible vector pairs and discarding those pixels which result in an angle value below t_a. It should be noted that the SAM between a pixel vector $\mathbf{f}_i$ and a different pixel vector $\mathbf{f}_j$ is defined as follows:

$$\mathrm{SAM}(\mathbf{f}_i, \mathbf{f}_j) = \cos^{-1} \frac{\mathbf{f}_i \cdot \mathbf{f}_j}{\left\|\mathbf{f}_i\right\| \cdot \left\|\mathbf{f}_j\right\|} \tag{4}$$

   It should be noted that, although other spectral similarity metrics have been used in hyperspectral imaging research (Chang 2003), the SAM is widely regarded as a standard spectral similarity metric in remote sensing operations, mainly because it is invariant under the multiplication of the input vectors by constants and, consequently, is invariant to unknown multiplicative scalings that may arise due to differences in illumination and sensor observation angle.
5. *Spectral unmixing*. Although this step is not part of the PPI algorithm itself, we have decided to include it as part of our algorithmic description of the full hyperspectral data processing chain available in Research Systems ENVI to account for the need of interpreting mixed pixels as combinations of pure pixels in hyperspectral image processing applications. This step is accomplished as follows. For each sample pixel vector $\mathbf{f}_i$ in $\mathbf{F}$, a set of abundance fractions specified by $\{a_{i1}, a_{i2}, \cdots, a_{iE}\}$ is obtained using the set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$, so that $\mathbf{f}_i$ can be expressed as a linear combination of endmembers as follows:

$$\mathbf{f}_i = \mathbf{e}_1 \cdot a_{i1} + \mathbf{e}_2 \cdot a_{i2} + \cdots + \mathbf{e}_E \cdot a_{iE} \tag{5}$$

It should be noted that, in the expression above, abundance sum-to-one and non-negativity constraints are imposed, i.e. $\sum_{e=1}^{E} a_{ie} = 1$ and $a_{ie} \geq 0$ for all $i = \{1 \cdots T\}$, where T is the total number of pixels in the image $\mathbf{F}$, and for all $e = \{1 \cdots E\}$, where E is the total number of endmembers extracted by the PPI.

From the algorithmic description above, it is clear that the PPI is not an iterative algorithm (Chang and Plaza 2006). In order to set parameter values for the PPI, the authors recommend using as many random skewers as possible in order to obtain optimal results. As a result, the

PPI can only guarantee to produce optimal results asymptotically and its computational complexity is very high. According to our experiments using standard AVIRIS hyperspectral data sets (typically, 614 × 512 pixels and 224 spectral bands per frame), the PPI generally requires a very high number of skewers (in the order of $K = 10^4$ or $K = 10^5$) to produce an accurate final set of endmembers (Plaza et al. 2004), and results in processing times above 1 hr when the algorithm is run on a latest-generation desktop PC. Such a response time is unacceptable in time-critical remote sensing applications. In the following section, we provide an overview of HPC paradigms applied to speed up computational performance of the PPI using different kinds of parallel and distributed computing architectures.

3 Parallel Implementations

This section first develops a parallel implementation of the hyperspectral data processing chain presented in the previous section which has been specifically developed to be run on massively parallel, homogeneous clusters of Beowulf type. Then, an FPGA-implementation aimed at onboard hyperspectral data processing is provided.

3.1 Multiprocessor Implementation

In this subsection, we describe a master–slave parallel version of the proposed hyperspectral data processing chain. To reduce code redundancy and enhance reusability, our goal was to reuse much of the code for the sequential algorithm in the parallel implementation. For that purpose, we adopted a spatial-domain decomposition approach (Seinstra, Koelma, and Geusebroek 2002; Veeravalli and Ranganath 2003), that subdivides the image cube into multiple blocks made up of entire pixel vectors, and assigns one or more blocks to each processing element (see Figure 3).

It should be noted that the PPI algorithm is mainly based on projecting pixel vectors which are always treated as a whole. This is a result of the convex geometry process implemented by the PPI, which is based on the spectral "purity" or "convexity" of the entire spectral signature associated to each pixel. Therefore, a spectral-domain partitioning scheme (which subdivides the whole multiband data into blocks made up of contiguous spectral bands or sub-volumes, and assigns one or more sub-volumes to each processing element) is not appropriate in our application (Plaza et al. 2006). This is because the latter approach breaks the spectral identity of the data as each pixel vector is split amongst several processing elements. A further reason that justifies the above decision is that, in spectral-domain partitioning, the calculations made for each hyperspectral pixel need to originate from several processing elements, and thus require intensive inter-processor communication. Therefore, in our proposed implementation, a master–worker spatial domain-based decomposition paradigm is adopted, where the master processor sends

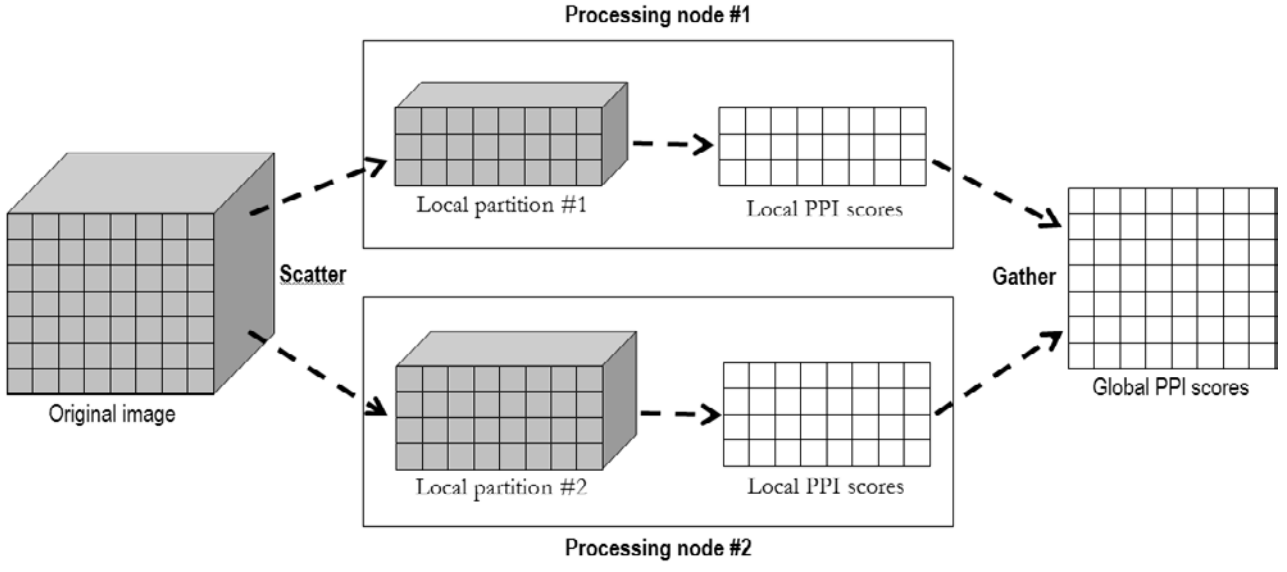

Fig. 3 Domain decomposition adopted in the parallel implementation of the endmember extraction part of the considered hyperspectral data processing chain.

partial data to the workers and coordinates their actions. Then, the master gathers the partial results provided by the workers and produces a final result.

As was the case with the serial version, the inputs to our cluster-based implementation are a hyperspectral data cube **F** with N dimensions; a maximum number of endmembers to be extracted, E; the number of random skewers to be generated during the process, K; a cut-off threshold value, t_v; and a threshold angle, t_a. The output of the algorithm is a set of E endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$. The parallel algorithm is given by the following steps:

1. *Data partitioning*. Produce a set of L spatial-domain homogeneous partitions of **F** and scatter all partitions by indicating all partial data structure elements which are to be accessed and sent to each of the workers.
2. *Skewer generation*. Generate K random unit vectors $\{\mathbf{skewer}_j\}_{j=1}^{K}$ in parallel, and broadcast the entire set of skewers to all the workers.
3. *Extreme projections*. For each $\mathbf{skewer}_j$, project all the sample pixel vectors at each local partition l onto $\mathbf{skewer}_j$ to find sample vectors at its extreme projections, and form an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}^{(l)}(\mathbf{skewer}_j)$. Now calculate the number of times each pixel vector $\mathbf{f}_i^{(l)}$ in the local partition is selected as extreme using the following expression:

$$N_{PPI}^{(l)}(\mathbf{f}_i^{(l)}) = \sum_{j=1}^{K} I_{S_{extrema}^{(l)}(\mathbf{skewer}_j)}(\mathbf{f}_i^{(l)}) \tag{6}$$

4. *Candidate selection*. Select those pixel vectors with $N_{PPI}^{(l)}(\mathbf{f}_i^{(l)}) > t_v$ and send them to the master node.
5. *Endmember selection*. The master gathers all the individual endmember sets provided by the workers and forms a unique set $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the SAM for all possible pixel vector pairs in parallel and discarding those pixels which result in angle values below t_a.
6. *Spectral unmixing*. The master broadcasts the set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ to all workers. Each worker then obtains a set of abundances $\{a_{i1}^{(l)}, a_{i2}^{(l)}, \cdots, a_{iE}^{(l)}\}$ for each pixel vector $\mathbf{f}_i^{(l)}$ in its local partition l, using the set $\{\mathbf{e}_e\}_{e=1}^{E}$ so that the following expression is satisfied taking into account the abundance sum-to-one and abundance non-negativity constraints:

$$\mathbf{f}_i^{(l)} = \mathbf{e}_1 \cdot a_{i1}^{(l)} + \mathbf{e}_2 \cdot a_{i2}^{(l)} + \cdots + \mathbf{e}_E \cdot a_{iE}^{(l)} \tag{7}$$

It should be noted that the proposed parallel algorithm has been implemented in the C++ programming language, using calls to *message passing interface* (MPI; Gropp et al. 1999). We emphasize that, in order to implement step one of the parallel algorithm, we resorted to MPI *derived datatypes* to directly scatter hyperspectral data structures, which may be stored non-contiguously in memory, in a single communication step. As a result, we avoid creating all partial data structures on the master node (thus making a better use of memory resources and compute power).

3.2 FPGA Implementation

In this subsection, we describe a hardware-based parallel strategy for implementation of the hyperspectral data processing chain which is aimed at enhancing replicability and reusability of slices in FPGA devices through the utilization of systolic array design (Valero-Garcia et al. 1992). One of the main advantages of systolic array-based implementations is that they are able to provide a systematic procedure for system design that allows for the derivation of a well defined processing element-based structure and an interconnection pattern which can then be easily ported to real hardware configurations (Zhang and Pal 2002). Using this procedure we can also calculate the data dependencies prior to the design in a very straightforward manner. Before describing our implementation, we emphasize that our proposed design intends to maximize computational power of the hardware and minimize the cost of communications. These goals are particularly relevant in our specific application, where hundreds of data values will be handled for each intermediate result, a fact that may introduce problems associated with limited resource availability and inefficiencies in hardware replication and reusability.

The rationale behind our systolic array-based parallelization can be summarized as follows. It has been shown in previous sections that the PPI algorithm consists of computing a very large number of dot-products, and all these dot-products can be performed simultaneously. As a result, a possible way of parallelization is to have a hardware system able to compute K dot-products in the same time against the same pixel $\mathbf{f}_i$, where K is the number of skewers. Supposing such a system, the PPI can be simply written as described in Algorithm 1.

The **par** loop in Algorithm 1 expresses that K dot products are first performed in parallel, then K Min and Max operations are also computed in parallel. Now, if we suppose that we cannot simultaneously compute K dot products but only a fraction K/P, where P is the number of available processing units in the underlying parallel platform, then the *extreme projections* step can be split into P passes, each performing $T \times K/P$ dot products, as indicated in Algorithm 2. From an architectural point of view, each processor receives successively the T pixels, computes T dot-products, and keeps in memory the two

```
for (n = 0; n < N; n++) { //N denotes the number of bands
  par (k = 0; k < K; k++) { //K denotes the number of skewers
    dp[k]=dot_product(pixels[n],skewers[k]);
    if (dp[k] < Min[k]) { Min[k]=dp[k]; Reg_Min[k]=n; }
    if (dp[k] > Max[k]) { Max[k]=dp[k]; Reg_Max[k]=n; }
  }
}
```

Algorithm 1. Parallel implementation of *extreme projections* step.

```
for (p = 0; p < P; p++) { //P is the number of processors
 x = p × (K/P);
 for (n = 0; n < N; n++) { //N denotes the number of bands
  par(k = 0; k < K/P; k++) { //K denotes the number of skewers
   dp[x + k]=dot_product(pixels[n],skewers[x + k]);
   if (dp[x + k] < Min[x + k]) { Min[x + k]=dp[x + k]; Reg_Min[x + k]=n; }
   if (dp[x + k] > Max[x + k]) { Max[x + k]=dp[x + k]; Reg_Max[x + k]=n;
}
  }
 }
}
```

Algorithm 2. Parallel implementation of *extreme projections* step (rewritten to be split into *P* algorithm iterations).

producing the Min and the Max dot products. In this scheme, each processor holds a different skewer which must be input before each new pass.

Figure 4 describes the systolic array design adopted for the proposed FPGA implementation. Here, local results remain static at each processing element, while pixel vectors are input to the systolic array from top to bottom and skewer vectors are fed to the systolic array from left to right. In Figure 4, asterisks represent delays while $\mathbf{skewer}_j^{(n)}$ denotes the value of the *n*th band of the *j*th skewer, with $j \in \{1, \cdots, K\}$ and $n \in \{1, \cdots, N\}$, being *N* the number of bands of the input hyperspectral scene. Similarly, $\mathbf{f}_j^{(n)}$ denotes the reflectance value of the *n*th band of the *i*th pixel, with $i \in \{1, \cdots, T\}$, being *T* is the total number of pixels in the input image. The processing nodes labeled as *dot* in Figure 4 perform the individual products for the skewer projections. On the other hand, the nodes labeled as *max* and *min* respectively compute the maxima and minima projections after the dot product calculations have been completed. In fact, the *max* and *min* nodes can be respectively seen as part of a 1-D systolic array which avoids broadcasting the pixel while simplifying the collection of the results.

The operational functionality of each *dot* processing node in Figure 4 is depicted in Figure 5. Each processing node accumulates the positive or negative values of the pixel input according to the skewer input. For instance, if the *i*th component of the skewer is 0, then the *i*th component of the pixel vector is summed up to the accumulator (AC). If it equals 1, it is subtracted. This unit is composed of a single 16-bit Add/Sub module. This module computes the dot product by summing up positive or negative pixel values. The skewer is stored in a 1-bit, *N*-word memory, where *N* is the number of spectral channels. The initialization mechanism is not represented. It should be noted that Figure 5 also illustrates the performance of the *min* and *max* processing nodes in Figure 4 (their performance is similar and hence they are represented in the figure as a single unit called Min/Max). This unit performs bit-serially a comparison between a Min/Max value. If the value of the dot-product exceeds the corresponding Min/Max value, then the current dot-product value is substituted and the number of the pixel which has caused this change is memorized.

Basically, a systolic cycle in the architecture described in Figure 4 consists of computing a single dot-product between a pixel and a skewer, and to memorize the index of the pixel if the dot-product is higher or smaller than a previously computed max/min value. Remember that a pixel is a vector of *N* spectral values, just like a

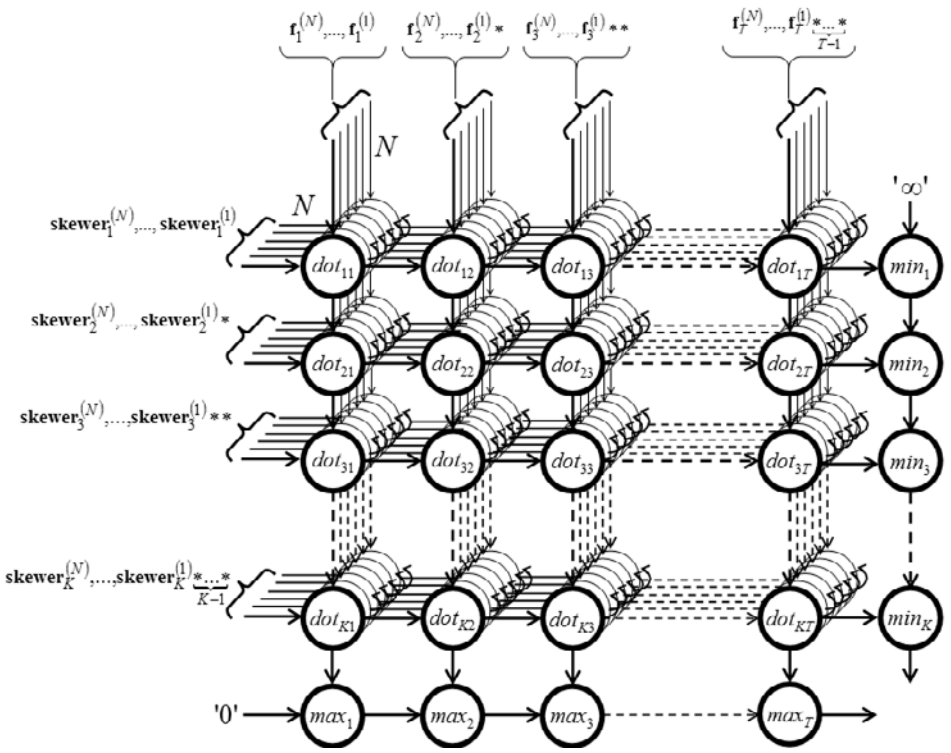

Fig. 4 Systolic array design for the proposed FPGA implementation of the hyperspectral data processing chain.

skewer. A dot-product calculation (dp) betwen a pixel $\mathbf{f}_i$ and a $\mathbf{skewer}_j$ can be simply obtained by using the expression $\sum_{k=1}^{N} \mathbf{f}_i^{(k)} \times \mathbf{skewer}_i^{(k)}$. Therefore, a full vector dot product calculation requires N multiplications and $N - 1$ additions, where N is the number of spectral bands. It has been shown in previous work that the skewer values can be limited to a very small set of integers when N is large, as in the case of hyperspectral images. A particular and interesting set is $\{1, -1\}$ since it avoids the multiplication (Lavernier et al. 1999). The dot product is thus reduced to an accumulation of positive and negative values (the self-connections in the dot nodes of Figure 4 represent the accumulation of intermediate results in those nodes). With the above assumptions in mind, the *dot* nodes only need to accumulate the positive or negative values of the pixel input according to the skewer input. These units are thus only composed of a single 16-bit addition/subtraction operator. If we suppose that an addition or a subtraction is executed every clock cycle, then the calculation of a full dot product requires N clock cycles. During the first systolic cycle, dot_{11} starts processing the first band of the first pixel vector, $\mathbf{f}_1$. During the second systolic cycle, the node dot_{12} starts processing the first band of pixel $\mathbf{f}_2$, while the node dot_{11} processes the second band of pixel $\mathbf{f}_1$, and so on.

The main advantage of the systolic array described in Figures 4 and 5 is its scalability. Depending of the resources available on the reconfigurable board, the number of processors can be adjusted without modifying the control of the array. In other words, although in Figure 4 we represent an ideal systolic array in which T pixels can be processed, this is not the usual situation, and the number of pixels usually has to be divided by P, the number of available processors. In this scenario, after T/P systolic cycles, all the nodes are working. When all the pixels have been flushed through the systolic array, T/P additional systolic cycles are required to collect the results for the considered set of P pixels and a new set of P different pixels would be flushed until processing all T pixels in the original image. In summary, the principle of our parallelization framework is that K/P processors perform successively N dot products, as shown in Algorithm 2. The pixels are broadcast to all the processors and the computation is pipelined (systolized) to provide a highly scalable system.

Finally, to obtain the vector of endmember abundances $\{a_{i1}, a_{i2}, \cdots, a_{iE}\}$ for each pixel $\mathbf{f}_i$, we multiply

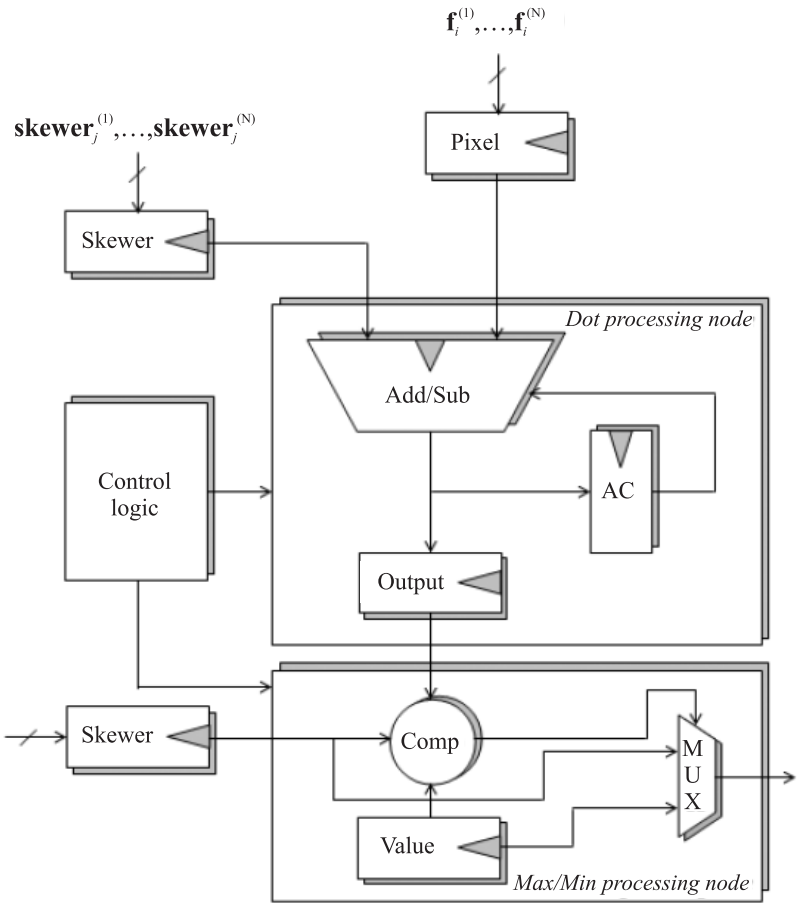

Fig. 5 Architecture of each dot and max/min processing nodes in the proposed systolic array design.

each $\mathbf{f}_i$ by $(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T$, where $\mathbf{M} = \{\mathbf{e}_e\}_{e=1}^{E}$ and the superscript "*T*" denotes the matrix transpose operation. As recently described (Dou et al. 2005), this operation can be done using a so-called parallel block algorithm, which has been adopted in this work to carry out the final spectral unmixing step added to our description of the PPI algorithm using part of the systolic array design outlined above.

Based on the design described above, we have developed a high-level implementation of PPI using Handel-C,[1] a design and prototyping language that allows using a pseudo-C programming style. The final decision on implementing our design using Handel-C instead of other well-known hardware description languages such as VHDL or Verilog was taken because a high-level language may allow users to generate hardware versions of available hyperspectral analysis algorithms in a relatively short time. For illustrative purposes, the source code in Handel-C corresponding to the *extreme projections* step of our FPGA implementation of the PPI algorithm is shown in Algorithm 3. The skewer initialization and spectral unmixing-related portions of the code are not shown for simplicity. The code assumes that the number of endmembers to be found by the PPI algorithm is known in advance. The high-level implementation in Algorithm 3 was compiled and transformed to an EDIF specification automatically by using the DK3.1 software package. With this specification, and using other tools such as Xilinx ISE[2] to simplify the final steps of the hardware implementation, we also incorporated hardware-specific limitations and constraints to the mapping process into a Virtex-II FPGA. These tools allowed us to evaluate the total amount of resources needed by the whole implementation, along with sub-totals related to different functional units available in the FPGA that will be discussed in the following section, along with our results

```
void main(void) {
  unsigned int 16 max[E]; //E is the number of endmembers
  unsigned int 16 end[E];
  unsigned int 16 i;
  unsigned int 10000 k; //k denotes the number of skewers
  unsigned int 224 N; //N denotes the number of bands
  par (i = 0; i < E; i++) max[i] = 0;
  par (k = 0; k < E; k++) {
    par (k = 0; k < E; k++) {
      par (j = 0; j < N; j++) {
        Proc_Element[i][k](pixels[i][j],skewers[k][j],0@i,0@k);
      }
    }
  }
  for (i = 0; i < E; i++) {
    max[i]=Proc_Element[i][k](0@max[i], 0, 0@i, 0@k);
  }
  phase_1_finished=1
  while (!phase_2) { //Waiting to enter phase 2 }
  for (i = 0; i < E; i++) end[i]=0;
  for (i = 0; i < E; i++) {
    par (k = 0; k < E; k++) {
      par (j = 0; j < N; j++) {
        end[i]=end[i]&&Proc_Element[i][k](pixels[i][j],skewers[k][j],0,0);
      }
    }
  }
  phase_2_finished=1
  global_finished=0
  for (i = 0; i < E; i++)   global_finished=global_ finished&&end[i];
```

Algorithm 3. Source code of the Handel-C (high level) FPGA implementation of the Pixel Purity Index algorithm.

in terms of algorithm performance and execution time for the two parallel implementations of PPI developed in this section.

4 Experimental Results

This section provides an assessment of the effectiveness of the parallel versions of the hyperspectral data processing described throughout this paper. Before describing our study on performance analysis, we first describe the parallel computing architectures used in this work. These include Thunderhead, a massively parallel Beowulf cluster at NASA's Goddard Space Flight Center, and a Xilinx Virtex-II XC2V6000-6 FPGA. Next, we describe the hyperspectral data sets used for evaluation purposes. A detailed survey on algorithm performance in a real application is then provided, along with a discussion of the advantages and disadvantages of each particular approach. The section concludes with a discussion of the results obtained for the data processing chain implemented using different HPC architectures.

4.1 Parallel Computing Architectures

This subsection provides an overview of the HPC platforms used in this study for demonstration purposes. The first system considered is Thunderhead (see Figure 6), a 256-processor homogeneous cluster which can be seen as an evolution of the HIVE project, started in 1997 to build a homogeneous commodity cluster to be exploited in a remote sensing applications. It is composed of 256 dual 2.4 GHz Intel Xeon nodes, each with 1 GB of memory

Fig. 6 Thunderhead Beowulf cluster at NASA's Goddard Space Flight Center in Maryland.

and 80 GB of main memory. The total peak performance of the system is 2457.6 Gflops. Along with the 512-processor computer core, Thunderhead has several nodes attached to the core with 2 GHz optical fiber Myrinet. The proposed cluster-based parallel version of the PPI algorithm proposed in this paper was run from one such nodes, called *thunder1*. The operating system used at the time of experiments was Linux Fedora Core, and MPICH[3] was the message-passing library used.

In order to test the proposed systolic array design in a hardware-based computing architecture, our parallel design was implemented on a Virtex-II XC2V6000-6 FPGA of the Celoxica's ADMXRC2 board. It contains 33, 792 slices, 144 Select RAM Blocks and 144 multipliers (of 18-bit × 18-bit). Concerning the timing performances, we decided to pack the input/output registers of our implementation into the input/output blocks in order to try and reach the maximum achievable performance.

4.2 Hyperspectral Data

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 other buildings in the WTC complex. The data set selected for experiments was geometrically and atmospherically corrected prior to data processing, and consists of 614 × 512 pixels, 224 spectral bands and a total size of 140 MB. The spatial resolution is 1.7 m per pixel. Figure 7(left) 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. A detail of the WTC area is shown in a red rectangle. Vegetated areas appear green in Figure 7(left), while burned areas appear dark gray. Smoke appears bright blue as a result of high spectral reflectance in the 655 nm channel.

At the same time as data collection, a small U.S. Geological Survey (USGS) field crew visited lower Manhattan to collect spectral samples of dust and airfall debris deposits from several outdoor locations around the WTC area (see Figure 8). These spectral samples were then mapped into the AVIRIS data using reflectance spectroscopy and chemical analyses in specialized USGS laboratories. For illustrative purposes, Figure 7(right) shows a thermal map centered at the region where the buildings collapsed. The map shows the target locations of the thermal hot spots, shown as bright red, orange and yellow spots on Figure 7(right). The temperatures range from 700 °F (marked as "F" to 1300 °F (marked as "G". This thermal map, along with a dust/debris map produced by USGS (available online from http://pubs.usgs.gov/of/2001/ofr-01-0429), are used in this work as ground-truth to validate the proposed parallel implementations of the PPI algorithm.

4.3 Performance Evaluation

Before empirically investigating the parallel performance of the proposed algorithms, we first evaluate their endmember extraction and classification 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 different implementations of the PPI algorithm (endmember extraction) in the considered data processing chain, since the subsequent spectral unmixing step executed after endmember extraction only uses as input parameters the endmembers extracted in the previous stage. In all our considered implementations, the number of endmembers to be extracted was set to $E = 16$ after estimating the intrinsic dimensionality of the data using the *virtual dimensionality* concept in Chang (2003). 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 $t_a = 0.1$, which is a reasonable limit of tolerance for this metric, while the threshold value parameter

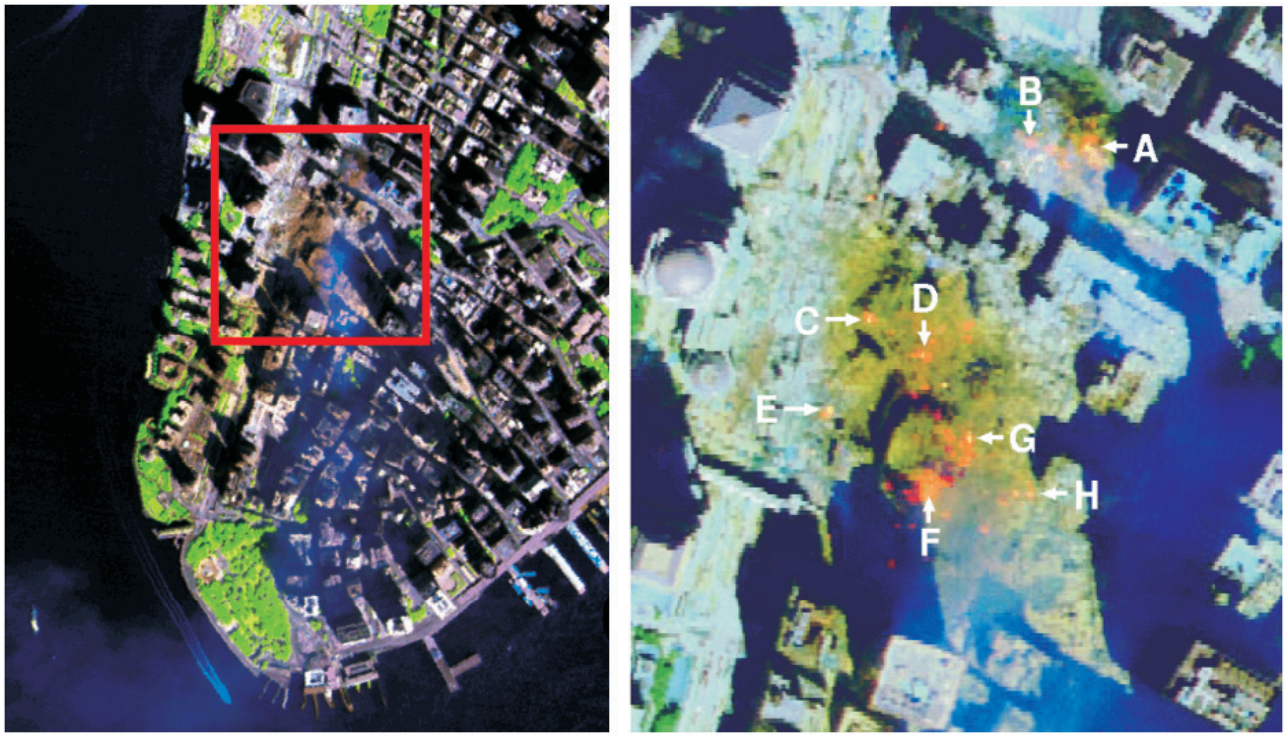

Fig. 7 False color composite of an AVIRIS hyperspectral image collected by NASA's Jet Propulsion Laboratory over lower Manhattan on September 16, 2001 (left). Location of thermal hot spots in the fires observed in World Trade Center area, available online: http://pubs.usgs.gov/of/2001/ofr-01-0429/hotspot.key.tgif.gif (right).

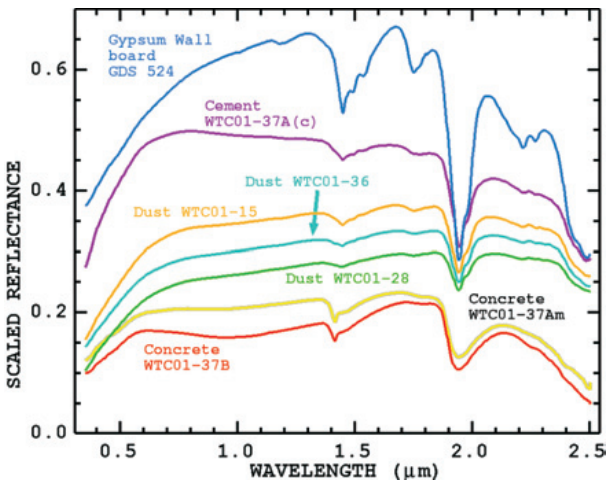

Fig. 8 Spectral signatures of sample materials in the WTC area, available online: http://speclab.cr.usgs.gov/wtc/clark-wtc-chapter/figure3.gif.

t_v was set to the mean of PPI scores obtained after $K = 10^4$ iterations. These parameter values are in agreement with those used before in the literature (Plaza et al. 2004).

Using the algorithm parameters above, an experiment-based cross-examination of algorithm endmember extraction accuracy is presented in Table 1, which tabulates the SAM scores obtained after comparing field spectra corresponding to different materials, obtained by USGS on the WTC area, with the corresponding endmembers extracted by the parallel algorithms. The smaller the SAM values across the field spectra considered in Table 1, the better the results. It should be noted that Table 1 only displays the smallest SAM scores of all endmembers with respect to each USGS signature for each algorithm.

On the other hand, Table 2 estimates the accuracy of the parallel algorithms in estimating the sub-pixel abundance of fires in Figure 7(right) by taking advantage of available USGS reference information regarding the area covered by each thermal hot spot at the time the AVIRIS data was collected (given in square meters). It should be noted that each pixel in the AVIRIS data is has a size of 1.7 square meters. As a result, all thermal hot spots are

Table 1
SAM-based spectral similarity scores between endmembers extracted by different parallel implementations of the hyperspectral data processing chain and USGS reference signatures.

Dust/debris class	ENVI	Multiprocessor	FPGA
Gypsum wall board – GDS 524	0.081	0.089	0.089
Cement – WTC01-37A(c)	0.094	0.094	0.099
Dust – WTC01-15	0.077	0.077	0.077
Dust – WTC01-36	0.086	0.086	0.086
Dust – WTC01-28	0.069	0.069	0.069
Concrete – WTC01-37Am	0.073	0.073	0.073
Concrete – WTC01-37B	0.090	0.090	0.090

Table 2
Comparison of area estimation (in square meters) for each thermal hot spot by different parallel implementations of the hyperspectral data processing chain (USGS ground area estimations are also included, along with latitude and longitude coordinates, and the temperature of fires).

Thermal hot spot	Latitude (North)	Longitude (West)	Temperature (Kelvin)	Area (USGS)	Area (ENVI)	Area (Multiprocessor)	Area (FPGA)
A	40°42′47.18″	74°00′41.43″	1000	0.56	0.53	0.53	0.53
B	40°42′47.14″	74°00′43.53″	830	0.08	0.06	0.10	0.06
C	40°42′42.89″	74°00′48.88″	900	0.80	0.78	0.78	0.78
D	40°42′41.99″	74°00′46.94″	790	0.80	0.81	0.81	0.83
E	40°42′40.58″	74°00′50.15″	710	0.40	0.55	0.59	0.57
F	40°42′38.74″	74°00′46.70″	700	0.40	0.36	0.31	0.31
G	40°42′39.94″	74°00′45.37″	1020	0.04	0.05	0.05	0.05
H	40°42′38.60″	74°00′43.51″	820	0.08	0.12	0.12	0.12

sub-pixel in nature and hence require a spectral unmixing step as the last one provided in our proposed PPI. Summarizing, experiments in Table 1 reveal that the proposed parallel implementations can extract endmembers which very accurately resemble ground (pure) spectral signatures, while experiments in Table 2 demonstrate that the parallel algorithms can provide accurate estimations of the area covered by thermal hot spots, which can lead to better fire characterization results. In particular, it is worth noting that the estimations for the thermal hot spots with higher temperature (labeled "A" and "G" in the table) were almost perfect, and identical for the three considered implementations.

It is important to note that the output produced by all parallel methods in Tables 1 and 2 was verified using not only our own serial implementations, but the original version of PPI available in Kodak's Research Systems ENVI software version 4.0 as well (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 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 of PPI produced slightly different results than those found by ENVI's PPI. In particular, only 1 out of 16 endmembers produced by ENVI's PPI (gypsum wall board) was not included in the final endmember set produced by the multiprocessor version of PPI, while 2 out of 16 endmembers produced by ENVI's PPI (gypsum wall board and cement) were not found by our FPGA implementation of PPI. However, we experimentally tested that the SAM scores between the endmembers that were different between the original and parallel algorithms were always very low (below 0.015), a fact that reveals that the final endmembers sets were almost identical, spectrally.

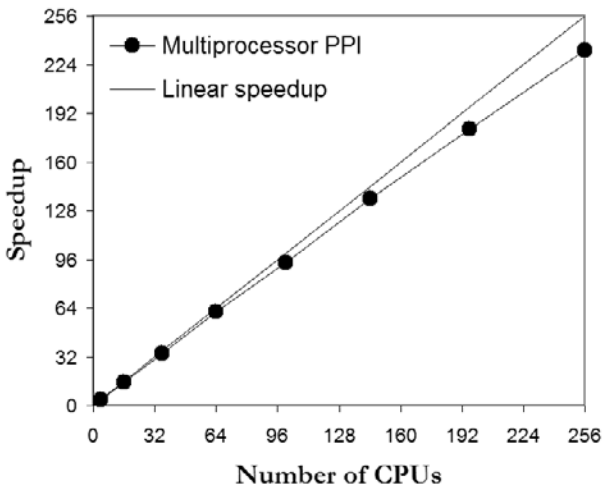

Fig. 9 Scalability of the multiprocessor implementation of the endmember extraction algorithm on NASA's Thunderhead cluster.

4.3.1 Experiments on the Beowulf cluster To empirically investigate the parallel properties of our multiprocessor PPI implementation, we tested its performance on NASA's Thunderhead Beowulf cluster. For that purpose, Figure 9 plots the speedups achieved by the multiprocessor runs over a single-processor run of our C++ implementation of the PPI algorithm using only one Thunderhead processor. It should be noted that the speedup factors in Figure 9 were calculated as follows: the real time required to complete a task on P processors, $T(P)$, was approximated by $T(P) = A_P + (B_P/P)$, where A_P is the sequential (non-parallelizable) portion of the computation and B_P is the parallel portion. In our parallel code, A_P corresponds to the *data partitioning* and *endmember selection* steps (performed by the master), while B_P corresponds to the *skewer generation*, *extreme projections*, *candidate selection* and *spectral unmixing* steps, which are performed (in "embarrassingly parallel" fashion) at the different workers. With the above assumptions in mind, we can define the speedup for P processors, S_P, as follows:

$$S_P = \frac{T(1)}{T(P)} \approx \frac{A_P + B_P}{A_P + (B_P/P)} \quad (8)$$

In the above relationship, known as Amdahl's Law (Hennessy and Patterson 2002), $T(1)$ denotes the single-processor execution time. It is obvious from this expression that the speedup of a parallel algorithm does not continue to increase with increasing the number of processors. The reason is that the sequential portion A_P is proportionally more important as the number of processors increase and, thus, the performance of the parallelization is generally degraded for a large number of processors. In our experiments, we have observed that although the speedup plot in Figure 9 flattens out a little for a large number of processors, it is still very close to linear speedup. For the sake of quantitative comparison, Table 3 reports the measured execution times by the multiprocessor runs of our parallel algorithm on Thunderhead. Results in Table 3 reveal that our multiprocessor implementation of PPI can effectively adapt to a massively parallel environment and provide a response in real-time (i.e. in less than 5 s for our considered problem size) but using a large number of processors.

To further explore the parallel properties of the considered algorithms in more detail, an in-depth analysis of computation and communication times achieved by the proposed multiprocessor implementation is also highly desirable. For that purpose, Table 4 shows the total time spent by the parallel implementation in communications and computations in the considered Beowulf cluster, where two types of computation times were analyzed, namely, sequential (those performed by the root node with no other parallel tasks active in the system, labeled as A_P in the table) and parallel (the rest of computations, i.e. those performed by the root node and/or the workers in parallel, labeled as B_P in the table). The latter includes the times in which the workers remain idle. Since communications (labeled as C_P in the table) only take place at the beginning (initial data scatter in the *data partitioning* step) and at the end (final data gather in the *endmember selection* step) of the parallel algorithm, these are not overlapped with the computations in our implementation.

It can be seen from Table 4 that, although A_P scores were non-zero mainly because of the *endmember selec-*

Table 3
Processing times and speedups achieved by the multiprocessor implementation of the hyperspectral data processing chain using different numbers of processors on NASA's Thunderhead Beowulf cluster.

Number of CPUs	1	4	16	36	64	100	144	196	256
Time (s)	1163.05	295.92	76.91	33.97	18.84	12.38	8.57	6.41	4.98
Speedup (S_P)	–	3.93	15.12	34.23	61.73	93.89	135.67	181.34	233.45

Table 4
Sequential computation, parallel computation, and communication times achieved by the multiprocessor implementation on Thunderhead. For illustrative purposes, load-balancing rates considering all processors, and considering all processors but the root, are also displayed.

Number of CPUs	4	16	36	64	100	144	196	256
Sequential computations (A_P)	1.63	1.26	1.12	1.19	1.06	0.84	0.91	0.58
Parallel computations (B_P)	292.09	73.24	30.46	15.64	9.26	6.08	4.28	3.37
Communications (C_P)	2.20	2.41	2.39	2.21	2.46	2.65	2.32	2.49
Load balancing considering all processors (D_{all})	1.15	1.10	1.09	1.11	1.07	1.10	1.05	1.04
Load balancing considering all processors but the root (D_{minus})	1.04	1.02	1.04	1.03	1.01	1.02	1.03	1.01

tion step of the parallel algorithm which is performed at the master node once the workers have finalized their parallel computations, these scores were always very low and irrelevant when compared with the B_P scores, which anticipates high parallel efficiency of the multiprocessor algorithm, even for a very high number of processors. On the other hand, it can also be seen from Table 4 that the impact of communications is not particularly significant since B_P scores were always much higher than C_P scores. Despite the above results seeming promising, the fact that B_P is by far the most significant fraction of the parallel algorithm requires a detailed study of load balance to fully substantiate the parallel properties of the considered algorithm.

To analyze the important issue of load balance in more detail, Table 4 also shows the *imbalance* scores achieved by the multiprocessor implementation of PPI on Thunderhead. The imbalance is defined as $D = R_{max}/R_{min}$, where R_{max} and R_{min} are the maxima and minima processor run times, respectively. Therefore, perfect balance is achieved when $D = 1$. In the table, we display the imbalance considering all processors, D_{all}, and also considering all processors but the root, D_{minus}. As we can see from Table 4, the multiprocessor PPI was able to provide values of D_{all} close to 1 in all considered networks. Further, the algorithm provided almost the same results for both D_{all} and D_{minus}, which indicates that the workload assigned to the master node is balanced with regards to that assigned to the workers.

Although the results presented above demonstrate that the proposed multiprocessor implementations of PPI algorithm is satisfactory from the viewpoint of algorithm scalability, code reusability and load balance in a massively parallel Beowulf cluster, there are several restrictions in order to incorporate this type of platform for onboard processing in remote sensing missions. Although the idea of mounting clusters and networks of processing elements onboard airborne and satellite hyperspectral imaging facilities has been explored in the past, the number of processing elements in such experiments has been very limited thus far, because of payload requirements in most remote sensing missions. For instance, a low-cost, portable Myrinet cluster of 16 processors was recently developed at NASA's GSFC for onboard analysis (see http://thunderhead.gsfc.nasa.gov/PDF/Low_Power.pdf for additional details). The portable system, called Proteus and composed of 12 mini-ITX (PC) nodes, was developed for the purpose of spacecraft/satellite data processing and also used as a mobile Beowulf cluster for field processing of collected data. The cost of the portable cluster was 3000 U.S. dollars. Unfortunately, it could still not facilitate real-time performance since the measured processing times were similar to those reported in Table 3 for the same number of processors on Thunderhead, and the incorporation of additional processing elements to the cluster was reportedly difficult due to heat, power and weight considerations which limited its exploitation for onboard processing.

4.3.2 FPGA experiments As an alternative to cluster computing, FPGA-based computing provides several advantages for image processing, such as increased computational power, adaptability to different applications via reconfigurability, and compact size.[4] Specifically, the cost of the Xilinx Virtex-II XC2V6000-6 FPGA used for experiments in this work is currently only slightly higher than that of the portable Myrinet cluster mentioned in the previous subsection. However, the mobile cluster required several 9U VME motherboards to accommodate the multiple processors, with an approximate weight of 14 lbs and required power of 300 watts. On the other hand, the Xilinx Virtex-II FPGA required only one 3U Compact PCI card (weight below 1 lb) and power of approximately 25 watts, offering full real-time reconfigurability. These are very important considerations from the viewpoint of remote sensing mission payload requirements, which are widely regarded as a key aspect for sensor design and operation. In particular, it is important to note that electronic compo-

nents and hardware likely to compromise mission payload are often discarded from Earth observation instruments, and therefore evaluating the potential use of FPGAs as an alternative to much heavier computer equipment is of great importance for remote sensing mission design and planning.

In order to fully substantiate the performance of our systolic array-based FPGA implementation, we should first describe the scalability of the systolic array. The peak performance of the array is mainly determined by the dot-product capacity, that is the number of additions/subtractions executed in 1 s. It is expressed (in millions of operations per second) as follows:

$$P_{peak} = F \times T/P \tag{9}$$

In the above expression, F is the frequency in MHz and P represents the number of processors of the systolic array. The above formula supposes that the array is constantly fed: on each cycle a new data is available on its input. Unfortunately, this may not be the case, especially if we consider a reconfigurable board plugged through the input/output (I/O) bus system of the micro-processor. The PPI algorithm proceeds into T/P passes, and each pass requires flushing the hyperspectral image from the main memory to the array. Thus, instead of considering that a data is present every clock cycle, it is better to consider the transfer capacity of the I/O bus for estimating the average performance of the array. The average performance is estimated as follows:

$$P_{average} = (B_w \times T)/P \tag{10}$$

In the above equation, B_w denotes the bandwidth expressed in Mbytes/s. Now, if we want to estimate the execution time for computing the PPI algorithm, t_{exec}, the bandwidth should be taken into consideration as follows:

$$t_{exec} = \frac{P \times (T \times N)}{B_w} \tag{11}$$

In the above expression, N is the number of spectral bands. Using the above rationale, we have performed an estimation of the computing time and speedup that can be achieved by the proposed FPGA implementation of the PPI on the considered AVIRIS scene (with 614 × 512 pixels and 224 spectral bands) using a reconfigurable board connected to a microprocessor through its I/O bus. Figure 10(left) shows the estimated computing times considering various bandwidths (from F = 10 Mbytes/s to F = 50 Mbytes/s) and various numbers of processors (P = 100, P = 200 and P = 400). On the other hand, Figure 10(right) shows the speedups compared to a single-processor run of the PPI in one of the Thunderhead nodes, again with a bandwidth ranging from 10 to 50 Mbytes/s and a systolic array with 100, 200 and 400 processors. As can be seen in Figure 10, the achieved speedups can be very high, reducing hours of computation to only a few seconds. In the following, we validate these estimations on a Xilinx FPGA architecture.

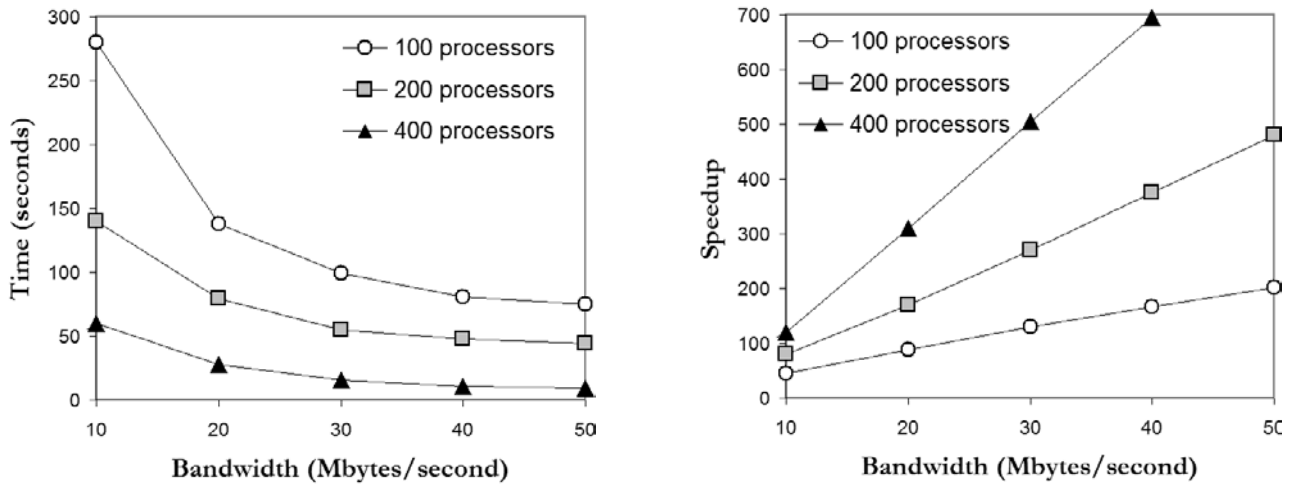

Fig. 10 Time, in seconds, for computing the hyperspectral data processing chain using a reconfigurable board connected to a PC through the I/O bus (left). Speedup compared to a single-processor version running on a single Thunderhead node (right).

Table 5
Summary of resource utilization for the FPGA-based implementation of the hyperspectral data processing chain (operation frequency is given in MHz and processing time is given in seconds).

Number of processors	Total gates	Total slices	Percentage of total	Operation frequency	Processing time
100	97,443	1185	3%	29,257	69.91
200	212,412	3587	10%	21,782	35.56
400	526,944	12,418	36%	18,032	20.48

Table 5 shows a summary of resource utilization by the proposed systolic array-based implementation of the PPI on a complete system (systolic array plus PCI interface), implemented on a XC2V6000-6 board, using different numbers of processors. We measured an average PCI bandwidth of 15 Mbytes between the PC and the board, leading to a speed-up of 120 when running the PPI algorithm with 400 processors. It should be noted that, in our experimentation, the performance was seriously limited by the transfer rate between the PC and the board: the array is able to absorb a pixel flow of above 40 Mbytes/s, while the PCI interface can only provide a flow of 15 Mbytes. This experiment, however, demonstrates that the considered board, even with a non-optimized PCI connection (with no DMA), can still yield very good speedup for the PPI algorithm, with a final measured processing time of about 20 s for $P = 400$ processors. This response, although not strictly in real-time, can be further increased by increasing the number of processors in the FPGA. However, in our opinion it is very important to leave room in the FPGA for additional algorithms so that dynamic algorithm selection can be performed on the fly. In addition, it is also worth noting that full reconfigurability requires additional logic and space in the FPGA (El-Araby et al. 2004). Another reason which led us to try to minimize resource utilization as much as possible is that radiation-tolerant FPGAs (required in satellite-based hyperspectral imaging applications) have two orders of magnitude fewer equivalent gates. Therefore, we have decided to report realistic experiments by resorting to a moderate amount of resources (gates) in the considered FPGA board. As shown in Table 5, when 400 processors are used, only 36% of the total resources in the FPGA are consumed (which is a reasonable figure in light of the above-mentioned considerations).

4.3.3 Discussion Once experimental results for Beowulf clusters and FPGA architectures have been provided, this section discusses the two discussed strategies for parallel implementation of hyperspectral image processing algorithms. Through the detailed analysis of a standard hyperspectral data processing chain available in commercial software but never implemented in parallel before, we have explored two different strategies to increase computational performance of the algorithm (which can take up to several hours of computation in latest-generation desktop computers). One of the considered strategy consisted of a multiprocessor implementation for commodity clusters. This approach seems particularly appropriate for information extraction from very large hyperspectral data archives after the remotely sensed images have been transmitted to Earth.

To fully address the real-time requirements introduced by many remote sensing applications, we have also developed a systolic array-based FPGA implementation of the hyperspectral data processing chain for onboard 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 version makes appropriate use of computing resources in the considered FPGA, and further provides a response in (near) real-time which is believed to be acceptable in most remote sensing applications.

To conclude this section, we emphasize that efficient onboard processing has been a long-awaited goal by the remote sensing community. In this regard, the reconfigurability of FPGA systems opens 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 real-time in applications that certainly demand so, such as military target detection, wildland fire monitoring and tracking, oil spill quantification, and so forth. Although the experimental results presented in this section are very encouraging, further work is still needed to arrive to optimal parallel design and implementations for the proposed data processing chain and other hyperspectral imaging algorithms for image analysis and compression.

5 Conclusions and Future Research

Remotely sensed hyperspectral data processing exemplifies a subject area that has drawn together an eclectic collection of participants. However, a common requirement in most available hyperspectral imaging applications is the need to develop efficient processing systems and architectures able to cope with the extremely high dimensionality of the data. In particular, there is a clear need to develop cost-effective algorithm implementations for dealing with hyperspectral imaging problems, and the goal to speed up algorithm performance has already been identified as an essential target in many on-going and planned remote sensing missions.

In this paper, we have taken a necessary first step toward the development of real-time hyperspectral imaging algorithms. Specifically, two innovative high-performance implementations of a standard hyperspectral data processing chain have been introduced and evaluated from the viewpoint of both algorithm accuracy and parallel performance, including a commodity cluster-based implementation and an FPGA-based hardware implementation. The analytical techniques presented in this work offer an excellent snapshot of the state-of-the-art in the field of HPC in remote sensing.

Performance data for the proposed implementations have been provided in the context of a real, time-critical application. These results reflect the versatility that currently exists in the design of high-performance approaches, a fact that currently allows users to select a specific high-performance architecture that best fits the requirements of their application domains. In this regard, one of the main purposes of this study has been to present current efforts toward the integration of hyperspectral image processing techniques with parallel and distributed computing practices, with the ultimate goal of designing (near) real-time systems expected to introduce substantial changes in the systems currently used by NASA and other agencies for exploiting the sheer volume of Earth and planetary remotely sensed data collected on a daily basis. In particular, one of the main conclusions of our study is that massively parallel clusters may be the tool of choice for processing massive archives of hyperspectral images which have already been transmitted to Earth (the increased storage capacity of those systems is essential to catalog the ever-growing amount of remotely sensed data that is now being collected on a daily basis). On the other hand, applications requiring a quick response onboard, FPGAs provide a solution which is scalable, compact in size and affordable in cost.

Although the experimental results presented in this work are encouraging (especially from the viewpoint of their suitability to address a very important application case study), further work is still needed to arrive to optimal parallel design and implementations for the considered data processing chain. We also plan to implement the proposed parallel techniques on other massively parallel computing architectures, such as NASA's Project Columbia and other computing environments of grid/heterogeneous type (Plaza, Plaza, and Valencia 2007). We are also developing real-time implementations of hyperspectral imaging algorithms on commodity graphics hardware (GPUs; Setoain et al. 2007), which also represent a very appealing type of high performance hardware architecture for onboard hyperspectral image processing.

Acknowledgments

This research was supported in part by the European Commission through the Marie Curie Research Training Network project *Hyperspectral Imaging Network* (MRTN-CT-2006-035927). The authors would like to gratefully thank John E. Dorband, James C. Tilton and J. Anthony Gualtieri for several helpful discussions, and also for their collaboration on experimental results using the Thunderhead Beowulf cluster at NASA's Goddard Space Flight Center. The first author would also like to acknowledge support received from the Spanish Ministry of Education and Science (Fellowship PR2003-0360), which allowed him to conduct postdoctoral research at NASA's Goddard Space Flight Center and University of Maryland, Baltimore County in 2004. Last but not least, the authors gratefully acknowledge the anonymous reviewers for their fruitful suggestions and comments, which greatly helped increase the quality and presentation of the manuscript.

Author Biographies

Antonio Plaza received his Ph.D. degree in computer science from the University of Extremadura, Spain, in 2002, where he is currently an Associate Professor with the Computer Science Department. He has also been a visiting researcher with the University of Maryland, NASA Goddard Space Flight Center and Jet Propulsion Laboratory. His main research interests include the development and efficient implementation of high-dimensional data algorithms on parallel homogeneous and heterogeneous computing systems and hardware-based computer architectures such as FPGAs and GPUs. He has authored or co-authored more than 150 publications including journal papers, book chapters and peer-reviewed conference proceedings, and currently serves as regular manuscript reviewer for more than 15 highly cited journals in the areas of parallel and distributed computing, computer architectures, pattern recognition, image processing and remote sensing. He is editing a book on High-Performance Computing in Remote Sensing (with Prof. Chein-I

Chang) for Chapman & Hall/CRC Press and a special issue on Architectures and Techniques for Real-Time Processing of Remotely Sensed Images for the Journal of Real-Time Image Processing. He is the coordinator of the Hyperspectral Imaging Network, an FP6 Marie Curie Research Training Network European project, and currently serves as Associate Editor for the IEEE Transactions on Geoscience and Remote Sensing journal in the areas of Hyperspectral Image Analysis and Signal Processing.

Chein-I Chang received his B.S. degree from Soochow University, Taipei, Taiwan, M.S. degree from the Institute of Mathematics at National Tsing Hua University, Hsinchu, Taiwan and M.A. degree from the State University of New York at Stony Brook, all in mathematics. He also received his M.S., M.S.E.E. degrees from the University of Illinois at Urbana-Champaign and Ph.D. degree in electrical engineering from the University of Maryland, College Park. Dr Chang has been with the University of Maryland, Baltimore County (UMBC) since 1987 and is currently professor in the Department of Computer Science and Electrical Engineering. He received an NRC (National Research Council) senior research associateship award from 2002–2003 sponsored by the US Army Soldier and Biological Chemical Command, Edgewood Chemical and Biological Center, Aberdeen Proving Ground, Maryland. Additionally, Dr Chang is currently holding a chair professorship of disaster reduction technology from 2006–2009 with the Environmental Restoration and Disaster Reduction Research Center, National Chung Hsing University, Taichung, Taiwan, ROC. He has three patents and several pending on hyperspectral image processing. He is on the editorial board of the Journal of High Speed Networks and was an associate editor in the area of Hyperspectral Signal Processing for the IEEE Transactions on Geoscience and Remote Sensing. Dr Chang is the author of Hyperspectral Imaging: Techniques for Spectral Detection and Classification published by Kluwer Academic Publishers in 2003, as well as several other books on hyperspectral imaging. He is a Fellow of SPIE and a member of Phi Kappa Phi and Eta Kappa Nu.

Notes

1 http://www.celoxica.com

2 http://www.xilinx.com

3 http://www-unix.mcs.anl.gov/mpi/mpich

4 http://www.xilinx.com/publications/xcellonline/xcell_47/xc_pdf/xc_boeing47.pdf

References

Achalakul, T. and Taylor, S. (2003). A distributed spectral-screening PCT algorithm, *Journal of Parallel and Distributed Computing* **63**(3): 373–384.

Aloisio, G. and Cafaro, M. (2003). A dynamic earth observation system, *Parallel Computing* **29**(10): 1357–1362.

Boardman, J., Kruse, F. A., and Green, R. O. (1993). Mapping target signatures via partial unmixing of AVIRIS data. In *Summaries of Airborne Earth Science Workshop*, JPL Publication 93–26, pp. 111–114.

Chang, C.-I (2003). *Hyperspectral imaging: Techniques for spectral detection and classification*, New York: Kluwer.

Chang, C.-I (2007). *Hyperspectral data exploitation: theory and applications*, New York: Wiley.

Chang, C.-I and Plaza, A. (2006). A fast iterative implementation of the pixel purity index algorithm, *IEEE Geoscience and Remote Sensing Letters* **3**(1): 63–67.

Chen, L., Fujishiro, I., and Nakajima, K. (2003). Optimizing parallel performance of unstructured volume rendering for the Earth Simulator, *Parallel Computing* **29**(3): 355–371.

Clark, R. N., *et al.* (2006). Environmental mapping of the World Trade Center area with imaging spectroscopy after the September 11, 2001 attack. In *Urban aerosols and their impacts: Lessons learned from the World Trade Center tragedy*, edited by J. Gaffney and N. A. Marley, Oxford University Press, pp. 66–83.

Dhodhi, M. K., Saghri, J. A., Ahmad, I., and Ul-Mustafa, R. (1999). D-ISODATA: A distributed algorithm for unsupervised classification of remotely sensed data on network of workstations, *Journal of Parallel and Distributed Computing* **59**(2): 280–301.

Diefendorff, K., Dubey, P. K., Hochsprung, R., and Scales, H. (2002). AltiVec extension to PowerPC accelerates media processing, *IEEE Micro* **20**(2): 85–96.

Dorband, J., Palencia, J., and Ranawake, U. (2003). Commodity computing clusters at Goddard Space Flight Center, *Journal of Space Communication* **1**(3): 113–123.

Dou, Y., Vassiliadis, S., Kuzmanov, G. K., and Gaydadjiev, G. N. (2005). 64-bit floating-point FPGA matrix multiplication. In *Proceedings of the 13th ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, pp. 86–95.

El-Araby, E., El-Ghazawi, T., Le Moigne, J., and Gaj, K. (2004). Wavelet spectral dimension reduction of hyperspectral imagery on a reconfigurable computer. In *Proceedings of the 4th IEEE International Conference on Field-Programmable Technology*, pp. 399–402.

Fry, T. W. and Hauck, S. (2002). Hyperspectral image compression on reconfigurable platforms. In *Proceedings of the 10th IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 251–260.

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

Gropp, A., Huss-Lederman, S., Lumsdaine, A., and Lusk, E. (1999). *MPI: The complete reference, Vol. 2: The MPI Extensions*, Cambridge, MA: MIT Press.

Hawick, K. A., Coddington, P. D., and James, H. A. (2003). Distributed frameworks and parallel algorithms for processing large-scale geographic data, *Parallel Computing* **29**(10): 1297–1333.

Hennessy, J. L. and Patterson, D. A. (2002). *Computer architecture: A quantitative approach*, 3rd edition, San Mateo, CA: Morgan Kaufmann.

Kalluri, S., Zhang, Z., JaJa, J., Liang, S., and Townshend, J. (2001). Characterizing land surface anisotropy from AVHRR data at a global scale using high performance computing, *International Journal of Remote Sensing* **22**(11): 2171–2191.

Keshava, N. and Mustard, J. F. (2002). Spectral unmixing, *IEEE Signal Processing Magazine* **19**(1): 44–57.

Lavernier, D., Fabiani, E., Derrien, S., and Wagner, C. (1999). Systolic array for computing the pixel purity index (PPI) algorithm on hyperspectral images. In *Proceedings of SPIE*, Vol. 4480, pp. 130–138.

Le Moigne, J., Campbell, W. J., and Cromp, R. F. (2002). An automated parallel image registration technique based on the correlation of wavelet features, *IEEE Transactions on Geoscience and Remote Sensing* **40**(8): 1849–1864.

Patrino, A. (2005). Preface to the special issue on climate modeling, *International Journal of High Performance Computing Applications* **19**(3): 175.

Plaza, A., Martinez, P., Perez, R., and Plaza, J. (2004). A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data, *IEEE Transactions on Geoscience and Remote Sensing* **42**(3): 650–663.

Plaza, A., Valencia, D., Plaza, J., and Martinez, P. (2006). Commodity cluster-based parallel processing of hyperspectral imagery, *Journal of Parallel and Distributed Computing* **66**(3): 345–358.

Plaza, A., Plaza, J., and Valencia, D. (2007). Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data, *Journal of Supercomputing* **40**(1): 81–107.

Schowengerdt, R. A. (2007). *Remote sensing*, 3rd edition, New York: Academic Press.

Seinstra, F. J., Koelma, D., and Geusebroek, J. M. (2002). A software architecture for user transparent parallel image processing, *Parallel Computing* **28**(7–8): 967–993.

Setoain, J., Prieto, M., Tenllado, C., Plaza, A., and Tirado, F. (2007). Parallel morphological endmember extraction using commodity graphics hardware, *IEEE Geoscience and Remote Sensing Letters* **4**(3): 441–445.

Skoglund, J. and Felsberg, M. (2005). Fast image processing using SSE2. In *Proceedings of the SSBA Symposium on Image Analysis*, 4 pp.

Sterling, T. (2002). *Beowulf cluster computing with Linux*, Cambridge, MA: MIT Press.

Tehranian, S., Zhao, Y., Harvey, T., Swaroop, A., and McKenzie, K. (2006). A robust framework for real-time distributed processing of satellite data, *Journal of Parallel and Distributed Computing* **66**(3): 403–418.

Tilton, J. C. (2005). Method for implementation of recursive hierarchical segmentation on parallel computers, *U.S. Patent Office, Washington, DC*, U.S. Pending Published Application 09/839147.

Valencia, D. and Plaza, A. (2006). FPGA-based compression of hyperspectral imagery using spectral unmixing and the pixel purity index algorithm, *Lecture Notes in Computer Science*, Vol. 3993, pp. 24–31.

Valero-Garcia, M., Navarro, J., Llaberia, J., Valero, M., and Lang, T. (1992). A method for implementation of one-dimensional systolic algorithms with data contraflow using pipelined functional units, *Journal of VLSI Signal Processing* **4**(1): 7–25.

Veeravalli, B. and Ranganath, S. (2003). Theoretical and experimental study on large size image processing applications using divisible load paradigm on distributed bus networks, *Image and Vision Computing* **20**(13): 917–935.

Vladimirova, T. and Wu, X. (2006). On-board partial run-time reconfiguration for pico-satellite constellations. In *Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems*, pp. 262–269.

Wang, P., Liu, K. Y., Cwik, T., and Green, R. O. (2002). MODTRAN on supercomputers and parallel computers, *Parallel Computing* **28**(1): 53–64.

Wang, Y., Rucker, J. T., and Fowler, J. E. (2004). Three-dimensional tarp coding for the compression of hyperspectral images, *IEEE Geoscience and Remote Sensing Letters* **1**(2): 136–140.

Winter, E. M., Schlangen, M. J., Hill, A. B., Simi, C. G., Winter, M. E. (2002). Tradeoffs for real time hyperspectral analysis. In *Proceedings of SPIE* **4725**: 366–371.

Zhang, D. and Pal, S. K. (2002). *Neural networks and systolic array design*, Singapore: World Scientific.