Clusters versus FPGAs for Spectral Mixture Analysis-Based Lossy Hyperspectral Data Compression

Antonio J. Plaza[a]

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

ABSTRACT

The increasing number of airborne and satellite platforms that incorporate hyperspectral imaging spectrometers has soon created the need for efficient storage, transmission and data compression methodologies. In particular, hyperspectral data compression is expected to play a crucial role in many remote sensing applications. Many efforts have been devoted to designing and developing lossless and lossy algorithms for hyperspectral imagery. However, most available lossy compression approaches have largely overlooked the impact of mixed pixels and subpixel targets, which can be accurately modeled and uncovered by resorting to the wealth of spectral information provided by hyperspectral image data. In this paper, we develop a simple lossy compression technique which relies on the concept of spectral unmixing, one of the most popular approaches to deal with mixed pixels and subpixel targets in hyperspectral analysis. The proposed method uses a two-stage approach in which the purest spectral signatures (also called endmembers) are first extracted from the input data, and then used to express mixed pixels as linear combinations of endmembers. 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, right after the terrorist attacks of September 11th. These data are used in this work to evaluate the impact of compression using different methods on spectral signature quality for accurate detection of hot spot fires. Two parallel implementations are developed for the proposed lossy compression algorithm: a multiprocessor implementation tested on Thunderhead, a massively parallel Beowulf cluster at NASA's Goddard Space Flight Center, and a hardware implementation developed on a Xilinx Virtex-II FPGA device. Combined, these parts offer a thoughtful perspective on the potential and emerging challenges of incorporating parallel data compression techniques into realistic hyperspectral imaging problems.

Keywords: Hyperspectral imaging, data compression, parallel implementations, clusters of computers, field programmable gate arrays (FPGAs)

1. INTRODUCTION

Hyperspectral data compression[1] has received considerable interest in recent years due to the enormous data volumes collected by latest-generation imaging spectrometers[2] such as NASA Jet Propulsion Laboratory's Airborne Visible Imaging Infra-Red Imaging Spectrometer (AVIRIS),[3] which is now able to collect hundreds of contiguous spectral bands with high between-band spectral correlation. Two approaches have been explored in the literature to address the relevant issue of hyperspectral data compression prior to the analysis: lossless and lossy, in accordance with redundancy removal. Which compression framework should be used depends heavily upon the application domain. Since we are interested in exploitation-based applications, our target hyperspectral data analysis techniques are generally determined by features of objects in the image data rather than the image itself As a result, lossless compression may not offer significant advantages over lossy compression in the sense of feature extraction.

The success of a lossy compression technique is generally measured by whether or not its effectiveness meets a preset desired goal which in turn determines which criterion should be used for compression. As an example, principal component analysis (PCA)[4] can be seen as a compression technique that represents data in a few principal components determined by data variances. Its underlying assumption is based on the fact that the data are well represented and structured in terms of variance, where most of data points are clustered and can be packed in a low dimensional space. However, it has

Further author information: (Send correspondence to Antonio J. Plaza)
Antonio J. Plaza: E-mail: aplaza@unex.es, Telephone: +34 927 257 195

Satellite Data Compression, Communication, and Processing IV
edited by Bormin Huang, Roger W. Heymann, Joan Serra-Sagrista
Proc. of SPIE Vol. 7084, 708402, (2008) · 0277-786X/08/$18 · doi: 10.1117/12.798326

been shown that signal-to-noise ratio (SNR) is generally a better measure than data variance to measure image quality in multispectral imagery. Similarly, the mean squared error (MSE) has been also widely used as a criterion for optimality in communications and signal processing such as quantization. However, it is also known that it may not be appropriate to be used as a measure of image interpretation. This is particularly true for hyperspectral imagery which can uncover many unknown signal sources, some of which may be very important in data analysis such as anomalies, small targets which generally contribute very little to SNR or MSE. In the PCA these targets may only be retained in minor components instead of principal components. So, preserving only the first few principal components may lose these targets. In SNR or MSE, such targets may very likely be suppressed by lossy compression if no extra care is taken since missing these targets may only cause inappreciable loss of signal energy or small error.

By realizing the importance of hyperspectral data compression, many efforts have been devoted to design and development of compression algorithms for hyperspectral imagery.[1] Some of these techniques are based on a direct extension of 2D image compression to 3D image compression where many 2D image compression algorithms that have proven to be efficient and effective in 2D images are extended to 3D algorithms. Two techniques of particular interest will be used in our investigation: the JPEG2000 multicomponent[5] and the 3D-SPIHT.[6,7] Despite a hyperspectral image can be considered as an image cube and therefore processed by the techniques above, our experimental results in this work show that a direct application of 3D image compression to hyperspectral data may not be able to accurately preserve the very rich spectral information that is present in hyperspectral data cubes. In particular, using MSE or SNR as a compression criterion may result in significant loss of spectral information for data analysis.

Our main focus in this work is to design lossy compression techniques able to reduce significantly the large volume of information contained in the original hyperspectral data cube while, at the same time, being able to retain information that is crucial to deal with mixed pixels and subpixel targets.[8] These two types of pixels are essential in many hyperspectral analysis applications, including military target detection and tracking, environmental modeling and assessment at subpixel scales. A subpixel target is a mixed pixel with size smaller than the available pixel size (spatial resolution). So, it is embedded in a single pixel and its existence can only be verified by using the wealth of spectral information provided by hyperspectral sensors. A mixed pixel is a mixture of two or more different substances present in the same pixel. In this case, spectral information can greatly help to effectively characterize the substances within the mixed pixel via spectral unmixing techniques. When hyperspectral image compression is performed, it is critical and crucial to take into account these two issues, which have been generally overlooked in the development of lossy compression techniques in the literature. In this work, we address this issue by designing a novel lossy hyperspectral data compression technique which relies on the application of linear spectral mixture analysis techniques[9] which are able to retain the information essential to characterize mixed pixels and subpixel targets while significantly reducing the original data volume.

In order to provide an experimental validation framework for the techniques developed in this work, we address a realistic and time-critical application in which a quick response of algorithm analysis is essential. Specifically, the application case study discussed in this paper consists of a detailed assessment of how accurately the proposed lossy compression techniques can retain the sub-pixel information required to characterize fires in a set of hyperspectral data cubes collected by the AVIRIS instrument over the World Trade Center area in New York City, shortly after September 11th, 2001. In order to address the computational requirements of the proposed application case study, we have implemented the proposed lossy compression techniques on massively parallel computing facilities, including Thunderhead *, a 256-processor Beowulf cluster at NASA's Goddard Space Flight Center in Maryland, and a Xilinx Virtex-II field programmable gate array (FPGA) with millions of logical gates and quite appealing from the viewpoint of onboard data compression. The detailed comparison of the proposed techniques both from the viewpoint of spectral fidelity after data compression and computational performance offers a thoughtful perspective on the potential and emerging challenges of incorporating parallel processing systems into the design of lossy hyperspectral image compression techniques.

The remainder of the paper is organized as follows. Section 2 describes the proposed linear spectral mixture-based lossy compression techniques. Section 3 develops parallel versions of the proposed algorithms for efficient implementation in clusters of computers and FPGAs. Section 4 provides experimental results comparing the performance of the proposed techniques with regards to existing ones in terms of both information preservation after compression and parallel performance. Section 5 concludes with some remarks and hints at future research lines.

*http://thunderhead.gsfc.nasa.gov

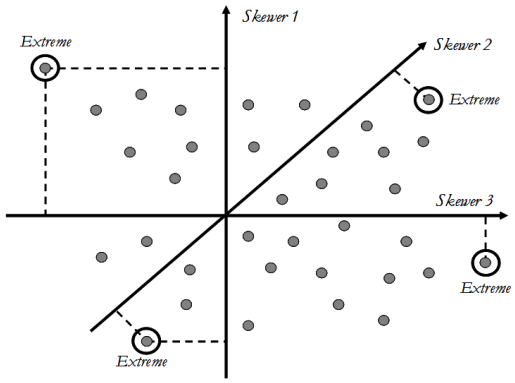

Figure 1. Toy example illustrating the performance of the PPI endmember extraction algorithm in a 2-dimensional space.

2. LINEAR SPECTRAL MIXTURE-BASED COMPRESSION

This section develops an application-oriented, lossy compression algorithm for hyperspectral image data which relies on the utilization of the linear mixture model.[9] The idea of the proposed lossy data compression algorithm is to represent a hyperspectral image cube by a set of fractional abundance images.[10] Let us assume that a remotely sensed hyperspectral scene with N bands is denoted by $\mathbf{F}$, in which a pixel at discrete spatial coordinates is represented by a vector $\mathbf{f} = [f_1, f_2, \cdots, f_N]$, where f_k denotes the spectral response at the k-th wavelength, with $k = 1, \ldots, N$. Under the linear mixture model assumption, each pixel vector in the original scene can be modeled using the following expression:

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

where $\mathbf{e}_i$ designates the i-th pure spectral component (endmember) residing in the pixel, a_i is a scalar value designating the fractional abundance of the endmember $\mathbf{e}_i$ at the pixel $\mathbf{f}$, E is the total number of endmembers, and $\mathbf{n}$ is a noise vector. The solution of the linear spectral mixture problem described in (1) and adopted in our work relies on the correct determination of a set $\{\mathbf{e}_i\}_{i=1}^{E}$ of endmembers and their correspondent abundance fractions $\{a_i\}_{i=1}^{E}$ at each pixel $\mathbf{f}$. Two physical constrains are generally imposed into the model described in (1), these are the abundance non-negativity constraint (ANC), i.e., $a_i \geq 0$, and the abundance sum-to-one constraint (ASC), i.e., $\sum_{i=1}^{E} a_i = 1$.[11] By using the linear mixture model above, we can represent each pixel vector $\mathbf{f}$ by its associated, linearly derived abundance vector $\{a_i\}_{i=1}^{E}$ with E dimensions, which can be seen as a fingerprint of $\mathbf{f}$ with regards to E endmembers which are generally extracted from the original input scene by a fully automatic algorithm.

One of the most successful algorithms for automatic endmember extraction in the literature has been the PPI algorithm.[12] 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 Fig. 1). 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:

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:

$$|\mathbf{f}_i \cdot \mathbf{skewer}_j| = \frac{\mathbf{f}_i \cdot \mathbf{skewer}_j}{\|\mathbf{f}_i\| \cdot \|\mathbf{skewer}_j\|} \tag{2}$$

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 **x** to that particular set as follows:

$$I_S(\mathbf{f}_i) = \begin{cases} 1 \text{ if } \mathbf{x} \in S \\ 0 \text{ if } \mathbf{x} \notin S \end{cases} \tag{3}$$

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:

$$N_{PPI}(\mathbf{f}_i) = \sum_{j=1}^{K} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i) \tag{4}$$

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)[8] 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:

$$\text{SAM}(\mathbf{f}_i, \mathbf{f}_j) = \cos^{-1} \frac{\mathbf{f}_i \cdot \mathbf{f}_j}{\|\mathbf{f}_i\| \cdot \|\mathbf{f}_j\|} \tag{5}$$

It should be noted that, although other spectral similarity metrics have been used in hyperspectral imaging research, [8] 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*. For each sample pixel vector $\mathbf{f}_i$ in **F**, a set of abundance fractions specified by $\{a_{i1}, a_{i2}, \cdots, a_{iE}\}$ is obtained and used to represent the pixel in compressed version 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:

$$\mathbf{f}_i = \mathbf{e}_1 \cdot a_{i1} + \mathbf{e}_2 \cdot a_{i2} + \cdots + \mathbf{e}_E \cdot a_{iE} \tag{6}$$

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 **F**, and for all $e = \{1 \cdots E\}$, where E is the total number of endmembers extracted by the PPI. As a result, only the full set of endmembers and the pixel-level fractional abundances are required to express each pixel in compressed fashion. As shown in a previous study,[13] the number of skewers required for the PPI algorithm needs to be generally high (in the order of $K = 10^4$ or $K = 10^5$) to produce an accurate final set of endmembers, and results in high computation times. In the following section, we provide an overview of parallel computing techniques applied to speed up computational performance of the proposed lossy data compression framework.

3. PARALLEL IMPLEMENTATIONS

This section first develops a parallel implementation of the lossy hyperspectral compression technique 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.

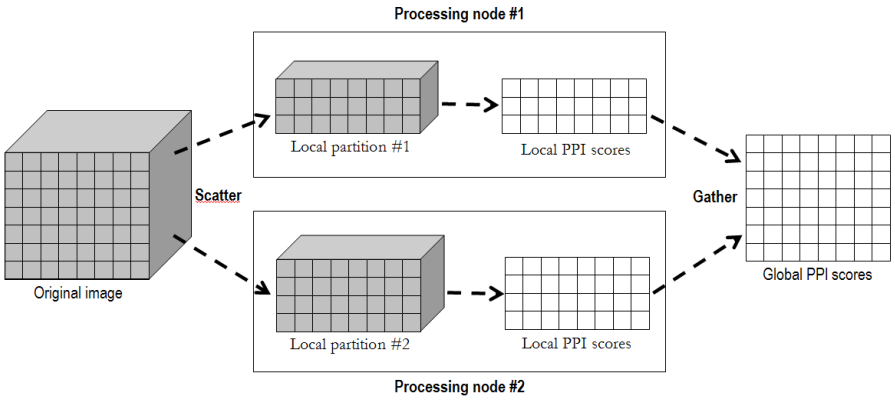

Figure 2. Domain decomposition adopted in the parallel implementation of the endmember extraction part of the considered hyperspectral data compression technique.

3.1 Multiprocessor implementation

In this subsection, we describe a master-slave parallel version of the proposed hyperspectral data compression framework. 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, [14] 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 Fig. 2).

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 multi-band data into blocks made up of contiguous spectral bands or sub-volumes, and assigns one or more sub-volumes to each processing element) is not appropriate in our application. [15] This is because the latter approach breaks the spectral identity of the data as each pixel vector is split amongst several processing element. A further reason that justifies the above decision is that, in spectral-domain partitioning, the calculations made for each hyperspectral pixel need to originate from several processing elements, and thus require intensive inter-processor communication. Therefore, in our proposed implementation, a master-worker spatial domain-based decomposition paradigm is adopted, where the master processor sends partial data to the workers and coordinates their actions. Then, the master gathers the partial results provided by the workers and produces a final result.

As it was the case with the serial version, the inputs to our cluster-based implementation 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; 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 scatter*. Produce a set of L spatial-domain homogeneous partitions of $\mathbf{F}$ 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 $\mathbf{skewer}_j$ denoted by $S^{(l)}_{extrema}(\mathbf{skewer}_j)$. 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:

$$N^{(l)}_{PPI}(\mathbf{f}_i^{(l)}) = \sum_{j=1}^{K} I_{S^{(l)}_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i^{(l)}) \tag{7}$$

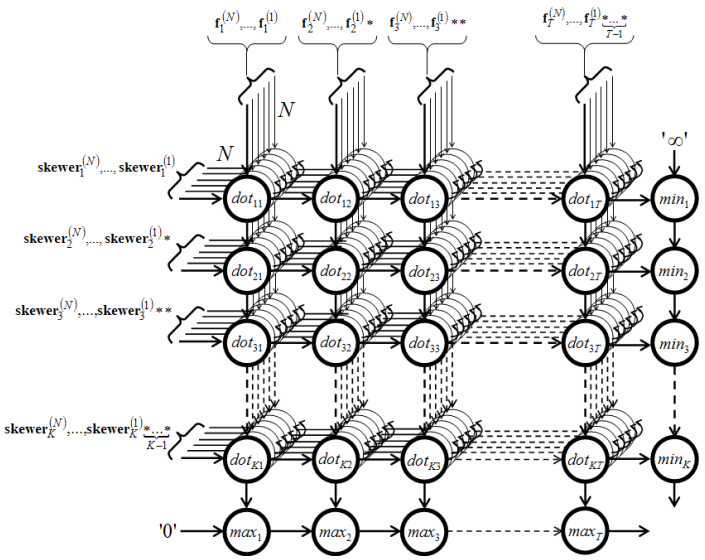

Figure 3. Systolic array design for the proposed FPGA implementation of the hyperspectral data compression technique.

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:

$$\mathbf{f}_i^{(l)} = \mathbf{e}_1 \cdot a_{i1}^{(l)} + \mathbf{e}_2 \cdot a_{i2}^{(l)} + \cdots + \mathbf{e}_E \cdot a_{iE}^{(l)} \tag{8}$$

7. *Data gather*. The master collects the individual results provided by the workers and forms the final compressed version of the original image.

It should be noted that the proposed parallel algorithm has been implemented in the C++ programming language, using calls to *message passing interface* (MPI). 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 [16] for implementation of the proposed hyperspectral data compression technique which is aimed at enhancing replicability and reusability of slices in FPGA devices through the utilization of systolic array design. [17] 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. Using this procedure, we can also calculate the data dependencies prior to the design, and 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 related with limited resource availability and inefficiencies in hardware replication and reusability.

Algorithm 1 Parallel implementation of *extreme projections* step of the proposed hyperspectral data compression algorithm

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

```
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; }
  }
 }
}
```

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 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.

Fig. 3 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 Fig. 3, 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, $\mathbf{f}_i^{(n)}$ 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 Fig. 3 perform the individual products for the skewer projections. On the other hand, the nodes labeled as *max* and *min* respectively compute the maxima and minima projections after the dot product calculations have been completed. In fact, the *max* and *min* nodes can be respectively seen as part of a 1-D systolic array which avoids broadcasting the pixel while simplifying the collection of the results. The operational functionality of each *dot* processing node in Fig. 3 is depicted in Fig. 4. 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 Fig. 4 also illustrates the performance of the *min* and *max* processing nodes in Fig. 3 (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.

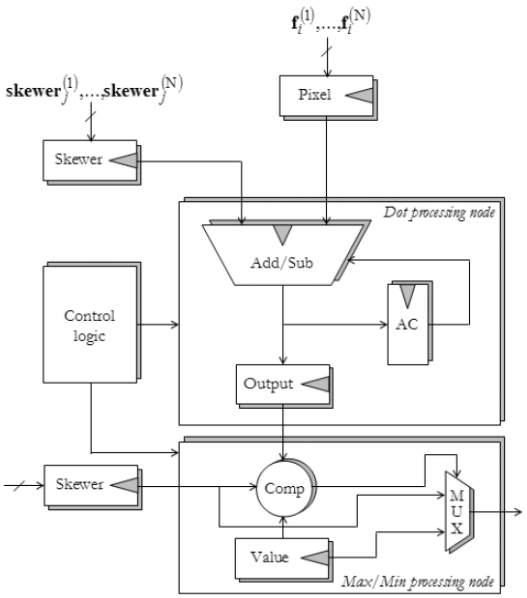

Figure 4. Architecture of each dot and max/min processing nodes in the proposed systolic array design.

Basically, a systolic cycle in the architecture described in Fig. 3 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 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 $\sum_{k=1}^{N} \mathbf{f}_i^{(k)} \times \mathbf{skewer}_j^{(k)}$. 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.[17] The dot product is thus reduced to an accumulation of positive and negative values (the self-connections in the dot nodes of Fig. 3 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 Figs. 3 and 4 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 Fig. 3 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 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. 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 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 [†], a design and prototyping language that allows using a pseudo-C programming style. For illustrative purposes, the source code in

[†]http://www.celoxica.com

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[‡] to simplify the final steps of the hardware implementation, we ported the implementation into a Virtex-II FPGA. These tools allowed us to evaluate the total amount of resources needed which will be discussed in the following section in the context of a real application case study.

Algorithm 3 Handel-C (high level) FPGA implementation of the endmember extraction stage (performed by the PPI module) in the proposed hyperspectral data compression algorithm

```
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];
```

4. EXPERIMENTAL RESULTS

This section evaluates the effectiveness of the proposed hyperspectral data compression technique. First, we describe the hyperspectral data sets used for evaluation purposes. A detailed survey on algorithm performance in a real application domain is then provided, along with a discussion on the advantages and disadvantages of the proposed lossy compression technique in terms of hyperspectral data fidelity after compression. The section concludes with an evaluation of parallel performance of the proposed high performance implementations.

4.1 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

[‡]http://www.xilinx.com

Figure 5. Spectral band of an AVIRIS hyperspectral image collected by NASA's Jet Propulsion Laboratory over lower Manhattan on Sept. 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).

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 meters per pixel. The leftmost part of Fig. 5 shows a spectral band of the data set selected for experiments, in which a detail of the WTC area is shown in a rectangle. The rightmost part of Fig. 5 shows a thermal map generated by U.S. Geological Survey and centered at the region where the buildings collapsed. The map displays the locations of the thermal hot spots with temperatures ranging from $700\ ^oF$ (marked as "F" to 1300^oF (marked as "G". This thermal map will be used in this work as ground-truth to substantiate the capacity of the proposed compression algorithm to retain the sub-pixel spectral information required to detect the hot spot fires in the WTC complex.

4.2 Experimental assessment of data fidelity after compression

Prior to a full examination and discussion of compression results, it is important to outline parameter values used for the different stages of the proposed hyperspectral lossy compression technique. Specifically, the number of endmembers to be extracted in the WTC data was set to $E = 20$ after estimating the intrinsic dimensionality of the data using the *virtual dimensionality* concept.[8] 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 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.[13]

In order to explore the degree of spectral fidelity of the compressed images obtained after applying the proposed compression method, Table 1 reports the SAM-based spectral similarity scores among the pixels labeled as hot spots in the original image (see rightmost part of Fig. 5) and the pixels at the same spatial locations in the images obtained after applying the proposed lossy hyperspectral data compression technique (in the table, the lowest the scores, the highest the spectral similarity). In experiments, compression ratios of 20:1, 40:1 and 80:1 (given by different tested values of input parameter E) were used. For illustrative purposes, we have also included the results provided by two standard methods in our comparison, i.e., the wavelet-based JPEG2000 multicomponent[5] and the 3D-SPIHT.[7] The JPEG2000 implementation used for our experiments was the one available in kakadu software [§]. Both techniques are 3D compression algorithms that treat the hyperspectral data as a 3D volume, where the spectral information is the third dimension. As expected, as the compression ratio was increased, the quality of extracted endmembers was decreased. Results in Table 1 show that, for the same compression ratio, a 3D lossy compression algorithm may result in significant loss of spectral information which can be preserved better, in turn, by an application-oriented algorithm such as the proposed lossy compression approach.

[§] http://www.kakadusoftware.com

Table 1. SAM-based spectral similarity scores between the original hyperspectral image pixels labeled as hot spots in the AVIRIS scene over the World Trade Center and the corresponding pixels in the hyperspectral images obtained after compression using different techniques.

Hot	Lat.	Long.	Temp.	JPEG2000			3D-SPIHT			Proposed		
spot	(North)	(West)	(Kelvin)	20:1	40:1	80:1	20:1	40:1	80:1	20:1	40:1	80:1
'A'	$40^o42'47.18"$	$74^o00'41.43"$	1000	0.114	0.125	0.136	0.107	0.112	0.124	0.065	0.082	0.092
'B'	$40^o42'47.14"$	$74^o00'43.53"$	830	0.124	0.131	0.140	0.109	0.113	0.129	0.058	0.077	0.094
'C'	$40^o42'42.89"$	$74^o00'48.88"$	900	0.098	0.117	0.129	0.106	0.111	0.126	0.068	0.079	0.093
'D'	$40^o42'41.99"$	$74^o00'46.94"$	790	0.135	0.144	0.151	0.115	0.123	0.138	0.073	0.082	0.099
'E'	$40^o42'40.58"$	$74^o00'50.15"$	710	0.123	0.139	0.150	0.110	0.119	0.141	0.071	0.080	0.090
'F'	$40^o42'38.74"$	$74^o00'46.70"$	700	0.116	0.128	0.146	0.099	0.110	0.123	0.055	0.069	0.083
'G'	$40^o42'39.94"$	$74^o00'45.37"$	1020	0.119	0.133	0.149	0.103	0.120	0.128	0.062	0.071	0.088
'H'	$40^o42'38.60"$	$74^o00'43.51"$	820	0.130	0.144	0.152	0.119	0.131	0.145	0.069	0.078	0.091

It should be noted that all the algorithms reported on Table 1 required several minutes of computation to compress the AVIRIS WTC data set in a PC with AMD Athlon 2.6 GHz processor and 512 MB of RAM (specifically, the proposed lossy compression algorithm required almost one hour).

In the following, we report the increase in performance in the proposed compression algorithm using different types of parallel processing architectures. Before describing such experiments, it is important to note that the output produced by all parallel implementations developed for the proposed compression algorithm were verified using not only the corresponding serial implementations, but also the original version of PPI algorithm available in Kodak's Research Systems ENVI software version 4.0 (using the same algorithm parameters) to make sure that exactly the same results were obtained by all tested modules in all cases.

4.3 Experimental assessment of parallel performance

Before describing our parallel performance results, we provide an overview of the HPC platforms used in this study for demonstration purposes. The first considered system is Thunderhead, 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 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 fibre Myrinet. The proposed cluster-based parallel version of the PPI algorithm proposed in this paper was run from one of such nodes, called *thunder1*. The operating system used at the time of experiments was Linux Fedora Core, and MPICH[¶] was the message-passing library used. In order to test the proposed systolic array design in a hardware-based computing architecture, our lossy compression algorithm was also 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 $\times$ 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.

4.3.1 Performance results on the Beowulf cluster

To empirically investigate the parallel properties of our multiprocessor implementation of the proposed data compression algorithm, we tested its performance on NASA's Thunderhead Beowulf cluster. For that purpose, Fig. 6 plots the speedups achieved by several multiprocessor runs over a single-processor run of the compression algorithm using only one Thunderhead processor. It should be noted that the speedup factors in Fig. 6 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 'embarrasingly parallel' fashion) at the different workers. With the above assumptions in mind, we can define the speedup for P processors, S_P, as follows:

[¶]http://www-unix.mcs.anl.gov/mpi/mpich

Table 2. Processing times and speedups achieved by the multiprocessor implementation of the proposed hyperspectral data compression algorithm using different numbers of processors on NASA's Thunderhead Beowulf cluster.

Number of CPUs	**1**	**4**	**16**	**36**	**64**	**100**	**144**	**196**	**256**
Time (seconds)	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

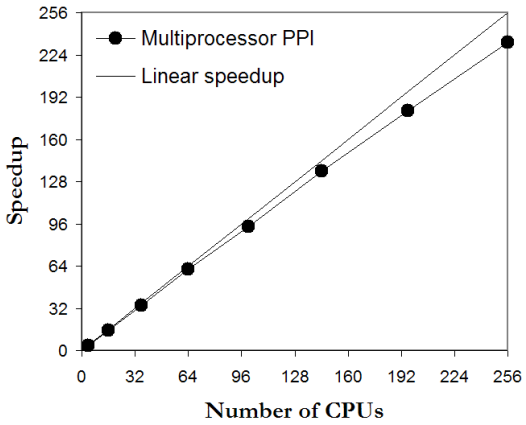

Figure 6. Scalability of the multiprocessor implementation of the proposed data compression algorithm on NASA's Thunderhead cluster.

$$S_P = \frac{T(1)}{T(P)} \approx \frac{A_P + B_P}{A_P + (B_P/P)} \tag{9}$$

In the above relationship, $T(1)$ 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 Fig. 6 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 2 reports the measured execution times by the multiprocessor runs of our parallel algorithm on Thunderhead. Results in Table 2 reveal that our multiprocessor implementation can effectively adapt to a massively parallel environment and provide a response in near real-time (i.e., in less than 5 seconds for our considered problem size) but using a large number of processors.

Although the results presented above demonstrate that the proposed multiprocessor implementation is satisfactory, 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, due to payload requirements in most remote sensing missions. For instance, a low-cost, portable Myrinet cluster of 16 processors was recently developed at NASA's Goddard Space Flight Center for onboard analysis [‖]. 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 3,000 U.S. dollars. Unfortunately, it could still not facilitate real-time performance since the measured processing times were similar to those reported in Table 2 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 Performance results on the field programmable gate array

As an alternative to cluster computing, FPGA-based computing provides several advantages for onboard data compression, such as increased computational power, adaptability to different applications via reconfigurability, and compact

[‖] http://thunderhead.gsfc.nasa.gov/PDF/Low_Power.pdf

size**. 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 consideration 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 components and hardware susceptible of compromising 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 one second. It is expressed (in millions of operations per second) as follows:

$$P_{peak} = F \times T/P \tag{10}$$

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 trough 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:

$$P_{average} = (B_w \times T)/P \tag{11}$$

In the above equation, B_w denotes the bandwidth expressed in Mbytes/second. 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:

$$t_{exec} = \frac{P \times (T \times N)}{B_w} \tag{12}$$

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 with 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. The leftmost part of Fig. 7 shows the estimated computing times considering various bandwidths (from $F = 10$ Mbytes/second to $F = 50$ Mbytes/second) and various numbers of processors ($P = 100$, $P = 200$ and $P = 400$). On the other hand, the rightmost part of Fig. 7 shows the speedups compared to a single-processor run of the compression algorithm in one Thunderhead node, again with a bandwidth ranging from 10 to 50 Mbytes/second and a systolic array with 100 , 200 and 400 processors. As it can be seen in Fig. 7, 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.

Table 3 shows a summary of resource utilization by the proposed systolic array-based implementation 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 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/second, 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, with

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

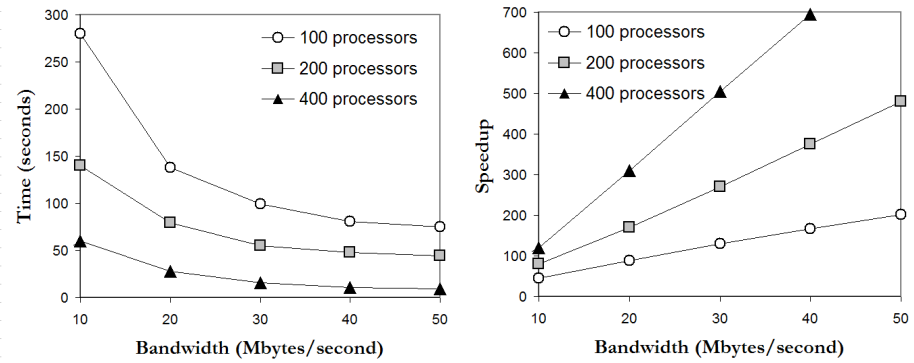

Figure 7. Time, in seconds, for the proposed hyperspectral data compression algorithm using a reconfigurable board connected to a PC through the I/O bus (left). Speedup compared to a single-processor version running on a single Thunderhead node (right).

Table 3. Summary of resource utilization for the FPGA-based implementation of the proposed hyperspectral data compression algorithm (operation frequency is given in MHz and processing time is given in seconds).

Number of processors	Total gates	Total slices	% of total	Operation frequency	Processing time
100	97,443	1,185	3%	29,257	69,91
200	212,412	3,587	10%	21,782	35,56
400	526,944	12,418	36%	18,032	20,48

a final measured processing time of about 20 seconds for $P = 400$ processors. This response, although not strictly in real-time, can still 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. [16] 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 by Table 3, 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).

5. CONCLUSIONS AND FUTURE RESEARCH

High performance compression of hyperspectral imagery is relevant topic in the remote sensing community. In particular, the wealth of spectral information provided by hyperspectral sensors is essential in many applications, and in many cases needs to be retained by compression algorithms with a high degree of fidelity. Standard 3D lossy compression techniques may cause significant loss of crucial information that is provided by mixed pixels and subpixel targets, which can now be uncovered in target detection or endmember extraction applications. This paper investigates the importance of mixed pixels in hyperspectral data compression and further proposes an innovative, application-oriented data compression technique which is based on two stages: endmember extraction followed by linear spectral unmixing. Experimental results in this work demonstrate that the proposed simple lossy compression algorithm can successfully retain the spectral information required to characterize mixed pixels and provide very high compression ratios with no apparent spectral signature degradation.

As shown by this paper, the extremely high computational requirements introduced by hyperspectral imaging applications (and the fact that these systems will continue increasing their spatial and spectral resolutions in the near future) make them an excellent case study to illustrate the need for high performance computing systems in remote sensing data compression applications. In this regard, one of the main purposes of this study has been to present current efforts towards the integration of the proposed lossy hyperspectral data compression technique with parallel and distributed computing practices, with the ultimate goal of designing a (near) real-time system. 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 near real-time data compression onboard, FPGAs may provide a solution which is scalable, compact in size and affordable in cost. Although the experimental results presented in 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 hyperspectral compression algorithm. In this context, we are currently developing real-time implementations of hyperspectral compression algorithms on commodity graphics hardware (GPUs),[18] which also represent an appealing type of high performance hardware architecture for onboard hyperspectral data compression.

ACKNOWLEDGMENTS

This research was supported in part by the European Commission through the Marie Curie Reseach Training Network project *Hyperspectral Imaging Network* (MRTN-CT-2006-035927). 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.

REFERENCES

1. G. Motta, F. Rizzo, and J. A. Storer, *Hyperspectral data compression*, Springer-Verlag, New York, 2006.
2. A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for earth remote sensing," *Science* **228**, pp. 1147–1153, 1985.
3. R. O. Green, "Imaging spectroscopy and the airborne visible-infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment* **65**, pp. 227–248, 1998.
4. J. A. Richards, "Analysis of remotely sensed data: the formative decades and the future," *IEEE Transactions on Geoscience and Remote Sensing* **43**, pp. 422–432, 2005.
5. D. S. Taubman and M. W. Marcellin, *JPEG2000: Image compression fundamentals, standard and practice*, Kluwer, Boston, 2002.
6. A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology* **6**, pp. 243–250, 1996.
7. B.-J. Kim, Z. Xiong, and W. A. Pearlman, "Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT)," *IEEE Transactions on Circuits and Systems for Video Technology* **10**, pp. 1374–1387, 2000.
8. C.-I. Chang, *Hyperspectral imaging: techniques for spectral detection and classification*, Kluwer Academic and Plenum Publishers, New York, 2003.
9. 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," *Journal of Geophysical Research* **91**, pp. 8098–8112, 1986.
10. Q. Du and C.-I. Chang, "Linear mixture analysis-based compression for hyperspectral image analysis," *IEEE Transactions on Geoscience and Remote Sensing* **42**, pp. 875–891, 2004.
11. D. Heinz and C.-I. Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing* **39**, pp. 529–545, 2001.
12. J. W. Boardman, "Automating spectral unmixing of aviris data using convex geometry concepts," in *Summaries of Airborne Earth Science Workshop*, R. O. Green, ed., *JPL Publication* **93-26**, pp. 111–114, 1993.
13. A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing* **42**, pp. 650–663, 2004.
14. A. Plaza and C.-I. Chang, *High performance computing in remote sensing*, CRC Press, Boca Raton, 2006.
15. A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing* **66**, pp. 345–358, 2006.
16. A. Plaza and C.-I. Chang, "Preface to the special issue on high performance computing for hyperspectral imaging," *International Journal of High Performance Computing Applications* **22**, 2008.
17. D. Lavernier, E. Fabiani, S. Derrien, and C. Wagner, "Systolic array for computing the pixel purity index algorithm on hyperspectral images," *Proceedings of SPIE* **4480**, pp. 130–138, 1999.
18. J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geoscience and Remote Sensing Letters* **43**, pp. 441–445, 2007.