Towards Real-Time Compression of Hyperspectral Images Using Virtex-II FPGAs

Antonio Plaza

Department of Computer Science, University of Extremadura
Avda. de la Universidad s/n, E-10071 Caceres, SPAIN
E-mail: aplaza@unex.es

Abstract. Hyperspectral imagery is a new type of high-dimensional image data which is now used in many Earth-based and planetary exploration applications. Many efforts have been devoted to designing and developing compression algorithms for hyperspectral imagery. Unfortunately, most available 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 an FPGA-based data 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 pixels in the image (endmembers) are first extracted and then used to express mixed pixels as linear combinations of end-members. The result is an intelligent, application-based compression technique which has been implemented and tested on a Xilinx Virtex-II FPGA.

1 Introduction

Due to significantly improved spectral resolution provided by latest-generation hyper-spectral imaging sensors, hyperspectral imagery expands the capability of multispectral imagery in many ways, such as subpixel target detection, object discrimination, mixed pixel classification and material quantification [1]. Each pixel in a hyperspectral image is composed of hundreds of reflectance values which define a 'spectral signature' for each pixel. By realizing the importance of hyperspectral data compression, many efforts have been devoted to designing and developing compression algorithms for hyperspectral imagery [2]. Two types of data compression can be performed, lossless and lossy, in accordance with redundancy removal. More specifically, lossless data compression is generally considered as data compaction which eliminates unnecessary redundancy without loss of information. By contrast, lossy data compression removes unwanted redundancy or insignificant in-formation which results in entropy reduction. Which type of compression should be used depends heavily upon the application under study. For example, in medical imaging, lossless compression is preferred. However, in this case only small compression ratios can be achieved (typically, 3:1

or below). On the other hand, video processing such as high definition television (HDTV) can greatly benefit from lossy compression. For remotely sensed imagery, both types of compression have been investigated in the past [2].

Our main focus in this work is to design compression techniques able to reduce significantly the large volume of information contained in hyperspectral data while, at the same time, being able to retain information that is crucial to deal with mixed pixels and subpixel targets. These two types of pixels above are essential in many hyperspectral analysis applications, including military target detection and tracking, environmental modeling and assessment at sub-pixel scales, etc. A subpixel target is a mixed pixel with size smaller than the available pixel size (spatial resolution) [3]. 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 [4]. In this case, spectral information can greatly help to effectively characterize the substances within the mixed pixel via spectral unmixing techniques [5]. 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 [6].

The possibility of real-time, onboard data compression is a highly desirable feature to overcome the problem of transmitting a sheer volume of high-dimensional data to Earth control stations via downlink connections. 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 towards onboard and real-time analysis of remote sensing data [7, 8]. 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 [9].

In this work, we explore a solution based on mapping the proposed compression algorithm on FPGA hardware. The remainder of the paper is organized as follows. Section 2 develops a new application-oriented lossy compression algorithm which utilizes a two-stage approach: first, a pixel purity index (PPI) algorithm is used to extract the purest pixels (endmembers) in the image, and then a linear spectral unmix-ing (LSU) procedure is used to express mixed pixels as linear combinations of endmembers, weighted by their respective abundance fractions. Section 3 maps the proposed compression algorithm in hardware using systolic array design. Section 4 provides experimental evidence about the algorithm performance using a real image data set collected by the NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS). Parallel performance data are given using a Xilinx Virtex-II FPGA. Finally, Section 5 concludes with some remarks and hints at plausible future research.

2 Hyperspectral Data Compression

2.1 Compression Algorithm

The idea of the proposed data compression algorithm is to represent a hyperspectral image cube by a set of fractional abundance images [10]. More precisely, for each N-dimensional pixel vector $\mathbf{f}_i$, its associated abundance vector $\mathbf{a}_i$ of E dimensions is used as a fingerprint of $\mathbf{f}$ with regards to E endmembers obtained by the pixel purity index (PPI) algorithm. The implementation of the proposed data compression algorithm can be summarized by the following steps:

1. Use the PPI algorithm to generate a set of E endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.
2. For each pixel vector $\mathbf{f}_i$ in the input scene, use the LSU algorithm to estimate the corresponding endmember abundance fractions $\mathbf{a}_i = \{a_{i1}, a_{i2}, \cdots, a_{iE}\}$ and approximate $\mathbf{f}_i = \mathbf{e}_1 \cdot a_{i1} + \mathbf{e}_2 \cdot a_{i2} + \cdots + \mathbf{e}_E \cdot a_{iE}$. Note that this is a reconstruction of $\mathbf{f}_i$.
3. Construct E fractional abundance images, one for each endmember.
4. Apply lossless predictive coding to reduce spatial redundancy within each of the E fractional abundance images, using Huffman coding to encode predictive errors.

2.2 Pixel Purity Index (PPI)

The PPI is a well-known approach to deal with the problem of mixed pixels in hyperspectral imaging. In this work, we use an improved version of the PPI algorithm [4] as the first step of our compression algorithm. Due to the algorithm's propriety and limited published results, we provide an outline of the algorithm which is based on limited published results and our own interpretation [3].

The PPI generates 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. 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 PPI 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 process; and a threshold angle, t_a, used to discard redundant endmembers. The output of the algorithm is a set of E final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$. The algorithm is summarized as follows:

1. Produce a set of K randomly generated unit vectors $\{\mathbf{skewer}_j\}_{j=1}^{K}$.

2. For each $\mathbf{skewer}_j$, all sample pixel vectors $\mathbf{f}_i$ in the original data set $\mathbf{F}$ are projected onto $\mathbf{skewer}_j$ via dot products of $|\mathbf{f}_i \cdot \mathbf{skewer}_j|$ to find sample

vectors at its extreme (maximum and minimum) projections, thus forming an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}(\mathbf{skewer}_j)$. 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:

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

3. Calculate the PPI score associated to the pixel vector $\mathbf{f}_i$ using the following equation:

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

4. Find the pixels with value of $N_{PPI}(\mathbf{f}_i)$ above t_v, and form a unique set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.

2.3 Linear Spectral Unmixing (LSU)

For each sample pixel vector $\mathbf{f}_i$ in $\mathbf{F}$, a set of abundance fractions specified by $\mathbf{a}_i = \{a_{i1}, a_{i2}, \cdots, a_{iE}\}$ 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:

$$\mathbf{f}_i = \mathbf{e}_1 \cdot a_{i1} + \mathbf{e}_2 \cdot a_{i2} + \cdots + \mathbf{e}_E \cdot a_{iE} \tag{3}$$

In order to achieve the decomposition above, we multiply each pixel $\mathbf{f}_i$ 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. 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 PPI.

3 FPGA-Based Hardware 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 de-sign [11]. 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 [12].

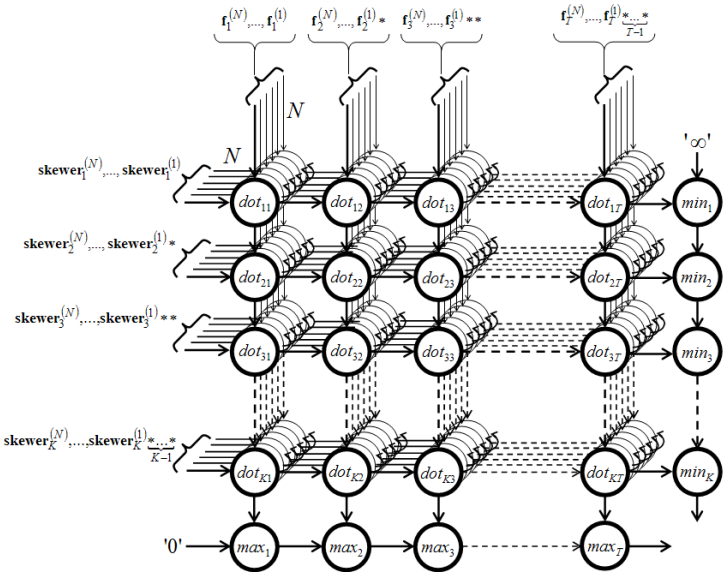

Fig. 1. Systolic array design for the proposed FPGA implementation of the PPI.

The rationale behind our systolic array-based parallelization can be summarized as follows. 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 and $i = \{1 \cdots T\}$, with T being the total number of pixels in the input scene. 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, then the PPI algorithm can be split into P passes, each performing dot-products, where T is the total number of input pixels to be fed to the systolic. From an architectural point of view, each processor receives T pixels, computes T dot-products, and keeps in memory the two pixels having produced 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. 1 illustrates the above principle, in which 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. 1, 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. 1 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 main advantage of the systolic array described in Fig. 1 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 order to reduce the number of passes, we decide to allocate the maximum number of processors in the available FPGA components. In other words, although in Fig. 1 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 thus 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. Finally, to obtain the vector of endmember abundances $\{a_{i1}, a_{i2}, \cdots, a_{iE}\}$ for each pixel $\mathbf{f}_i$, the multiplication of each $\mathbf{f}_i$ by $(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T$, where $\mathbf{M} = \{\mathbf{e}_e\}_{e=1}^{E}$, is done as described in [13], i.e. using a simple parallel block algorithm.

The algorithm described above was synthesized using Handel-C, a hardware design and prototyping language that allows using a pseudo-C programming style. The source code in Handel-C corresponding to step 2 of our FPGA implementation of the PPI algorithm is shown in Algorithm 1. The implementation was compiled and transformed into an EDIF specification automatically by using the DK3.1 software package. We also used other tools such as Xilinx ISE 6.1i[1] to carry out automatic place and route (PAR), and to adapt the final steps of the hardware implementation to the Virtex-II FPGA used in experiments.

4 Experimental Results

This section provides an assessment of the effectiveness of the hardware-based compression algorithm described in sections 2 and 3. The algorithm was implemented on a Virtex-II XC2V6000-6 FPGA, which contains 33,792 slices, 144 Select RAM Blocks and 144 multipliers (of 18-bit x 18-bit). The algorithm was applied to a real hyperspectral scene collected by an AVIRIS flight over the Cuprite mining district in Nevada, which consists of 614×512 pixels and 224 bands. The site has several exposed minerals of interest. Fig. 2(left) shows a spectral band of the image, and Fig. 2(right) plots the spectra of five minerals measured in the field by U.S. Geological Survey (USGS).

In order to explore the quality of the compressed images produced by the proposed compression method, Table 1 reports the spectral angle similarity scores [1, 3] among the USGS reference signatures in Fig. 2 and the PPI-extracted endmembers from the resulting images after data compression (the lowest the scores,

[1] http://www.xilinx.com

Algorithm 1 Handel-C implementation of the PPI for FPGAs.

```
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];
```

Table 1. Spectral similarity scores among USGS spectra and endmembers extracted from the original image, and from several reconstructed versions of the image after applying PPI/LSU, JPEG2000 and SPIHT algorithms with different compression ratios.

Mineral	**Original**	**PPI/LSU:**			**JPEG2000:**			**SPIHT:**		
signature	**image**	20:1	40:1	80:1	20:1	40:1	80:1	20:1	40:1	80:1
Alunite	0.063	0.069	0.078	0.085	0.112	0.123	0.133	0.106	0.119	0.129
Buddingtonite	0.042	0.053	0.061	0.068	0.105	0.131	0.142	0.102	0.125	0.127
Calcite	0.055	0.057	0.063	0.074	0.102	0.128	0.139	0.097	0.122	0.134
Kaolinite	0.054	0.059	0.062	0.071	0.114	0.140	0.151	0.110	0.134	0.146
Muscovite	0.067	0.074	0.082	0.089	0.123	0.145	0.167	0.116	0.139	0.152

the highest the similarity), using compression ratios of 20:1, 40:1 and 80:1 (given by different tested values of input parameter E). Spectral similarity scores below 0.1 are widely considered as a requirement in many applications [4].

As expected, the highest-quality endmembers were extracted from the original data set. As the compression ratio was increased, the quality of extracted endmembers was decreased. For illustrative purposes, we have also included the results provided by two standard methods in our comparison, i.e., the wavelet-based JPEG2000 multi-component [14] and the set partitioning in hierarchical trees (SPIHT) [15]. The JPEG2000 implementation used for our experiments was the one available in kakadu software [2]. Both techniques are 3-D compression algorithms that treat the hyperspectral data as a 3-D volume, where the spectral information is the third dimension. Results in Table 1 show that such 3-D techniques, which enjoy great success in classical image processing, may not necessarily find equal success in hyperspectral image compression. Specifically, techniques able to preserve the spectral information are required to characterize mixed pixels and subpixel targets. As demonstrated by Table 1, for the same compression ratio, a 3-D lossy compression may result in significant loss of spec-

[2] http://www.kakadusoftware.com

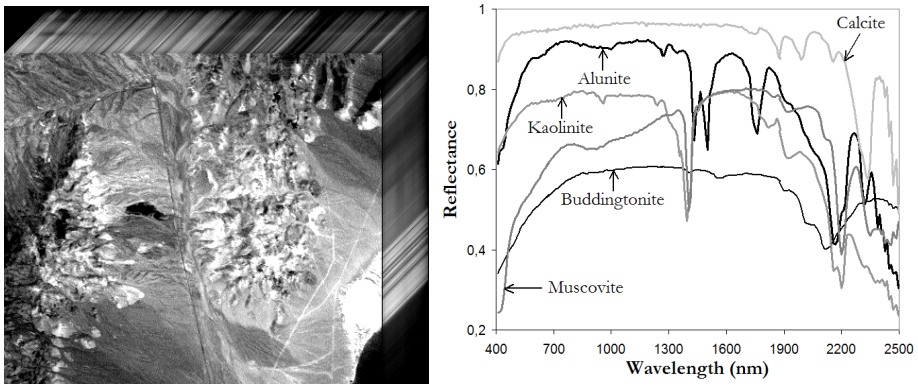

Fig. 2. AVIRIS hyperspectral image (left) and USGS mineral signatures (right).

tral information which can be preserved much better, in turn, by an application-oriented algorithm such as the proposed PPI/LSU. It should be noted that the serial versions of the algorithms in Table 1 required several minutes of computation to compress the AVIRIS Cuprite data set in a PC with AMD Athlon 2.6 GHz processor and 512 MB of RAM (specifically, the PPI/LSU algorithm required almost one hour).

The average performance of the systolic array is mainly determined by the dot-product capacity, that is the number of additions/subtractions executed in one second. Fig. 3(left) shows the estimated computing times considering various bandwidths (from $B_w = 10$ to $B_w = 50$ Mbytes/second) and various numbers of processors ($P = 100$, $P = 200$ and $P = 400$). On the other hand, Fig. 3(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/second and a systolic array with 100, 200 and 400 processors. As shown by Fig. 3, theoretical speedups can be very high.

In order to validate the estimations in Fig. 3 on a real FPGA architecture, Table 2 shows a summary of resource utilization by the proposed systolic array-based implementation of the PPI/LSU compression algorithm on a complete system (systolic array plus PCI interface), implemented on a Xilinx 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 speedup of 120 when running the PPI/LSU with a maximum number of $P = 400$ processors. The optimum trade-off between the achieveable parallelism versus clockrate was found for the maximum number of processors used since the balance between the speedup found and the operation frequency (around 18 MHz) was satisfactory while at the same leaving enough room in the FPGA for implementation of additional algorithms (only 36% of the FPGA resources were used).

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

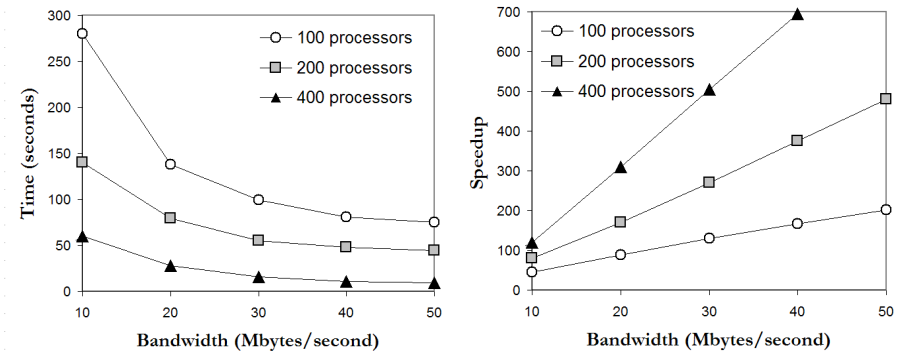

Fig. 3. Time in seconds for computing the full hyperspectral data compression algorithm using a reconfigurable board connected to a PC through the I/O bus (left) and speedup compared to a single-processor version running on a single AMD PC (right).

Table 2. Resource utilization for the FPGA implementation (operation frequency given in MHz).

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

can only provide a flow of 15 Mbytes. This experiment, however, demonstrated that the considered board, even with a non-optimized PCI connection (with no DMA), can still yield very good speedup for the PPI/LSU, with a final execution time for all the compression procedure of only 7.94 seconds for $P = 400$ processors. This response is not strictly in real-time since the cross-track scan line in AVIRIS, a pushbroom instrument [1], is quite fast (8.3 msec). This introduces the need to process a full image cube (614×512 pixels with 224 bands) in no more than 5 seconds to fully achieve real-time performance. However, we anticipate that the proposed FPGA design can still be significantly optimized to fulfill real-time requirements (even without increasing the number of processors) by improving the communication bandwidth between the PC and the FPGA board, which seems feasible given the limited flow of the considered PCI interface.

5 Conclusions

The wealth of spectral information provided by hyperspectral sensors is essential in many applications, and needs to be retained by compression algorithms. Standard 3-D lossy compression techniques may cause significant loss of crucial information that is provided by mixed pixels and subpixel targets, which are essential in hyperspectral imaing applications. In order to satisfy (near) real-time requirements, we have developed an FPGA-based algorithm for onboard data compression. A major goal is to overcome 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 applications. It should be noted that efficient onboard compression 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. Although the experimental results presented in this paper are encouraging, further work is still needed to arrive to optimal parallel design and implementations for the proposed and other hyperspectral compression algorithms.

References

1. Chang, C.-I.: Hyperspectral imaging: Detection & classification. Kluwer, 2003.
2. Motta, G., Rizzo, F., Storer, J. A.: Hyperspectral data compression. Springer-Verlag: New York, 2005.
3. Plaza, A., Chang, C.-I.: Impact of initialization on design of endmember extraction algorithms. IEEE Trans. Geoscience and Remote Sensing **44** (2006) 3397-3407.
4. Plaza, A., Martinez, P., Perez, R., Plaza, J.: A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data. IEEE Trans. Geoscience and Remote Sensing **42** (2004) 650-663.
5. Chang, C.-I., Plaza, A.: A Fast Iterative Implementation of the Pixel Purity Index Algorithm. IEEE Geoscience and Remote Sensing Letters **3** (2006) 63-67.
6. Du, J., Chang, C.-I.: Linear Mixture Analysis-Based Compression for Hyperspectral Image Analysis. IEEE Trans. Geoscience and Remote Sensing **42** (2004) 875-891.
7. El-Araby, E., El-Ghazawi, T., Le Moigne, J.: Wavelet spectral dimension reduction of hyperspectral imagery on a reconfigurable computer. Proc. of the 4th IEEE International Conference on Field-Programmable Technology **1** (2004) 861-867.
8. Fry, T., Hauck, S.: Hyperspectral image compression on reconfigurable platforms. Proc. of the 10th IEEE Symposium on Field-Programmable Custom Computing Machines **1** (2002) 305-312.
9. Plaza, A., Valencia, D., Plaza, J., Martinez, P.: Commodity cluster-based parallel processing of hyperspectral imagery. Journal of Parallel and Distributed Computing **66** (2006) 345-358.
10. Ramakhrishna, B., Plaza, A., Chang, C.-I., Ren, H., Du, Q., Chang, C.-C.: Spectral/spatial hyperspectral image compression. Chapter 11 in: Hyperspectral data compression. G. Motta, F. Rizzo, J. A. Storer, Eds. (2005) 309-346.
11. Valero-Garcia, M., Navarro, J., Llaberia, J., Valero, M., Lang, T.: A method for implementation of one-dimensional systolic algorithms with data contraflow using pipelined functional units. Journal of VLSI Signal Processing **4** (1992) 7-25.
12. Zhang, D., Pal, S. K.: Neural Nets & Systolic Array Design. World Scientific, 2002.
13. Dou, Y., Vassiliadis, S., Kuzmanov, G., Gaydadjiev, G.: 64-bit floating-point FPGA matrix multiplication. Proc. of the 13th ACM/SIGDA International Symposium on FPGAs **1** (2005) 123-129.
14. Taubman, D.S., Marcellin, M.W.: JPEG2000: Image Compression Fundamentals, Standard and Practice. Kluwer: Boston, 2002.
15. Said, A., Pearlman, W.A.: A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. IEEE Transactions on Circuits and Systems **6** (1996) 243-350.