Commodity cluster and hardware-based massively parallel implementations of hyperspectral imaging algorithms

Antonio Plaza[a], Chein-I Chang[b], Javier Plaza [a], David Valencia[a]

[a]Computer Science Department, University of Extremadura
Avda. de la Universidad s/n, E-10071 Cáceres, Spain

[b]Remote Sensing Signal and Image Processing Laboratory
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County, Baltimore, MD 21250

ABSTRACT

The incorporation of hyperspectral sensors aboard airborne/satellite platforms is currently producing a nearly continual stream of multidimensional image data, and this high data volume has soon introduced new processing challenges. The price paid for the wealth spatial and spectral information available from hyperspectral sensors is the enormous amounts of data that they generate. Several applications exist, however, where having the desired information calculated quickly enough for practical use is highly desirable. High computing performance of algorithm analysis is particularly important in homeland defense and security applications, in which swift decisions often involve detection of (sub-pixel) military targets (including hostile weaponry, camouflage, concealment, and decoys) or chemical/biological agents. In order to speed-up computational performance of hyperspectral imaging algorithms, this paper develops several fast parallel data processing techniques. Techniques include four classes of algorithms: (1) unsupervised classification, (2) spectral unmixing, and (3) automatic target recognition, and (4) onboard data compression. A massively parallel Beowulf cluster (Thunderhead) at NASA's Goddard Space Flight Center in Maryland is used to measure parallel performance of the proposed algorithms. In order to explore the viability of developing onboard, real-time hyperspectral data compression algorithms, a Xilinx Virtex-II field programmable gate array (FPGA) is also used in experiments. Our quantitative and comparative assessment of parallel techniques and strategies may help image analysts in selection of parallel hyperspectral algorithms for specific applications.

Keywords: Hyperspectral imaging, parallel computing, commodity clusters, field programmable gate arrays.

1. INTRODUCTION

The development of computationally efficient techniques for transforming the massive amount of hyperspectral data collected on a daily basis into scientific understanding is critical for space-based Earth science and planetary exploration[1]. In particular, the wealth of spatial and spectral information provided by last-generation hyperspectral instruments has opened ground-breaking perspectives in many applications. Examples include environmental modeling and assessment, target detection for military and homeland defense/security purposes, urban planning and management studies, risk/hazard prevention and response including wild-land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination. Most of the above-cited applications are based on swift decisions which depend on high computing performance of algorithm analysis. However, near real-time performance is a very ambitious goal since the price paid for the rich information available from hyperspectral sensors is the enormous amounts of data that they generate.

In recent years, several efforts have been directed towards the incorporation of high-performance computing models in remote sensing applications[2]. With the aim of creating a cost-effective parallel computing system from commodity components to satisfy specific computational requirements for the Earth and space sciences community, the Center of Excellence in Space and Data Information Sciences (CESDIS), located at the NASA's Goddard Space Flight Center in Maryland, developed the concept of Beowulf cluster[3]. The purpose of commodity cluster computing is to maximize the performance-to-cost ratio of computing by using low-cost commodity components and free-source, Linux-type operating

Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XII,
edited by Sylvia S. Shen, Paul E. Lewis, Proc. of SPIE Vol. 6233,
623316, (2006) · 0277-786X/06/$15 · doi: 10.1117/12.665464

systems and GNU software to assemble a system with computational speed in the order of hundreds of Gflops and memory capacity sufficient for storing and processing large data sets. The processing power offered by such commodity cluster-based systems has been traditionally employed to tackle information extraction and mining from large data archives, containing high-dimensional imagery collected and transmitted to Earth. The possibility of real-time, onboard data compression is also a highly desirable feature to overcome the problem of transmitting a sheer volume of high-dimensional data to Earth control stations via downlink connections. For that purpose, specialized processing units such as field-programmable gate arrays (FPGAs)[4] can be readily employed.

Despite the growing interest in hyperspectral imaging, only a few research efforts devoted to the design of parallel implementations exist in the open literature. Most of them are only partially available, and subject in many cases to non-disclosure restrictions. However, with the recent explosion in the amount of hyperspectral imagery, parallel processing has become a requirement in virtually every remote sensing application. Further, the size, volume and dimensionality of image data in such applications are ever growing. As a result, this paper takes a necessary first step towards the comparison of highly innovative techniques and strategies for implementation of hyperspectral imaging algorithms on commodity cluster- and hardware-based architectures. The paper is organized as follows. Section 2 briefly describes data partitioning strategies for the considered problem. Sections 3 to 6 introduce several parallel hyperspectral imaging algorithms. As representative case studies, this paper focuses on four highly representative classes of hyperspectral methods, i.e., unsupervised classification, spectral unmixing, automatic target detection, and data compression. Section 7 provides a detailed survey on the scalability and parallel performance of the algorithms on two high-performance computing platforms: (1) Thunderhead, a Beowulf cluster made up of hundreds of CPUs and available at NASA's Goddard Space Flight Center, and (2) A Xilinx Virtex-II FPGA formed by several millions of gates, and with high computational power and compact size, which make this reconfigurable device very appealing for onboard, real-time data compression. Finally, section 5 concludes the paper with several remarks and hints at plausible future research.

2. DATA PARTITIONING STRATEGIES

Two types of data partitioning strategies can be adopted in the design of hyperspectral image analysis algorithms, namely, *spatial*-domain parallelism and *spectral*-domain parallelism. Spectral-domain parallelism subdivides the whole multi-band data into blocks made up of contiguous spectral bands (sub-volumes), and assigns one or more sub-volumes to each processing unit. On the other hand, spatial-domain parallelism subdivides the image into multiple blocks made up of entire pixel vectors, and assigns one or more blocks to each processing unit. It should be noted that the former approach breaks the spectral identity of the data because each pixel vector is split amongst several processing elements. Alternatively, hybrid approaches can also be adopted in which both spatial and spectral domain partitioning are used.

While the selection of the most appropriate decomposition framework is clearly application-dependent, it is worth noting that most available hyperspectral imaging algorithms treat the rich spectral information available from hyperspectral image cubes *as a whole*, i.e., they take advantage of the very fine spectral detail provided by hyperspectral sensors to accurately characterize observed objects and substances. With the above considerations in mind, a spatial-domain decomposition partitioner (SDP) has been developed in our application. This module partitions the data so that each pixel vector is never partitioned among different processing units. As a result, no partitioning of data is accomplished in the spectral domain, thus preserving the entire spectral signature of each hyperspectral image pixel. There are several reasons that justify the above decision[5]. First, the application of spatial-domain partitioning is a natural approach for low level image processing, as many operations require the same function to be applied to a small set of elements around each data element present in the image data structure. A second reason has to do with the cost of inter-processor communication. In spectral-domain parallelism, the calculations made for each hyperspectral pixel need to originate from several processing units, and thus require intensive inter-processor communication as shown in previous work[5].

3. PARALLEL ALGORITHMS FOR FULL-PIXEL CLASSIFICATION

Having the spatial-domain data decomposition framework above in mind, this section reports two parallel algorithms for unsupervised classification of hyperspectral image data. The first one uses the principal component transform (PCT)[6] to summarize and decorrelate the information in hyperspectral images by reducing redundancy and packing the residual information into a small set of images, termed principal components. The second one is a parallel version of the ISODATA classification procedure[7], regarded as the benchmark for all unsupervised classification algorithms.

3.1. Parallel spectral-screening principal component transform algorithm (P-PCT)

PCT is a highly compute-intensive algorithm amenable to parallel implementation. In order to speed performance up, the P-PCT algorithm uses a standard master-slave decomposition technique, where the master coordinates the actions of the workers, gathers the partial results from them and provides the final result. Here, the original parallel algorithm[6] is modified to include the spectral angle mapper (SAM)[1] as the similarity criterion used by the algorithm.

P-PCT algorithm

Inputs: N-Dimensional (N-D) image cube, $\boldsymbol{f}$, number of unique spectra, c.

Output: 2-D image with a classification label for each pixel $\boldsymbol{f}(x,y)$ in the original image.

1. Divide the original image cube $\boldsymbol{f}$ into K spatial-domain partitions using the SDP module, where K is the number of workers in the system. Send each partition, consisting of a set of pixel vectors, to a worker. Each worker proceeds concurrently to form a unique spectral set by calculating the spectral angle for all vector pairs as follows.

$$\mathrm{SAM}[\boldsymbol{f}(x,y),\boldsymbol{f}(x',y')]=\cos^{-1}\left(\frac{\boldsymbol{f}(x,y)\cdot\boldsymbol{f}(x',y')}{\|\boldsymbol{f}(x,y)\|\,\|\boldsymbol{f}(x',y')\|}\right)$$

2. The K unique sets are sent back to the master and combined, one pair at a time. Upon completion, there will be only one unique set left with p unique pixel vectors.
3. Calculate the N-D mean vector $\boldsymbol{m}$ concurrently, where each component is the average of the pixel values of each spectral band of the unique set. This vector is formed once all the processors finish their parts.
4. All the pixel vectors in the unique set are divided into K parts and sent to K workers. Each worker then computes the covariance component and forms a covariance sum.
5. Calculate the covariance matrix sequentially as the average of all the matrices calculated in step 4.
6. Obtain a transformation matrix $\boldsymbol{T}$ by calculating and sorting the eigenvectors of the covariance matrix according to their corresponding eigenvalues, which provide a measure of their variances. As a result, the spectral content is forced into the front components. Since the degree of data dependency of the calculation is high and its complexity is related to the number of spectral bands rather than the image size, this step is also done sequentially at the master.
7. Transform each pixel vector in the original hyperspectral image independently using $\boldsymbol{T}\cdot[\boldsymbol{f}(x,y)-\boldsymbol{m}]$. This step is done in parallel, where all workers transform their respective portions of data concurrently.
8. Finally, a parallel post-processing step is applied to perform classification at a pixel level in the PCT-transformed space. First, K partitions of a reduced, c-dimensional data cube given by the first PCT components are sent to the workers, along with the spatial locations of the p unique pixel vectors resulting from step 2. Each worker then labels each pixel in its corresponding partition with a class label given by the most spectrally similar unique pixel vector in the PCT-reduced space, and sends back the result to the master, which composes a final 2-D classification image.

3.2. Parallel ISODATA algorithm (P-ISODATA)

One of the main drawbacks of PCT-based techniques is that they rely on the statistical significance of the spectra, rather than the *uniqueness* of the spectra. As a result, small objects and ground features (which may be crucial in security/defense applications) would likely manifest themselves in the last principal components, thus being discarded prior to classification. In order to resolve this issue, the P-ISODATA algorithm[7] was designed as the first parallel approach able to deal with the entire high-dimensional volume directly, thereby preserving all the spectral information in the data. Here, we provide a modification of this parallel algorithm that makes use of the SAM as a similarity measure to cluster data elements into different classes. A master-slave parallel implementation of the algorithm can be summarized as follows.

P-ISODATA algorithm

Inputs: N-D image cube $\boldsymbol{f}$, number of clusters c, convergence threshold t.

Outputs: 2-D image with a classification label for each pixel $\boldsymbol{f}(x,y)$ in the original image.

1. Divide the original image cube $\boldsymbol{f}$ into K equally sized blocks such that there is no overlapping among different blocks. Send each partition, consisting of a set of pixel vectors, to a worker, along with a set of c randomly selected pixel vectors assumed to be initial centroids.

2. Each worker labels each pixel $\boldsymbol{f}(x,y)$ in the corresponding partition. Let us denote by n_{ij} the number of pixels belonging to the j-th cluster of the i-th worker, and by $\boldsymbol{f}_k^j(x,y)$ the k-th pixel of the j-th cluster. Then, to minimize inter-processor communication, each worker sends the number of pixels belonging to each cluster and the summation of feature vectors of all pixels belonging to each cluster, that is, $\boldsymbol{Y}_{ij} = \left[\sum_{\boldsymbol{k}=1}^{\boldsymbol{n_{ij}}} \boldsymbol{f}_k^1(x,y), \cdots, \sum_{\boldsymbol{k}=1}^{\boldsymbol{n_{ij}}} \boldsymbol{f}_k^c(x,y)\right]$.
3. The master collects all the information provided by the workers and combines it to obtain a new centroid for each cluster j using $\boldsymbol{c}_j = \left(\sum_{i=1}^{\mathrm{p}_j} \boldsymbol{Y}_{ij} \Big/ \sum_{i=1}^{\mathrm{p}_j} n_{ij}\right)$, where p_j is the number of pixels in the cluster.
4. The master compares the current centroids and the new centroids. If the difference between them is less than the convergence threshold *t*, then convergence occurs and the master informs all workers about the current status of convergence. Otherwise, steps 2-4 are repeated until the convergence status is true.
5. Following convergence, each worker i computes the summation of the Euclidean distance of all pixels within a cluster j from its centroid $\boldsymbol{c}_j$. Each worker then sends this information to the master, which combines it to obtain the deviation of pixels within each cluster j. It should be noted that this information is only exchanged when convergence has occurred (instead of doing so every time new centroids are calculated) in order to minimize inter-processor communication.
6. The master now decides about splitting or merging the resulting clusters, based on parameters such as the inter-cluster distances (separation), the intra-cluster distances (compactness), the ratio of standard deviations along different axes for each cluster, and the number of pixels per cluster. The above procedure is repeated until no cluster is further eligible for splitting or merging. When the stopping criterion is satisfied, each worker sends the label (cluster number) associated with each local pixel to the master, which combines all the individual results and forms the final 2-D image.

4. PARALLEL ALGORITHMS FOR AUTOMATIC TARGET DETECTION

Two unsupervised target detection algorithms for hyperspectral imagery are considered in this section. These are the automatic target generation process (ATGP)[1] and the unsupervised fully constrained least squares (UFCLS)[1] algorithm. These two algorithms extract targets in sequential fashion, so that the output is an ordered set of *t* spectrally disting targets which correspond to actual pixel vectors in the scene.

4.1. Parallel automatic target generation process (P-ATGP)

The ATGP algorithm is a relatively fast approach which finds a set of spectrally distinct target pixels vectors using of orthogonal subspace projections in the spectral domain. Below, we provide a step-by-step description of our parallel version of this algorithm.

P-ATGP algorithm

Inputs: N-D image cube $\boldsymbol{f}$, number of target pixel vectors to be detected *t*.

Output: Set of target pixel vectors, $\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \ldots, \mathbf{t}^{(t)}$.

1. Divide the original image cube $\boldsymbol{f}$ into *K* spatial-domain partitions using the SDP module, where *K* is the number of workers in the system. Send each partition, consisting of a set of pixel vectors, to a worker.
2. Each worker finds in parallel the brightest pixel at its local partition using $\mathbf{t}_i^{(1)} = \arg\left\{\max_{(x,y)} \boldsymbol{f}(x,y)^{\mathrm{T}} \cdot \boldsymbol{f}(x,y)\right\}$, where the superscript “T” denotes the vector transpose operation and $i = 1,2,\ldots,K$. Each worker then sends the spatial locations of the pixel identified as the brightest ones in its local partition back to the master.
3. Once all the workers have completed their parts, the master finds the brightest pixel of the input scene $\mathbf{t}^{(1)}$, where the superscript “(1)” indicates that $\mathbf{t}^{(1)}$ is the first pixel selected throughout the process, by applying the argmax operator in Step 2 using only the pixels at that the spatial locations provided by the workers and selecting the one that results in the maximum score. Then, the master sets $\mathrm{U} = \mathbf{t}^{(1)}$ to initialize the algorithm. At this point, it is worth noting that the brightest pixel is not the only possible selection for initialization of ATGP. However, according to

previous experiments in the literature[1], the brightest pixel is always extracted later on by the algorithm if it is not used as the initial pixel vector and therefore represents a reasonable choice to begin the process.

4. The master now broadcasts matrix $\mathbf{U}$ to all the workers, each of which works in parallel to find the pixel in the local partition with the maximum orthogonal projection relative to the pixel vectors in $\mathbf{U}$, using an orthogonal space projector given by the following expression: $P_{\mathbf{U}}^{\perp} = \mathbf{I} - \mathbf{U}\left(\mathbf{U}^{\mathrm{T}}\mathbf{U}\right)^{-1}\mathbf{U}^{\mathrm{T}}$, where $\mathbf{I}$ is the identity matrix. The orthogonal space projector $P_{\mathbf{U}}^{\perp}$ is now applied to all pixel vectors in each local partition to obtain $\mathbf{t}_i^{(2)} = \arg\left\{\max_{(x,y)}\left[\left(P_{\mathbf{U}}^{\perp}\boldsymbol{f}(x,y)\right)^{\mathrm{T}}\left(P_{\mathbf{U}}^{\perp}\boldsymbol{f}(x,y)\right)\right]\right\}$. Each worker then sends the spatial location of the resulting local pixels to the master node.
5. The master finds a second target pixel, $\mathbf{t}^{(2)}$, by applying $P_{\mathbf{U}}^{\perp}$ to the pixel vectors at the spatial locations provided by the workers, and selecting the one which results in the maximum score. The master then sets $\mathbf{U} = \left\{\mathbf{t}^{(1)}\ \mathbf{t}^{(2)}\right\}$.
6. Repeat from step 4 until a set of *t* unique pixel vectors, $\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \ldots, \mathbf{t}^{(t)}$, have been extracted.

4.2. Parallel unsupervised fully constrained least squares algorithm (P-UFCLS)

The UFCLS algorithm[1] generates a set of *t* targets using the concept of least square-based error (LSE) minimization. As opposed to ATGP, which assumes that target pixels are "pure," i.e., made up of a single underlying constituent material, the UFCLS incorporates the concept of mixed pixels into the target detection process. Our parallel version of UFCLS (called P-UFCLS) takes this issue into account and uses the following steps:

P-UFCLS algorithm

Inputs: N-D image cube $\boldsymbol{f}$, number of target pixel vectors to be detected *t*.

Output: Set of target pixel vectors, $\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \ldots, \mathbf{t}^{(t)}$.

1. Execute Steps 1-3 of P-ATGP to obtain the brightest pixel of the input scene $\mathbf{t}^{(1)}$, and broadcast $\mathbf{U} = \mathbf{t}^{(1)}$ to all workers.
2. Each worker forms a local error image $E^{(i)}$ by calculating the least squares-based error for each pixel vector in the input data represented in terms of a fully constrained linear mixture[1] of all the spectra in $\mathbf{U}$.
3. Each worker then finds the pixel $\boldsymbol{f}(x,y)$ in the local partition with the largest associated score in the error image $E^{(i)}$. The spatial coordinates of this pixel (and its associated error score) are sent back to the master.
4. The master now obtains a second target, $\mathbf{t}^{(2)}$, by selecting the pixel vector with the largest associated error score from the pixel vectors at the spatial locations provided by the workers. Then, the master broadcasts $\mathbf{U} = \left\{\mathbf{t}^{(1)}\ \mathbf{t}^{(2)}\right\}$ to all the workers.
5. Repeat from Step 4 and repeatedly incorporate a new target pixel $\mathbf{t}^{(2)}, \mathbf{t}^{(3)}, \ldots, \mathbf{t}^{(t)}$ to $\mathbf{U}$ until a set of *t* unique pixel vectors have been extracted.

5. PARALLEL ALGORITHMS FOR SPECTRAL UNMIXING

Spectral unmixing is a commonly used procedure in which the measured spectrum of a mixed pixel is decomposed into a collection of spectrally pure constituent spectra, or *endmembers*, and a set of correspondent fractions, or *abundances*, that indicate the proportion of each endmember in each pixel[8]. In this section, we describe several parallel techniques for endmember extraction and abundance estimation in hyperspectral image scenes. Two well-known algorithms for endmember extraction have been implemented: the pixel purity index (PPI) and the N-FINDR algorithm.

2.1. Parallel pixel purity index-like algorithm (P-PPI)

The PPI algorithm[9] first reduces the dimensionality of the input data and then proceeds by generating a large number of random, N-D 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 purest ones. A master-slave parallel version of the PPI algorithm, denoted by PPPI (P3I), is given below.

P-PPI algorithm
Input: N-D image cube $\boldsymbol{f}$, Number of skewers s, Number of endmembers, p, Threshold value t.

Output: set of endmembers $\{\mathbf{e}_i\}_{i=1}^{p}$.

1. Use steps 1-7 of the P-PCT algorithm above to reduce the dimensionality of the input data from N to p.
2. Generate a set of s randomly generated unit N-D vectors called "skewers," denoted by $\{\mathbf{skewer}_j\}_{j=1}^{s}$, and broadcast the entire set to all the workers.
3. For each skewer$_j$, project all the data sample vectors at each local partition K onto skewer$_j$ to find sample vectors at its extreme positions, and form an extrema set for skewer$_j$ denoted by $S^{(K)}(\mathbf{skewer}_j)$.
4. Define an indicator function of a set S by $I_S(\mathbf{x}) = \begin{cases} 1; \text{if } \mathbf{x} \in S \\ 0; \text{if } \mathbf{x} \notin S \end{cases}$, and use it to calculate $N_{PPI}^{(K)}[\boldsymbol{f}(x,y)] = \sum_j I_{S^{(K)}(\mathbf{skewer}_j)}[\boldsymbol{f}(x,y)]$ for each pixel $\boldsymbol{f}(x,y)$ at the local partition. Select those pixels with $N_{PPI}^{(K)}[\boldsymbol{f}(x,y)] > t$ and send them to the master.
5. The master collects all the partial results and merges them together to form a final set of endmembers $\{\mathbf{e}_i\}_{i=1}^{p}$.

2.2. Parallel N-FINDR-like algorithm (P-FINDR)

Winter's N-FINDR algorithm[10] identifies the set of pixels which define the simplex with the maximum volume, potentially inscribed within the dataset. After a dimensionality reduction step, a random set of pixel vectors is selected, and their corresponding volume is calculated. A trial volume is then calculated for every pixel in each endmember position by replacing that endmember and recalculating the volume. If the replacement results in a volume increase, the pixel replaces the endmember. This procedure is repeated until there are no replacements of endmembers left. A parallel version of this algorithm is given below.

P-FINDR algorithm
Input: N-D image cube $\boldsymbol{f}$, Number of endmembers, p

Output: set of final endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$.

1. Use steps 1-7 of the S-PCT algorithm above to reduce the dimensionality of the input data from N to p.
2. Select a random set of p initial pixels $\{\mathbf{e}_j^{(0)}\}_{j=1}^{p} \subseteq \{\mathbf{e}_i^{(0)}\}_{i=1}^{m}$ and find $V(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_p^{(0)})$, the volume of the simplex defined by $\{\mathbf{e}_j^{(0)}\}_{j=1}^{p}$, denoted by $S(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_p^{(0)})$.
3. Calculate the volume of p simplexes, $V(\boldsymbol{g}(x,y), \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_p^{(0)}), \ldots, V(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \boldsymbol{g}(x,y))$ in parallel each of which is formed by replacing one endmember $\mathbf{e}_j^{(0)}$ with the sample vector $\boldsymbol{g}(x,y)$. Each worker performs replacements using pixels in its local partition.
4. If none of these p recalculated volumes is greater than $V(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_p^{(0)})$, then no endmember in $\{\mathbf{e}_j^{(0)}\}_{j=1}^{p}$ is replaced. Otherwise, the master replaces the endmember which is absent in the largest volume among the p simplexes with the vector $\boldsymbol{g}(x,y)$. Let such endmember be denoted by $\mathbf{e}_j^{(0)}$. A new set of endmembers is then produced by letting $\mathbf{e}_j^{(1)} = \boldsymbol{g}(x,y)$ and $\mathbf{e}_i^{(1)} = \mathbf{e}_i^{(0)}$ for $i \neq j$.
5. Repeat from step 2 until no replacements occur. The final combination gives the final set of endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$.

2.3. Parallel linear spectral unmixing algorithm (P-LSU)

Once a set of spectral endmembers has been identified, an inversion model is required to estimate the fractional abundances of each of the endmembers at the mixed pixels. Here, we use a commonly adopted technique in the hyperspectral analysis literature, i.e., the LSU[1] technique[1]. It can be briefly described as follows. Suppose that there are p endmembers $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_p$ in a hyperspectral image scene, and let $\boldsymbol{f}(x,y)$ be a mixed pixel vector. LSU assumes that the spectral signature of $\boldsymbol{f}(x,y)$ can be represented by a linear mixture of $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_p$ with abundance fractions specified by $a_1, a_2, \ldots, a_p$. Then, we can model the spectral signature of an image pixel $\boldsymbol{f}(x,y)$ by a linear regression form $\boldsymbol{f}(x,y) = \mathbf{e}_1 \cdot a_1 + \mathbf{e}_2 \cdot a_2 + \cdots + \mathbf{e}_p \cdot a_p$. Two constraints should be imposed on this model to produce adequate solutions. These are the abundance sum-to-one constraint, that is, $\sum_{i=1}^{p} a_i = 1$, and the abundance non-negativity constraint, that is, $a_i \geq 0$ for $1 \leq i \leq p$. The process of LSU works on a pixel-by-pixel basis with no data dependencies involved, so the parallel implementation is simply given by the algorithm below.

P-LSU algorithm

Input: N-D data cube $\boldsymbol{f}$, Set of endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$.

Output: 2-D image with a classification label for each pixel $\boldsymbol{f}(x,y)$ in the original image.

1. Divide the original data cube $\boldsymbol{f}$ into K spatial-domain-based partitions, where K is the number of workers.
2. Broadcast the set $\{\mathbf{e}_j\}_{j=1}^{p}$ to all the workers.
3. For each pixel $\boldsymbol{f}(x,y)$ in the local partition, obtain a set of abundance fractions specified by $a_1(x,y), a_2(x,y), \ldots, a_p(x,y)$ using $\{\mathbf{e}_j\}_{j=1}^{p}$, so that $\boldsymbol{f}(x,y) = \mathbf{e}_1 \cdot a_1(x,y) + \mathbf{e}_2 \cdot a_2(x,y) + \cdots + \mathbf{e}_p \cdot a_p(x,y)$, taking into account the abundance sum-to-one and abundance non-negativity constraints.

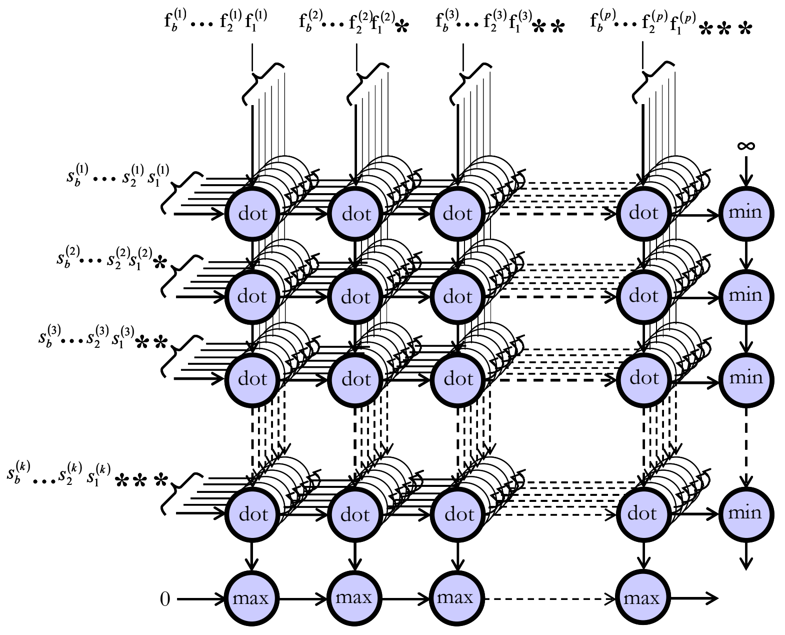
6. PARALLEL ALGORITHM FOR DATA COMPRESSION

Our main focus in this section is to design a compression technique 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 (in other words, we are interested in application-based compression rather than general-purpose compression). The two types of pixels above are essential in many hyperspectral analysis applications, including military target detection and tracking. When hyperspectral image compression is performed, it is critical and crucial to retain the above pixels, an issue that has been generally overlooked in the development of lossy compression techniques[11]. The idea of the proposed data compression algorithm is to represent a hyperspectral image cube by a set of p fractional abundance images. More precisely, for each N-D pixel vector $\boldsymbol{f}$, its associated set of p LSU-derived abundances $a_1(x,y), a_2(x,y), \ldots, a_p(x,y)$ is used as a fingerprint of $\boldsymbol{f}$ with regards to p endmembers obtained by the PPI algorithm. The implementation of the proposed P-PPI/P-LSU algorithm can be briefly summarized as follows:

P-PPI/P-LSU Algorithm

1. Use the P-PPI algorithm to generate a set of p endmembers $\{\mathbf{e}_i\}_{i=1}^{p}$.
2. For each pixel vector $\boldsymbol{f}(x,y)$, use the P-LSU algorithm to estimate the corresponding endmember abundance fractions $a_1(x,y), a_2(x,y), \ldots, a_p(x,y)$ and approximate $\boldsymbol{f}(x,y) = \mathbf{e}_1 \cdot a_1(x,y) + \mathbf{e}_2 \cdot a_2(x,y) + \cdots + \mathbf{e}_p \cdot a_p(x,y)$. Note that this is a reconstruction of $\boldsymbol{f}(x,y)$.
3. Construct p fractional abundance images, one for each P-PPI-derived endmember.
4. Apply lossless predictive coding to reduce spatial redundancy within each of the p fractional abundance images, using Huffman coding to encode predictive errors.

The algorithm above has been implemented in hardware using a systolic array-based approach. Fig. 1 shows our proposed systolic array design for FPGA implementation of the P-PPI algorithm, 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. The nodes labeled as "dot" in Fig. 1 perform the individual products for the skewer projections, while the nodes labeled as "max" and "min" respectively compute the maxima and minima projections after the dot product calculations have been completed (asterisks in Fig. 1 represent delays). In Fig. 1, $s_j^{(i)}$ denotes the j-th value of the i-th skewer vector, and $f_j^{(m)}$ denotes the reflectance value of the j-th band of the m-th pixel. Part of the systolic array design outlined above can also be employed to carry out the P-LSU-based inversion process. To obtain the abundance fractions $a_1(x,y), a_2(x,y), \ldots, a_p(x,y)$ for each pixel vector $\boldsymbol{f}(x,y)$, we need to multiply $\boldsymbol{f}(x,y)$ by $(\mathbf{M}^{\mathbf{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathbf{T}}$, where $\mathbf{M} = \{\mathbf{e}_i\}_{i=1}^{p}$ and the superscript "T" denotes the matrix transpose operation. We emphasize that we are still working towards the implementation of this step in hardware. Therefore, timing results in this work are given for the optimization introduced by P-PPI-based FPGA design.



$f_b^{(1)} \cdots f_2^{(1)} f_1^{(1)}$
$f_b^{(2)} \cdots f_2^{(2)} f_1^{(2)} *$
$f_b^{(3)} \cdots f_2^{(3)} f_1^{(3)} **$
$f_b^{(p)} \cdots f_2^{(p)} f_1^{(p)} ***$
∞
$s_b^{(1)} \cdots s_2^{(1)} s_1^{(1)}$
$s_b^{(2)} \cdots s_2^{(2)} s_1^{(2)} *$
$s_b^{(3)} \cdots s_2^{(3)} s_1^{(3)} **$
$s_b^{(k)} \cdots s_2^{(k)} s_1^{(k)} ***$
dot
min
max
0

Fig. 1. Systolic array design for the proposed FPGA implementation of the data compression algorithm.

7. EXPERIMENTAL RESULTS

This section provides an assessment of the effectiveness of the proposed parallel algorithms in providing significant performance gains without loss of accuracy in the analysis of real hyperspectral data sets. The section is organized as follows. First, we provide an overview of the parallel computing architectures used in experiments. Second, we describe the hyperspectral data sets used for demonstration purposes. Finally, a detailed computational cost-performance analysis of our proposed parallel algorithms is given in the context of realistic applications supported by hyperspectral imagery.

7.1. Parallel computing platforms

Two high-performance computing platforms have been used in experiments. In order to test the performance of the proposed data compression algorithm, mainly designed for onboard operation and processing, we have used 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 synthesis in the FPGA board was performed using Handel-C (http://www.celoxica.com), a hardware design and prototyping language that allows using a pseudo-C programming style. 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 to adapt the final implementation to the Virtex-II FPGA used in experiments.

The remaining parallel techniques developed in this work have been tested on Thunderhead, a Beowulf cluster at NASA's Goddard Space Flight Center. The Thunderhead system can be seen as an evolution of the HIVE (Highly Parallel Virtual Environment) project[3], started in spring of 1997 to build a commodity cluster intended to be exploited by different users in a wide range of scientific applications. The idea was to have workstations distributed among many offices and a large number of compute nodes (the compute core) concentrated in one area. The workstations would share the compute core as though it was apart of each. The HIVE was also the first commodity cluster to exceed a sustained 10 Gflops on an algorithm. Thunderhead is now composed of 256 dual 2.4 Ghz Intel Xeon nodes, each with 1 GB of memory and 80 GB of main memory, interconnected via 2 GHz optical fibre Myrinet. Our parallel algorithm was run from one of such nodes, called thunder1. The operating system used at the time of experiments was Linux RedHat 8.0, and MPICH was the message-passing library used.

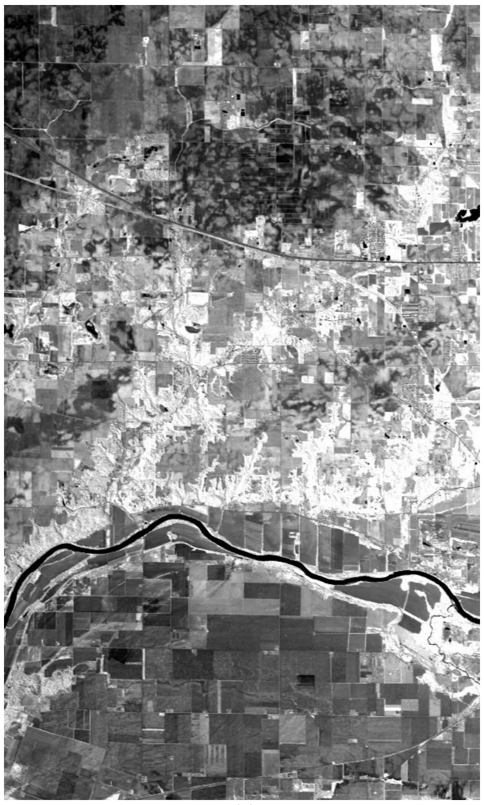
7.2. Hyperspectral data sets

Three well-known hyperspectral data sets have been used in experiments:

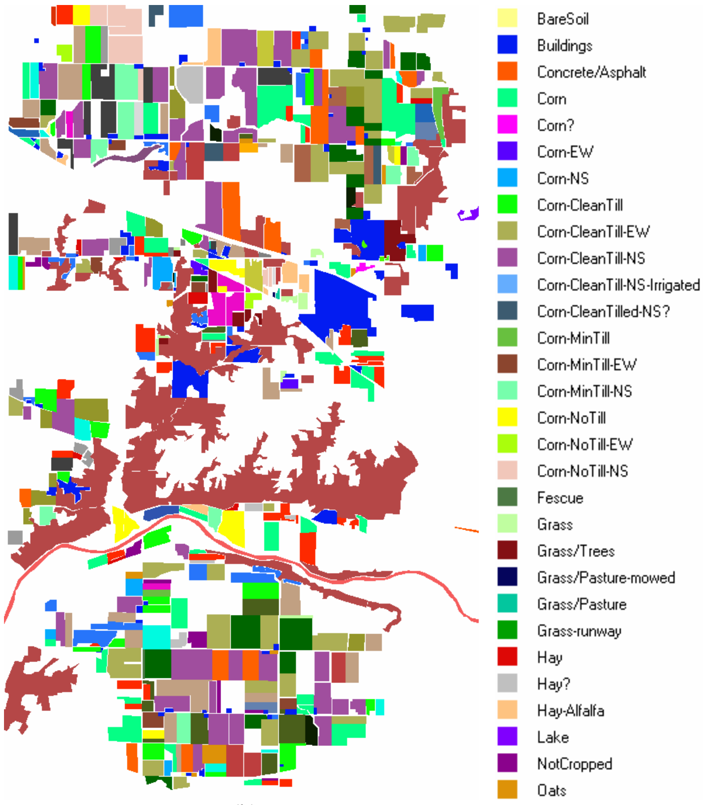
- In order to test the performance of parallel classification algorithms, a hyperspectral data set collected by the NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) over the Indian Pines test site in Northwestern Indiana. The scene, characterized by its very high spectral resolution (224 spectral bands in the range 0.4–2.5 μm) and moderate spatial resolution (20-meter pixels), covers a wide area (2166x614 pixels) and represents a very challenging classification problem, a fact that has made this scene a universal and extensively used benchmark to validate the accuracy of classification algorithms. Fig. 2(a) shows the spectral band at 587 nm and Fig. 2(b) shows the ground-truth map, in the form of a class assignment for each pixel with 30 ground-truth classes.
- A second AVIRIS data set was considered to test the performance of spectral unmixing and data compression algorithms. This scene corresponds to the well-known Cuprite mining district in Nevada, and is available online from http://aviris.jpl.nasa.gov/html/aviris.freedata.html. The scene comprises a relatively large area (614x512 pixels with spatial resolution of 20 meters) and 224 spectral bands between 0.4 and 2.5 μm, with nominal spectral resolution of 10 nm. The site is well understood mineralogically, and has several exposed minerals of interest including alunite, buddingtonite, calcite, kaolinite and muscovite. Reference ground signatures of the above minerals, available in the form of a U.S. Geological Survey (USGS) library (http://speclab.cr.usgs.gov/spectral-lib.html) will be used to assess signature purity in this work (reflectance data was used to allow such comparison).
- Finally, a hyperspectral scene collected in the framework of the HYperspectral Data Image Collection Experiment (HYDICE) will be used to assess parallel performance of target detection algorithms. The portion selected for experiments, which has a size of 64x64 pixels with 210 spectral bands in the wavelength region from 0.4 to 2.5 μm (and nominal spectral resolution of 10 nm), contains a large grass field background, a forest on the left edge, and 15 man-made targets (panels) located in the center of the grass field and arranged in the form of a 5x3 matrix. For each row, there are three panels p_{i1}, p_{i2}, p_{i3}, painted by the same material but with three different sizes. For each column, there are also 5 panels p_{1j}, p_{2j}, p_{3j}, p_{4j}, p_{5j} which have the same size but are painted using different materials. The sizes of the panels in the first, second and third columns are 3x3 meters, 2x2 meters, and 1 meter, respectively. Since the size of the panels in the third column is only 1 meter, they cannot be seen visually due to the fact that their size is smaller than the data 1.56-meter spatial resolution. Fig. 2(c) shows the area of interest selected for experiments, including the spatial locations of 15 targets, where red pixels are used to denote panel center pixels and yellow pixels denote panel pixels which are mixed with the background.

7.3. Performance evaluation

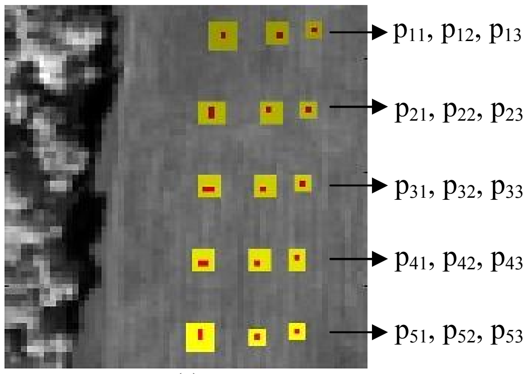
This section provides a performance assessment of the different parallel algorithms. Table 1(a) reports the overall classification accuracy scores produced by the considered P-PCT and P-ISODATA algorithms on the Indian Pines AVIRIS scene. Single-processor processing times (measured on a single Thunderhead node) are also given for each considered algorithm (in the parentheses). As shown by Table 1(a), the P-PCT produced higher classification scores than those found by P-ISODATA. To empirically investigate the scaling properties of the two considered parallel classification algorithms, Tables 1(b) and 1(c) respectively show the processing times and speedup factors as a function of the number of processors on Thunderhead (only 256 processors were available to us at the time of experiments). Overall, results in Table 1(c) reveal that the performance drop from linear speedup in both P-PCT and P-ISODATA algorithms increases significantly as the number of processors increase. This is due to data dependencies and sequential calculations present in the algorithmss, i.e., steps 5 and 6 (covariance matrix and eigenvector calculations) in the P-PCT, and step 6 (split and merge) in the P-ISODATA. It should be noted that the complexity of eigenvector calculations is related to the number of spectra, while the split and merge depends on the inherent complexity of the AVIRIS data.



(a)



(b)



(c)

Figure 2. (a) Spectral band at 587 nm wavelength of an AVIRIS scene collected over Indian Pines test site in Northwestern Indiana. (b) Ground-truth map for the Indian Pines scene with 30 mutually exclusive land-cover classes. (c) HYDICE image scene with spatial locations of 15 man-made targets (red pixels are panel centers and yellow pixels are mixed with the background).

	P-PCT	**P-ISODATA**
Classification accuracy (%)	82.25	69.84
Processing time (seconds)	41239	49912

(a)

	4	**16**	**36**	**64**	**100**	**144**	**196**	**256**
P-PCT	13521	4314	1759	884	572	392	314	265
P-ISODATA	21330	5907	2428	1299	865	630	444	386

(b)

	4	**16**	**36**	**64**	**100**	**144**	**196**	**256**
P-PCT	3.05	9.56	23.45	46.67	72.12	105.23	131.23	155.45
P-ISODATA	2.34	8.45	20.56	38.43	57.67	79.23	112.34	129.21

(c)

Table 1. (a) Classification accuracies and single-processing times (seconds) for the parallel classification algorithms (AVIRIS Indian Pines scene). (b) Parallel processing times (in seconds) on Thunderhead, using different numbers of processors. (c) Speedup factors.

	A	B	C	K	M
P-PPI (2745)	0.084	0.106	0.105	0.136	0.108
P-FINDR (695)	0.094	0.052	0.065	0.105	0.098

(a)

	4	16	36	64	100	144	196	256
P-PPI	1013	251	99	52	34	24	18	15
P-FINDR	199	46	24	13	9	6	5	4

(b)

	4	16	36	64	100	144	196	256
P-PPI	2.71	10.92	27.63	52.93	81.23	116.23	149.90	183.23
P-FINDR	3.50	15.23	28.45	54.67	79.34	109.23	135.67	156.78
P-LSU	3.75	15.39	35.21	63.01	97.23	139.23	186.75	240.34

(c)

Table 2. (a) Endmember extraction accuracy and single-processing times (in seconds, in the parentheses) for the parallel endmember algorithms (AVIRIS Cuprite scene). (b) Parallel processing times (in seconds) on Thunderhead. (c) Speedup factors.

On the other hand, Table 2(a) shows an experiment-based cross-examination of endmember extraction accuracy by the P-PPI and P-FINDR algorithms. The table tabulates the SAM scores obtained after comparing five USGS library spectra, labeled by "A" for alunite, "B" for buddingtonite, "C" for calcite, "K" for kaolinite and "M" for muscovite, with the corresponding endmembers extracted by two parallel algorithms above from the AVIRIS Cuprite scene. The smaller the spectral angle values across the five minerals considered in Table 2(a), the better the results. The total number of endmembers to be extracted, *p*, was set to 16 for all parallel methods after using the virtual dimensionality (VD) concept[1]. For the P-PPI, parameter *t* was set to the mean of N_{PPI} scores after $s = 10^3$ skewer iterations. For illustrative purposes, Tables 2(b) and 2(c) respectively report the parallel processing times and speedups achieved by multi-processor runs of the parallel algorithms over the corresponding single-processor runs in Table 2(a). As Table 2(c) shows, the scalability of P-PPI and P-FINDR was degraded as the number of processors was higher. This is mainly due to the fact that these algorithms involve operations that need to be completed sequentially at the master before distributing the results to the workers (e.g., initial volume estimation in the P-FINDR), or after all the workers have completed their parts (e.g., final endmember selection in the P-PPI). Quite opposite, the P-LSU provided much better scalability, which is not surprising given the very straightforward parallelization of LSU which involves almost no data dependences or sequential computations at the master node. Further, the P-LSU involves only minimal communication between the parallel tasks, namely, at the beginning and ending of such tasks.

	p_{11}	p_{21}	p_{31}	p_{41}	p_{51}
P-ATGP (36)	6th	17th	5th	18th	3rd
P-UFCLS (972)	9th	-	5th	-	4th

(a)

	4	16	36	64	100	144	196	256
P-ATGP	10.59	3.21	1.47	0.74	0.48	0.35	0.28	0.24
P-UFCLS	312	95	34	17	12	8	6	5

(b)

	4	16	36	64	100	144	196	256
P-ATGP	3.4	11.23	24.56	48.76	75.23	103.45	128.79	152.34
P-UFCLS	3.12	10.23	28.45	55.78	83.23	116.78	161.23	203.45

(c)

Table 3. (a) Endmember extraction accuracy and single-processing times (seconds) for the parallel target detection algorithms (HYDICE scene). (b) Parallel processing times (in seconds) on Thunderhead. (c) Speedup factors.

Regarding parallel target detection algorithms, Table 3(a) shows the order of extraction of panel center (red) pixels in the first column of the HYDICE data set (parameter *t* was set to 18 target pixels after calculating the virtual dimensionality of the data). As shown by Table 3(a), the P-ATGP was the only algorithm able to extract the five different center panel pixels among the first 18 targets. Quite opposite, the P-UFCLS could not detect the panel center pixel labeled as p_{21} and p_{41}. These panels are made up of the same material as p_{11} and p_{31}, respectively, but painted in different color. Also, the single-processor times given in Table 3(a) reveal that P-ATGP was faster than P-UFCLS in one Thunderhead processor. This is because the orthogonal projections implemented by the algorithm have similar computation cost, regardless of the number of target pixels involved. In turn, P-UFCLS incorporates LSE-based abundance estimation steps with complexity dependent on the number of target pixels used. Therefore, as the number of target pixels is increased, the computational complexity of this algorithm is higher. For illustrative purposes, Tables 3(b) and 3(c) show the processing times and the scalability of the considered parallel target detection algorithms, respectively. Although the P-ATGP consistently performed in less than one second using a moderate number of processors, the P-UFCLS scaled slightly better. This is because the P-ATGP involves several gather/scatter operations followed by compute-intensive orthogonal space projections which need to be completed sequentially at the master before a new parallel operation can be started by the workers. In turn, the P-UFCLS scales even better as a result of fewer data dependencies involved in this algorithm.

Although Tables 1-3 demonstrate that the proposed parallel implementations are quite satisfactory from the viewpoint of both processing times and scalability, many applications which demand a response in real-time. Although the idea of mounting clusters and networks of processing elements aboard 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 mission payload requirements in most remote sensing missions, which demand low-weight hardware components. As an alternative to cluster computing, FPGA-based processing provides several advantages, such as increased computational power, adaptability to different applications via reconfigurability, compact size, and lower cost than multi-processor systems. To conclude this section, Table 4 shows a summary of resource utilization by our systolic array-based implementation of the proposed data compression algorithm on the considered Xilinx FPGA, which was able to provide a response in only 40 milliseconds when applied to the AVIRIS Cuprite scene. Since the FPGA used in experiments has a total of 33,792 slices available, the results addressed in Table 4 indicate that there is still room in the FPGA for implementation of additional algorithms.

Number of gates	Number of slices	Percentage of slices used	Maximum operation frequency (MHz)
526,944	12,418	36%	18,032

Table 4. Summary of resource utilization for the FPGA-based implementation of the proposed data compression algorithm.

8. CONCLUSIONS

The aim of this paper has been the examination of different parallel computing-based strategies for remotely sensed hyperspectral imaging, with the purpose of evaluating the possibility of obtaining results in valid response times and with adequate reliability in several parallel platforms. Massively parallel computing architectures made up of commodity computing resources have gained popularity in the last few years due to the chance of building a "high performance system" at a reasonable cost. The scalability achieved by the proposed implementations in such low-cost systems offers an unprecedented opportunity to explore methodologies in other fields (e.g. data mining) that previously looked to be too computationally intensive for practical applications due to the immense files common to remote sensing databases. To address the real-time computational requirements introduced by many applications, we have also explored a systolic array-based FPGA implementation for onboard, hyperspectral data compression. Experimental results demonstrate that our hardware version makes an appropriate use of computing resources in the FPGA, and provides a response in (near) real-time. Although the experimental results presented in this paper are encouraging, further work is still needed to arrive to optimal parallel design and implementations of the considered and additional hyperspectral imaging algorithms.

REFERENCES

1. C.-I Chang, *Hyperspectral imaging: techniques for spectral detection and classification*, Kluwer/Plenum Publishers, 2003.
2. G. Aloisio, M. Cafaro, "A dynamic earth observation system," *Parallel Computing*, vol. 29, pp. 1357-1362, 2002.
3. J. Dorband, J. Palencia and U. Ranawake, "Commodity computing clusters at Goddard Space Flight Center," *Journal of Space Communication*, vol. 1, no. 3, 2003.
4. D. Valencia, A. Plaza, M. A. Vega-Rodriguez and R. Perez, "FPGA design and implementation of a fast pixel purity index algorithm for endmember extraction in hyperspectral imagery," in: *Proc. SPIE*, vol. 5995, 2005.
5. A. Plaza, D. Valencia, J. Plaza and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing*, vol. 66., no. 3, pp. 345-358, 2006.
6. T. Achalakul and S. Taylor, "A distributed spectral-screening PCT algorithm," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 373-384, 2003.
7. M. K. Dhodhi, J. A. Saghri, I. Ahmad and R. Ul-Mustafa, "D-ISODATA: A distributed algorithm for unsupervised classification of remotely sensed data on network of workstations," *Journal of Parallel and Distributed Computing*, vol. 59, pp. 280-301, 1999.
8. A. Plaza, P. Martinez, R. Perez, J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geoscience and Remote Sensing*, vol. 42, pp. 650-663, 2004.
9. J.W. Boardman, F.A. Kruse, R.O. Green, "Mapping target signatures via partial unmixing of AVIRIS data," in: *Summaries of JPL Airborne Earth Science Workshop*, Pasadena, CA, 1995.
10. M.E. Winter, "A proof of the N-FINDR algorithm for the automated detection of endmembers in a hyperspectral image," in: *Proc. SPIE*, vol. 5425, 2004.
11. Q. Du and C.-I Chang, "Linear mixture analysis-based compression for hyperspectral image analysis," *IEEE Trans. Geoscience and Remote Sensing*, vol. 42, no. 4, pp. 875-891, 2004.