Parallel Implementation of Hyperspectral Image Processing Algorithms

Antonio Plaza, David Valencia, Javier Plaza, Juan Sánchez-Testal, Sergio Muñoz and Soraya Blázquez

Department of Computer Science, University of Extremadura
Avda. de la Universidad s/n, E-10071 Cáceres, SPAIN
Contact e-mail: {aplaza, davaleco, jplaza}@unex.es

***Abstract*— High computing performance of algorithm analysis is essential in many hyperspectral imaging applications, including automatic target recognition for homeland defense and security, risk/hazard prevention and monitoring, wild-land fire tracking and biological threat detection. Despite the growing interest in hyperspectral imaging research, only a few efforts devoted to designing and implementing well-conformed parallel processing solutions currently exist in the open literature. With the recent explosion in the amount and dimensionality of hyperspectral imagery, parallel processing is expected to become a requirement in most remote sensing missions. In this paper, we take a necessary first step towards the quantitative comparison of parallel techniques and strategies for analyzing hyperspectral data sets. Our focus is on three types of algorithms: automatic target recognition, spectral mixture analysis and data compression. Three types of high performance computing platforms are used for demonstration purposes, including commodity cluster-based systems, heterogeneous networks of distributed workstations and hardware-based computer architectures. Combined, these parts deliver a snapshot of the state of the art in those areas, and offer a thoughtful perspective on the potential and emerging challenges of incorporating parallel computing models into hyperspectral remote sensing problems.**

I. Introduction

Hyperspectral imaging has been transformed in less than 30 years from being a sparse research tool into a commodity product available to a broad user community [1]. The development of computationally efficient techniques able to transform massive volumes of hyperspectral data sets, collected on a daily basis, into scientific understanding is critical for space-based Earth science and planetary exploration. To address the computational needs introduced by hyperspectral imaging applications, a few efforts have been directed towards the incorporation of high-performance computing models and platforms in remote sensing missions. For instance, the Center of Excellence in Space and Data Information Sciences (CESDIS), located at NASA's Goddard Space Flight Center in Maryland, developed the concept of Beowulf cluster 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 [2]. Although most dedicated parallel machines for hyperspectral image information extraction and mining have been chiefly homogeneous in nature [3], a current trend in the design of systems for analysis and interpretation of high-dimensional data sets is to utilize highly heterogeneous distributed platforms [4], which can benefit from local (user) computing resources and provide incremental scalability of hardware components, along with high communication speeds, at lower prices. In addition to cluster-based and distributed parallel computing facilities, critical hyperspectral imaging applications require analysis algorithms able to provide a response in (near) real-time. For this purpose, low-weight and low-power electronic components are mandatory to reduce payload and data transmission overheads. To achieve these goals, reconfigurable hardware platforms such as field programmable gate arrays (FPGAs) have opened many innovative perspectives, such as the possibility of on-board data processing and compression. In this paper, we provide a quantitative and comparative assessment of representative hyperspectral analysis algorithms for automatic target recognition and spectral mixture analysis. In addition, we also discuss a new parallel, exploitation-based algorithm for on-board hyperspectral data compression. Three different parallel computing platforms are used for demonstration purposes: a Beowulf cluster made up of 256 processors at NASA's Goddard Space Flight Center, a heterogeneous network of 16 distributed workstations at University of Maryland, and a Xilinx Virtex-II FPGA.

II. Parallel target detection algorithms

Two unsupervised target detection algorithms for hyperspectral imagery are considered in this section. These are the unsupervised fully constrained least squares (UFCLS) algorithm [1], and the iterative error analysis (IEA) algorithm [5]. The inputs to all discussed algorithms are a hyperspectral image $\mathbf{F}$ with n dimensions, where $\mathbf{F}(x, y)$ denotes the pixel vector at spatial coordinates (x, y), and a number of targets to be detected, t. The output in all cases is a set of target pixel vectors, denoted by $\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \ldots, \mathbf{t}^{(t)}$.

A. Parallel implementation of UFCLS (P-UFCLS)

The UFCLS [1] generates a set of t targets using the concept of least square-based error minimization. Our parallel version of UFCLS (called P-UFCLS) assumes that p processors are available and uses a master-slave implementation, in which a master processor coordinates the activities carried out by worker processors. The algorithm uses the following steps:

1) The master divides the original image cube $\mathbf{F}$ into $p-1$ chunks (the input data is partitioned so that each chunk

0-7803-9510-7/06/$20.00 © 2006 IEEE

contains entire pixel vectors only). Then, the master sends the $p-1$ partitions to the workers.

2) Each worker finds the brightest pixel in its local partition using $\mathbf{t}_i^{(1)} = arg\{max_{(x,y)}\mathbf{F}(x,y)^T \cdot \mathbf{F}(x,y)\}$, where the superscript T denotes the vector transpose operation and $i = 1, 2, \ldots, p-1$. 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, $\mathbf{t}^{(1)}$, 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 $\mathbf{U} = \mathbf{t}^{(1)}$ and broadcasts this matrix to all workers.
4) Each worker calculates the least squares-based error for each pixel vector in the input data represented in terms of a fully constrained linear mixture of all the spectra in $\mathbf{U}$. The workers then find the pixel vector with largest error score and sends its spatial coordinates and associated error to the master.
5) The master obtains a second target $\mathbf{t}^{(2)}$ by selecting the pixel vector with largest associated error score from the pixel vectors at the spatial locations provided by the workers and broadcasts $\mathbf{U} = \{\mathbf{t}^{(1)}, \mathbf{t}^{(2)}\}$ to the workers.
6) Repeat from step 4 to incorporate a new target pixel $\mathbf{t}^{(3)}, \mathbf{t}^{(4)}, \ldots, \mathbf{t}^{(t)}$ to $\mathbf{U}$ until a set of t target pixels have been extracted.

B. Parallel implementation of IEA (P-IEA)

The IEA [5] is similar to the UFCLS algorithm in the sense that both of them make use of least square-based error minimization to search for possible targets. While the P-UFCLS algorithm finds a pixel with the largest vector length to be used as its initial pixel to start the algorithm, our P-IEA calculates (in parallel) the sample mean vector for initialization. Another difference is that the P-UFCLS does not need any prior knowledge except the number of targets, t, to terminate the algorithm. As for the P-IEA, two additional parameters need to be incorporated. The first one is N_R, the number of pixels in $R^{(i)}$, which denotes the set of pixels with the largest errors in the resulting error images, $E^{(i)}$, at the worker processors, $i = 1, 2, \ldots, p-1$. The second parameter is θ, a spectral angle threshold used to find a set of spectrally similar pixels that will be averaged to generate new target pixels throughout the process. In this work, we set $N_R = 1$ and $\theta = 0$ to allow comparisons with P-UFCLS.

III. Parallel endmember extraction algorithms

Two algorithms for endmember extraction have been implemented by parallel approximations [6]: the pixel purity index (PPI) [7] and the N-FINDR [8] algorithm. The inputs to all discussed algorithms are a hyperspectral image $\mathbf{F}$ with n dimensions and a number of endmembers to be extracted, e. The output in all cases is a set of endmember pixels, denoted by $\{\mathbf{e}_i\}_{i=1}^{e}$.

A. Parallel PPI-like algorithm (P-PPI)

Boardman's PPI algorithm proceeds by generating a large number of random, n-dimensional unit vectors called *skewers* through the dataset. Every pixel in the input data is projected onto each skewer, and the number of times a given pixel is selected as extreme defines its *purity index*. Our master-slave parallel version (called P-PPI) is given by the following steps:

1) The master divides the original image cube $\mathbf{F}$ into $p-1$ chunks and sends them to the $p-1$ workers. The master also generates a random set of s skewers denoted by $\{\mathbf{skewer}_j\}_{j=1}^{s}$ and broadcasts them to the workers.
2) All data samples in each local partition are projected onto each $\mathbf{skewer}_j$ to find sample vectors at its extreme positions. As a result, an extrema set for each $\mathbf{skewer}_j$, denoted by $S(\mathbf{skewer}_j)$, is formed at the worker processors $i = 1, 2, \ldots, p-1$. If we assume that function $I_S^{(i)}$ is a logic function that returns '1' if a given pixel in the local partition is selected as extreme and '0' otherwise, we can obtain the number of times a given pixel is selected as extreme as: $N_{PPI}^{(i)}(\mathbf{F}(x,y)) = \sum_j I_{S(\mathbf{skewer}_j)}^{(i)}(\mathbf{F}(x,y))$. Each worker then selects those pixels with $N_{PPI}^{(i)}(\mathbf{F}(x,y))$ above a threshold parameter t, and sends their spatial locations to the master.
3) The master collects all the partial results and generates a unique set of final endmembers by using the spectral angle distance to retain only spectrally distinct pixels, until a final set of $\{\mathbf{e}_i\}_{i=1}^{e}$ endmembers is generated.

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

Winter's N-FINDR algorithm identifies the set of pixels which define the simplex with the maximum volume. A random set of pixel vectors is first selected, and their corresponding volume is calculated. A trial volume is then calculated for every pixel in each endmember position. If a replacement results in a volume increase, the pixel replaces the endmember and the procedure is repeated until no more replaces occur. A parallel approximation of this algorithm follows.

1) The master divides the original image cube $\mathbf{F}$ into $p-1$ chunks and sends them to the $p-1$ workers. It also selects a random set of e pixels from the input data and labels them as initial endmembers $\{\mathbf{e}_i^{(0)}\}_{i=1}^{e}$. Finally, the master calculates $V\{\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \ldots, \mathbf{e}_e^{(0)}\}$, the volume of the simplex defined by $\{\mathbf{e}_i^{(0)}\}_{i=1}^{e}$, and broadcasts this value to the workers.
2) The workers calculate the volume of e simplices $V\{\mathbf{F}(x,y), \mathbf{e}_2^{(0)}, ..., \mathbf{e}_e^{(0)}\}, ..., V\{\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, .., \mathbf{F}(x,y)\}$ in parallel, each of which is formed by replacing an endmember with the sample vector $\mathbf{F}$(x,y). Each worker performs replacements using pixels in its local partition.
3) If none of these e recalculated volumes is greater than $V\{\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \ldots, \mathbf{e}_e^{(0)}\}$, then no endmember is replaced. Otherwise, the master replaces the endmember which is absent in the largest volume among the e simplices generated in step 2 with the vector $\mathbf{F}$(x,y).

4) Repeat from step 2 until no replacements occur. The final combination gives the final set of endmembers $\{\mathbf{e}_i\}_{i=1}^{e}$.

IV. Parallel algorithm for data compression

In this section, we develop a new lossy compression technique able to reduce significantly the large volume of information contained in hyperspectral data while, at the same time, retaining mixed pixels and sub-pixel targets which are crucial in many applications. Since the standard approach for linear spectral unmixing (LSU) is an *embarrasingly parallel* problem (i.e., it can be parallelized with no data dependencies), we take advantage of this algorithm to produce, for each pixel vector **F**(x,y), a set of fractional abundances $\{a_1(x,y), a_2(x,y), \ldots, a_e(x,y)\}$ that can be used as a fingerprint of **F**(x,y) with regards to e endmembers, obtained in this work by the P-FINDR algorithm. Below, we describe the parallel algorithm and outline its FPGA-based implementation.

A. P-FINDR/P-LSU compression algorithm

1) Use the P-FINDR algorithm to obtain a set of endmembers $\{\mathbf{e}_i\}_{i=1}^{e}$.
2) Use an *embarrasingly parallel* version of linear spectral unmixing (P-LSU) to approximate each pixel vector $\mathbf{F}(x,y) = \mathbf{e}_1 \cdot a_1(x,y) + \mathbf{e}_2 \cdot a_2(x,y) + \ldots + \mathbf{e}_e \cdot a_e(x,y)$.
3) Construct e fractional abundance images, one for each P-FINDR-derived endmember, and apply lossless predictive coding to reduce spatial redundancy, using Huffman coding to encode predictive errors.

B. FPGA implementation

The algorithm above has been implemented in hardware using a standard systolic array-based architecture in which skewers are fed from top to bottom, and pixel vectors are fed from left to right [9]. Three types of processing nodes were used: *dot* nodes, which perform the individual products for the skewer projections, and *max* and *min* nodes, which respectively compute the maxima and minima projections after the dot product calculations have been completed. Part of the systolic array design is also employed to carry out the unmixing. Here, in order to obtain the abundance fractions for each pixel vector $\mathbf{F}(x,y)$, the systolic computes its product with $(\mathbf{M}^T \cdot \mathbf{M})^{-1} \cdot \mathbf{M}^T$, where $M = \{\mathbf{e}_i\}_{i=1}^{e}$.

V. Experiments

Before providing an an assessment of the effectiveness of the proposed parallel hyperspectral algorithms in the context of a mineral mapping application, we first provide a brief overview of the parallel computing architectures used in experiments.

A. Parallel computing architectures

Thee high performance computing platforms were used:

1) *Beowulf cluster*. We have used Thunderhead, a Beowulf cluster at NASA's Goddard Space Flight Center, 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 (see http://thunderhead.gsfc.nasa.gov for details).
2) *Heterogeneous network of workstations*. We have also used a heterogeneous network which consists of 16 different SGI, Solaris and Linux workstations, distributed among different locations and arranged in four relatively fast communication segments, interconnected by three slower communication links [10]. Implementations in both the Beowulf cluster and the heterogeneous network were carried out using C++ with calls to message passing interface (MPI).
3) *Field programmable gate array*. We have also used a Virtex-II XC2V6000-6 FPGA, which contains $33,792$ slices and 144 multipliers (of 18×18 bits). The synthesis in the FPGA board was performed using Handel-C (see http://www.celoxica.com), a design and prototyping language based on pseudo-C programming style.

TABLE I

Spectral angle-based similarity scores between USGS mineral spectra and pixels produced by parallel algorithms

	P-UFCLS	**P-IEA**	**P-PPI**	**P-FINDR**
Processing time (seconds)	972	1120	2745	695
Alunite	0.116	0.116	0.099	0.081
Buddingtonite	0.106	0.125	0.106	0.084
Calcite	0.105	0.105	0.105	0.105
Chlorite	0.125	0.144	0.125	0.096
Kaolinite	0.136	0.136	0.136	0.136
Jarosite	0.134	0.134	0.112	0.102
Montmorillonite	0.110	0.108	0.106	0.089
Muscovite	0.113	0.115	0.108	0.094
Nontronite	0.102	0.102	0.102	0.099
Pyrophilite	0.099	0.102	0.094	0.090

B. Parallel performance evaluation

The hyperspectral data set used in experiments is the well-known AVIRIS Cuprite scene, available online (in reflectance units) from http://aviris.jpl.nasa.gov/html/aviris.freedata.html. The scene comprises a relatively large area, with 20-meter pixels and 224 narrow spectral bands between 0.4 and 2.5 μm. The reflectance spectra of ten U.S. Geological Survey (USGS) ground-truth mineral spectra: *alunite*, *buddingtonite*, *calcite*, *chlorite*, *kaolinite*, *jarosite*, *montmorillonite*, *muscovite*, *nontronite* and *pyrophilite* (all available from http://speclab.cr.usgs.gov) were used for evaluation purposes. Table I tabulates the spectral angle mapper scores obtained after comparing library spectra with the corresponding endmembers extracted by the parallel algorithms (the smaller the scores across the five minerals considered in Table I, the better the results). The number of pixels to be extracted in all cases was set to 16 after calculating the intrinsic dimensionality of the data. For illustrative purposes, Table I also reports processing times (in seconds) using one single processor of the Thunderhead cluster. It can be seen from the table that all tested parallel target detection and endmember extraction algorithms were able to identify highly pure instances of the ten ground-truth minerals, with P-FINDR producing the most highly pure signatures. However, processing times measured

TABLE II

PROCESSING TIMES USING DIFFERENT PROCESSORS ON THUNDERHEAD

	4	**16**	**36**	**64**	**100**	**144**	**196**	**256**
P-UFCLS	312	95	34	17	12	8	6	5
P-IEA	345	103	39	20	14	10	7	6
P-PPI	1013	251	99	52	34	24	18	15
P-FINDR	199	46	24	15	9	6	5	4
P-LSU	123	31	15	8	6	4	3	2

TABLE III

PROCESSING TIMES ON THE 16-PROCESSOR HETEROGENEOUS NETWORK

	P-UFCLS	**P-IEA**	**P-PPI**	**P-FINDR**	**P-LSU**
Time (seconds)	116	139	272	54	42

on a single processor were significant in all cases.

To empirically investigate the scaling properties of the parallel algorithms, their performance was tested by timing the programs over various numbers of Thunderhead (see Table II). Results in Table II reveal that multi-processor runs of the parallel algorithms can significantly reduce the single-processor times given in Table I. The P-LSU achieved speedups close to optimal, while both P-IEA and P-UFCLS scaled slightly better than P-PPI and P-FINDR. This is because parallel target detection algorithms involved less data dependencies than parallel endmember extraction algorithms.

On the other hand, Table III tabulates the processing times obtained in the 16-processor heterogeneous network. Here, the data partitioning step was modified for all parallel algorithms so that the size of the data chunk sent by the master processor to each worker processor was adaptively established in runtime, taking into account the relative speed of each heterogeneous processor and the capacity of communication links. As we can see from Table III, the measured times were slightly higher than those reported for 16 processors on Thunderhead, which is a homogeneous system made up of identical processors and communication links. However, we experimentally tested that load balance among the different processors in the heterogeneous network was highly efficient, as a consequence of our adaptive data partitioning and distribution framework. With the above considerations in mind, results in Table III are very encouraging, in particular, given the inherent complexity of heterogeneous, Grid-like networks of computers.

Although Tables II and III demonstrate that the proposed parallel implementations can be effectively ported to large-scale multiprocessor systems, many applications demand a response in real-time. In this regard, onboard compression systems for hyperspectral imagery are essential in order to overcome data downlink restrictions introduced by the ever growing dimensionality of remotely sensed data sets. Although the idea of mounting clusters/networks of workstations aboard airborne and satellite hyperspectral imaging facilities has been explored in the past, mission payload requirements clearly demand low-weight electronics and hardware components. As an efficient alternative to the implementations discussed above, we have tested the performance of our P-FINDR/P-LSU data compression algorithm on the considered Xilinx Virtex-II FPGA. First, we conducted several experiments to evaluate the performance of the compression algorithm and observed that compression ratios above $60:1$ could be achieved with no apparent degradation in the quality of spectral information (including mixed pixels and sub-pixel targets). The algorithm mapped nicely on the FPGA, and was able to compress the full AVIRIS Cuprite data in only 40 milliseconds. Out of the total $33,792$ slices available in the FPGA, only 36% were required for the implementation of our algorithm, which indicates that there is still room in the FPGA for additional implementations. The reconfigurability of FPGA boards also opens the appealing possibility of adaptively selecting one out of a pool of algorithms to be applied *on the fly*, i.e., as the data is collected, from a ground control station on Earth.

VI. CONCLUSIONS

In this paper, we have examined different implementations of hyperspectral image processing algorithms, with the purpose of evaluating the possibility of obtaining results in valid response times and with adequate reliability across several types of last-generation parallel computing platforms. Experimental results demonstrate that massively parallel Beowulf clusters and low-cost heterogeneous networks of workstations offer an unprecedented opportunity to explore methodologies in fields (e.g., data mining) that previously looked to be too computationally intensive due to the immense volumes of information in remote sensing databases. To address the real-time computational requirements introduced by many applications, we have also developed an FPGA-based algorithm for onboard, hyperspectral data compression. This paper constitutes a first step towards a standardized, cross-platform assessment of parallel and distributed algorithms for hyperspectral imaging.

REFERENCES

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

[2] J. Dorband, J. Palencia, and U. Ranawake, "Commodity clusters at Goddard Space Flight Center," *J. Space Communication*, 2003.

[3] 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, March 2006.

[4] A. Lastovetsky, *Parallel computing on heterogeneous networks*, Wiley-Interscience: Hoboken, NJ, 2003.

[5] R. A. Neville, K. Staenz, T. Szeredi, J. Lefebvre, and P. Hauff, "Automatic endmember extraction from hyperspectral data for mineral exploration," in *21st Canadian Symposium on Remote Sensing*, 1999.

[6] A. Plaza, D. Valencia, J. Plaza, and C.-I Chang, "Parallel implementation of endmember extraction algorithms from hyperspectral data," *IEEE Geosci. Remote Sensing Letters*, vol. 3, no. 3, July 2006.

[7] J. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of aviris data," in *Summaries of JPL Airborne Earth Science Workshop*, 1995.

[8] M. E. Winter, "N-FINDR: an algorithm for fast autonomous spectral endmember determination in hyperspectral data," in *Proceedings SPIE Imaging Spectrometry V*, 1999, vol. 3753, pp. 266–277.

[9] D. Valencia and A. Plaza, "FPGA-based compression of hyperspectral imagery using spectral unmixing and the pixel purity index algorithm," *Lecture Notes in Computer Science*, vol. 3991, no. 1, pp. 888–891, 2006.

[10] A. Plaza, J. Plaza, and D. Valencia, "AMEEPAR: Parallel morphological algorithm for hyperspectral image classification in heterogeneous networks of workstations," *Lecture Notes in Computer Science*, vol. 3993, no. 3, pp. 24–31, 2006.