Computers & Geosciences 36 (2010) 1283–1291

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Improving the scalability of hyperspectral imaging applications on heterogeneous platforms using adaptive run-time data compression

Antonio Plaza *, Javier Plaza, Abel Paz

Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, Avda. de la Universdad s/n, E-10071 Cáceres, Spain

ARTICLE INFO

Article history:
Received 16 August 2009
Received in revised form
7 December 2009
Accepted 24 February 2010

Keywords:
Heterogeneous parallel computing
Adaptive run-time data compression
Wavelet transform
Hyperspectral imaging
Remote sensing

ABSTRACT

Latest generation remote sensing instruments (called hyperspectral imagers) are now able to generate hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth. In previous work, we have reported that the scalability of parallel processing algorithms dealing with these high-dimensional data volumes is affected by the amount of data to be exchanged through the communication network of the system. However, large messages are common in hyperspectral imaging applications since processing algorithms are pixel-based, and each pixel vector to be exchanged through the communication network is made up of hundreds of spectral values. Thus, decreasing the amount of data to be exchanged could improve the scalability and parallel performance. In this paper, we propose a new framework based on intelligent utilization of wavelet-based data compression techniques for improving the scalability of a standard hyperspectral image processing chain on heterogeneous networks of workstations. This type of parallel platform is quickly becoming a standard in hyperspectral image processing due to the distributed nature of collected hyperspectral data as well as its flexibility and low cost. Our experimental results indicate that adaptive lossy compression can lead to improvements in the scalability of the hyperspectral processing chain without sacrificing analysis accuracy, even at sub-pixel precision levels.

1. Introduction

High performance computing has become a standard solution in order to deal with high response times in large-scale remote sensing applications (Plaza and Chang, 2007). Although parallel processing in clusters of computers has been an active research area in the geosciences community (Dai, 2005; Tian et al., 2008; Sourbier et al., 2009), a recent trend in Earth-Observation (EO) data processing and other applications is to utilize highly heterogeneous and distributed parallel platforms (Lastovetsky, 2003; Zeng and McMechan, 2002). These systems can benefit from local (user) computing resources in order to efficiently store and handle high-dimensional data archives resulting from the EO campaigns of latest-generation imaging instruments, such as the NASA Jet Propulsion Laboratory's Airborne Visible Infrared Imaging Spectrometer (AVIRIS) (Green et al., 1998). This instrument can now record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 μm) of the reflected light of an area 2–12 km wide and several km long using 224 narrow spectral bands. The resulting data volume is commonly referred to as *hyperspectral image cube* (Goetz et al., 1985) (see Fig. 1). The increased spectral resolution of hyperspectral imagers with regards to multispectral instruments (Biehl and Landgrebe, 2002) (hundreds versus tens of spectral channels) has introduced new processing requirements (Chang, 2003).

In previous work, we have reported that the scalability of parallel hyperspectral imaging algorithms is directly related to the amount of information to be exchanged through the communication network of the system when the parallel algorithm is run (Plaza, 2008a), i.e. large message sizes (typical in hyperspectral imaging applications, since each pixel vector is made up of hundreds of spectral values) can be harmful for the scalability of the parallel code (Plaza, 2008b; Plaza et al., 2007). Our speculation in this work is that decreasing the amount of information to be exchanged can improve the scalability of hyperspectral imaging algorithms on heterogeneous networks. This dependence on the interconnect is even more critical in clusters of computers, where the interconnect is a typically an off-the-shelf hardware such as a Gigabit Ethernet, which suffers from high latency and relatively low-bandwidth (Kumar et al., 2008). The bottleneck becomes more severe when the interconnection network is saturated with a large number of processors sending large messages.

In this paper we investigate a new framework, based on the utilization of wavelet-based data compression techniques

* Corresponding author. Tel.: +34 927 257195; fax: +34 927 257203.
E-mail address: aplaza@unex.es (A. Plaza).
URL: http://www.umbc.edu/rssipl/people/aplaza.

0098-3004/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cageo.2010.02.009

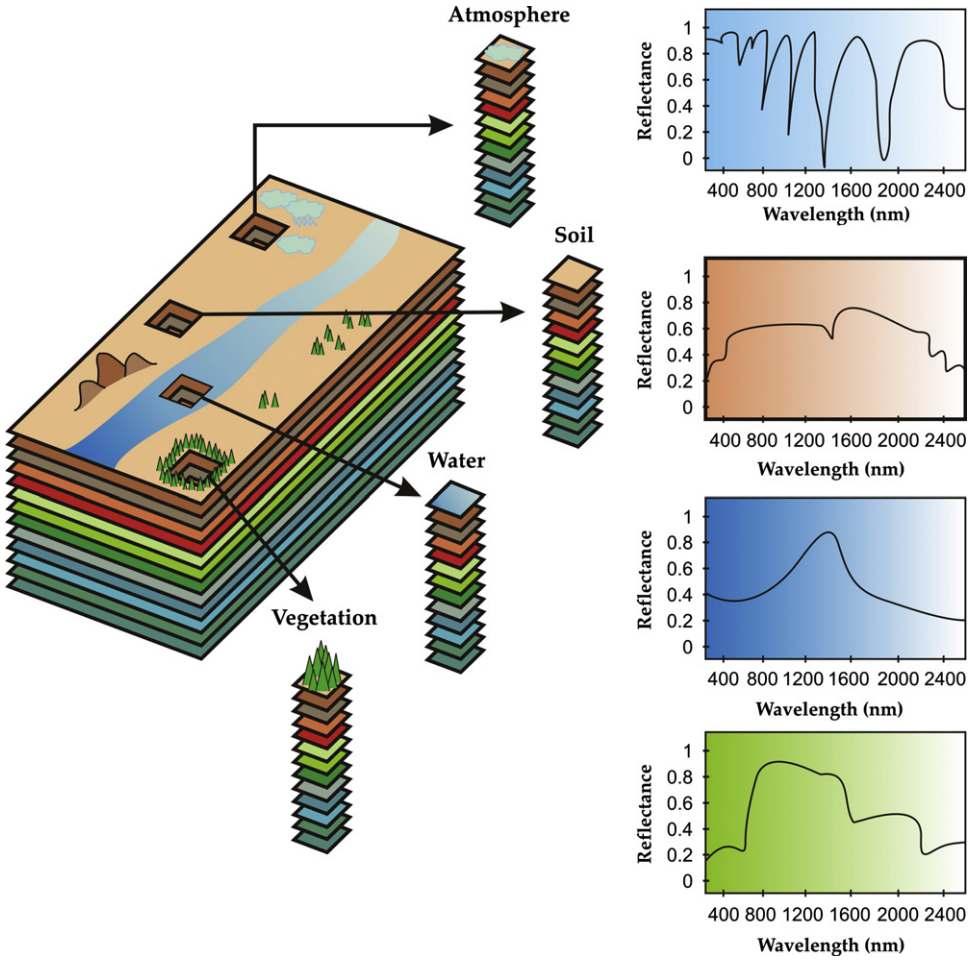

Fig. 1. Concept of hyperspectral imaging.

(Kaewpijit et al., 2003), for reducing the amount of communications required by a parallel hyperspectral image processing chain on heterogeneous platforms. Two types of data compression can be performed, lossless and lossy (Motta et al., 2006), in accordance with redundancy removal. Lossless compression involves no loss of information and enables perfect reconstruction of the pixel vector after reception. On the other hand, lossy compression involves some *acceptable* loss of information, where the term *acceptable* needs to be substantiated by analyzing the trade-off between the increase in parallel performance and the potential decrease in analysis accuracy of the algorithm. Which type of compression should be used depends heavily upon the application domain (Ramakhrishna et al., 2006). For example, in medical imaging applications, lossless compression is preferred to lossy compression in order to avoid potential lawsuits against doctors. However, since the best compression ratios achieved by lossless methods are in the order of 3:1, lossy compression is generally preferred in hyperspectral imaging applications, in which higher compression ratios are generally required (Motta et al., 2006). Parallel compression is also of great interest in hyperspectral data exploitation (Du et al., 2009).

In previous work, it has been demonstrated that spectral mixture analysis (Adams et al., 1986) represents an adequate alternative to achieve lossy compression of hyperspectral data while retaining the information that is most relevant for analyzing the data with sub-pixel precision (Du and Chang, 2004). Specifically, in this work we propose a lossy compression approach combined with a spectral unmixing-based (Keshava and Mustard, 2002) processing chain (available, among others, in the Environment for Visualizing Images (ENVI),[1] a very popular remote sensing software package) to achieve lossy but controlled compression at different ratios. This strategy is shown to increase the parallel performance of a heterogeneous implementation of the processing chain on a fully heterogeneous network of workstations, which is a standardized type of distributed parallel platform currently used by several agencies and organizations for storing and managing hyperspectral data sets distributed among different locations.

The remainder of the paper is organized as follows: In Section 2, we describe the standard hyperspectral image processing chain used in this work as a case study. In Section 3, we present the parallel heterogeneous implementation. Section 4 discusses the proposed wavelet-based compression scheme. Section 5 presents an experimental validation of the proposed approach, with and without data compression. Section 6 concludes and provides hints at future research directions.

2. Hyperspectral image processing chain

2.1. Problem formulation

Let us assume that a hyperspectral scene with N bands is denoted by $\mathbf{F}$, in which a pixel of the scene is represented by a vector $\mathbf{f}_i = [f_{i1}, f_{i2}, \ldots f_{in}] \in \Re^N$, where $\Re$ denotes the set of real

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

numbers in which the pixel's spectral response f_{ik} at sensor wavelengths $k=1,\ldots,N$ is included. Under the linear mixture model assumption (Plaza et al., 2004), each pixel vector in the scene can be modeled using:

$$\mathbf{f}_i = \sum_{e=1}^{E} \mathbf{e}_e \cdot a_{\mathbf{e}_e} + \mathbf{n}, \quad (1)$$

where $\mathbf{e}_e$ denotes the spectral response of a pure spectral signature (*endmember* (Plaza et al., 2002) in hyperspectral imaging terminology), $a_{\mathbf{e}_e}$ is a scalar value designating the fractional abundance of the endmember $\mathbf{e}_e$, E is the total number of endmembers, and $\mathbf{n}$ is a noise vector. The use of spectral endmembers allows one to deal with the problem of mixed pixels (Penn, 2002), which arise when the spatial resolution of the sensor is not high enough to separate different materials. For instance, it is likely that the pixel labeled as 'vegetation' in Fig. 1 actually comprises a mixture of vegetation and soil. In this case, the measured spectrum can be decomposed into a linear combination of pure spectral endmembers of soil and vegetation, weighted by abundance fractions that indicate the proportion of each endmember in the mixed pixel (Keshava and Mustard, 2002). The solution of the linear spectral mixture problem in (1) relies on the correct determination of a set $\{\mathbf{e}_e\}_{e=1}^{E}$ of endmembers and their correspondent abundance fractions $\{a_{\mathbf{e}_e}\}_{e=1}^{E}$ at each image pixel $\mathbf{f}_i$.

2.2. Processing chain

The inputs to the hyperspectral processing chain considered in this work are a hyperspectral image cube $\mathbf{F}$ with N spectral bands and T pixel vectors; the number of endmembers to be extracted, E, a maximum number of projections, K; a cut-off threshold value, ν_c, used to select as final endmembers only those pixels that have been selected as extreme pixels at least ν_c times after K projections; and a threshold angle, ν_a, used to discard redundant endmembers. The processing chain can be summarized by the following steps:

(1) *Skewer generation*: Produce a set of K randomly generated unit vectors for pixel purity indexing (Boardman, 1993), denoted by $\{\mathbf{skewer}_j\}_{j=1}^{K}$.

(2) *Extreme projections*: For each $\mathbf{skewer}_j$, all sample pixel vectors $\mathbf{f}_i$ in the original data set $\mathbf{F}$ are projected onto $\mathbf{skewer}_j$ via dot products of $|\mathbf{f}_i \cdot \mathbf{skewer}_j|$ to find the pixel vectors at extreme (maximum and minimum) projections, forming an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}(\mathbf{skewer}_j)$. Define an indicator function of a set S, denoted by $I_S(\mathbf{f}_i)$, to denote membership of an element $\mathbf{f}_i$ to that particular set as $I_S(\mathbf{f}_i)=1$ if $\mathbf{f}_i \in S$. Using the indicator function above, calculate the number of times that a given pixel has been selected as extreme using

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

(3) *Endmember selection*: Find the pixels with value of $N_{times}(\mathbf{f}_i)$ above ν_c and form a unique set of E endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the spectral angle distance (SAD) for all possible endmember pairs and discarding those which result in an angle value below ν_a. SAD is invariant to multiplicative scalings that may arise due to differences in illumination and sensor observation angle (Chang, 2003). The SAD between two endmembers $\mathbf{e}_i$ and $\mathbf{e}_j$ is given by

$$\mathrm{SAD}(\mathbf{e}_i,\mathbf{e}_j) = \cos^{-1}\frac{\mathbf{e}_i \cdot \mathbf{e}_j}{\|\mathbf{e}_i\| \cdot \|\mathbf{e}_j\|}. \quad (3)$$

(4) *Spectral unmixing*: For each pixel vector $\mathbf{f}_i$ in $\mathbf{F}$, a set of abundance fractions specified by $\{a_{\mathbf{e}_1}, a_{\mathbf{e}_2}, \ldots, a_{\mathbf{e}_E}\}$ 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 by minimizing the term $\mathbf{n}$ in the following expression: $\mathbf{f}_i = \mathbf{e}_1 \cdot a_{\mathbf{e}_1} + \mathbf{e}_2 \cdot a_{\mathbf{e}_2} + \cdots + \mathbf{e}_E \cdot a_{\mathbf{e}_E} + \mathbf{n}$, thus solving the linear mixture problem in (1) (Chang, 2003).

3. Parallel heterogeneous implementation

Let us assume that p_i denotes a processing node weighted by its relative cycle-time t_i. Similarly, let us assume that c_{ij} denotes the maximum capacity of the slowest link in the path of physical communication links from p_i to p_j. In order to balance the load in a fully heterogeneous environment, processor p_i should accomplish a share of $\alpha_i \cdot W$ of the original workload, denoted by W, to be accomplished by a certain algorithm, with $\alpha_i \geq 0$ for $1 \leq i \leq P$ and $\sum_{i=1}^{P} \alpha_i = 1$, being P the total number of processors in the system. With the above notation in mind (Legrand et al., 2004), we have developed a master-slave heterogeneous implementation of the processing chain in Section 2 using the C++ programming language, with calls to message passing interface (MPI). The proposed implementation consists of the following stages:

3.1. Workload estimation algorithm

This stage first obtains necessary information about the heterogeneous system, including the number of available processing nodes P, each processor's identification number $\{p_i\}_{i=1}^{P}$, and processor cycle-times $\{t_i\}_{i=1}^{P}$. Then, a master processor performs the following steps:

(1) Set $\alpha_i = \lfloor (1/t_i)/\sum_{i=1}^{P}(1/t_i) \rfloor$ for all $i \in \{1,\ldots,P\}$, i.e. this step first approximates the $\{\alpha_i\}_{i=1}^{P}$ so that $\alpha_i \cdot t_i \approx const$ for all processors.

(2) Iteratively increment some α_i until the set of $\{\alpha_i\}_{i=1}^{P}$ best approximates the total workload to be completed, i.e. for $m = \sum_{i=1}^{P} \alpha_i$ to W, find $k \in \{1,\ldots,P\}$ so that $t_k \cdot (\alpha_k + 1) = \min\{t_i \cdot (\alpha_i + 1)\}_{i=1}^{P}$, and then set $\alpha_k = \alpha_k + 1$.

(3) Produce P partitions of the input hyperspectral data set as follows (Plaza et al., 2008):

   (a) Establish an initial partitioning of the data so that the number of pixel vectors in each partition is proportional to the estimated values of $\{\alpha_i\}_{i=1}^{P}$, and assuming that no upper bound exist on the number of pixel vectors that can be stored by the local memory.

   (b) For each processor p_i, check if the number of pixel vectors assigned to it is greater than the upper bound. For all the processors whose upper bounds are exceeded, assign them a number of pixels equal to their upper bounds. Now, solve the partitioning problem of a set with remaining pixel vectors over the remaining processors until all pixel vectors in the input data have been assigned.

   (c) Iteratively recalculate the workload assigned to each processor using:

$$W_i^k = W_i^{k-1} - \sum_{j \in N(i)} c_{ij}\left(\frac{W_i^{k-1}}{t_i} - \frac{W_j^{k-1}}{t_j}\right) \quad (4)$$

where $N(i)$ denotes the set of neighbors of processing node p_i, and W_i^k denotes the workload of p_i (i.e., the number of pixel vectors assigned to this processor) after the k-th iteration. This scheme has been demonstrated to

converge to an average workload $\overline{W_i} := (\sum_{j=1}^{P} W_j / \sum_{j=1}^{P} t_j) t_i$ (Elsasser et al., 2002).

3.2. Parallel algorithm

Using the P heterogeneous partitions obtained in the previous step, the following parallel algorithm is now applied:

(1) *Skewer generation*: Generate K random unit vectors $\{\mathbf{skewer}_j\}_{j=1}^{K}$ in parallel, and broadcast the entire set of skewers to all the workers.
(2) *Extreme projections*: For each $\mathbf{skewer}_j$, each worker projects all the sample pixel vectors at its local partition l (where $1 \le l \le P$) onto $\mathbf{skewer}_j$ to find sample vectors at its extreme projections, and forms an extrema set for $\mathbf{skewer}_j$ which is denoted by $S^{(l)}_{extrema}(\mathbf{skewer}_j)$. Now each worker calculates the number of times that each pixel vector $\mathbf{f}_i^{(l)}$ in each local partition is selected as extreme using

$$N^{(l)}_{times}(\mathbf{f}_i^{(l)}) = \sum_{j=1}^{K} I_{S^{(l)}_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i^{(l)}). \quad (5)$$

(3) *Endmember selection*: Each worker selects those pixel vectors which satisfy $N^{(l)}_{times}(\mathbf{f}_i^{(l)}) > v_c$, and then sends the spatial coordinates of those pixels to the master node. The master now forms a unique set $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the SAD for all possible pixel vector pairs provided by the workers in parallel, and discarding those pixels which result in angle values below v_a.
(4) *Spectral unmixing*: The master broadcasts the final set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ to all workers. Each worker then obtains a set of fractional abundances $\{a^{(l)}_{\mathbf{e}_1}, a^{(l)}_{\mathbf{e}_2}, \ldots, a^{(l)}_{\mathbf{e}_E}\}$ for each pixel vector $\mathbf{f}_i^{(l)}$ in its local partition l, so that the term $\mathbf{n}$ in the following expression is minimized: $\mathbf{f}_i^{(l)} = \mathbf{e}_1 \cdot a^{(l)}_{\mathbf{e}_1} + \mathbf{e}_2 \cdot a^{(l)}_{\mathbf{e}_2} + \cdots + \mathbf{e}_E \cdot a^{(l)}_{\mathbf{e}_E} + \mathbf{n}$. The workers finally send the local results to the master, which combines them and forms the final output.

4. Adaptive data compression

The idea of the adopted lossy compression method is to apply a discrete wavelet transform (Kaewpijit et al., 2003) in the spectral domain before communicating each vector through the network. This includes the pixels distributed to slave processors in the *workload estimation* step, the skewers broadcast by the master in the *skewer generation* step, the final endmembers broadcast by the master in the *spectral unmixing* step, and the local results returned by the workers to the master in the same step. The lossy compression procedure does not only reduce the data volume to be communicated, but it can also preserve the characteristics of the spectral signatures. This is due to the intrinsic property of wavelet transforms, which preserve high and low-frequency features during the signal decomposition, therefore retaining the peaks and valleys found in typical spectra.

One of the advantages of using a one-dimensional wavelet transform (across spectral bands) is the fact that this is a very localized operation, which means less accesses to secondary storage and better cache memory system utilization. This can be greatly beneficial in terms of achieving data locality while, at the same time, reducing interprocessor communications. Another advantage of using the wavelet transform for compression purposes is that its performance can be better for larger dimensions. This property, which fits well the analysis of hyperspectral signals due to their high dimensionality, results from the very nature of wavelet compression, where significant features of the signal might be lost when the signal is undersampled. A final major advantage is that the compression process is pixel-based, and therefore it can be effectively integrated in the heterogeneous algorithm described in Section 3 and executed in parallel. A general description of the encoder and decoder modules of the adopted wavelet-based compression strategy follows:

- *Encoder*: Each time the master processor or a slave processor needs to communicate a pixel vector through the system, its associated signal (spectral signature) is decomposed (encoded) using a Daubechies wavelet (filter size four) (Kaewpijit et al., 2003). This transform decomposes each signature into a set of composite bands that are linear, weighted combinations of the original spectral bands. An example of the encoding process for a standard AVIRIS spectral signature is shown in Fig. 3. As the number of wavelet decomposition levels increases, the structure of the spectral signature becomes smoother than the structure of the original signature. It can be seen that the overall structure of the signal is recognizable until level 2 and more degraded at level 3.
- *Decoder*: An approximation of the original spectrum at pixel vector can be reconstructed (decoded) after reception of the compressed signal and the wavelet coefficients (which are transmitted through the network) using an inverse discrete wavelet transform (Kaewpijit et al., 2003). Because the reconstructed spectral data are produced from the approximation, the more levels is which the signal is decomposed, the more different the reconstructed signal is with regards to the original signal, because of the loss of high-pass components.

At this point, it is important to emphasize that the needed level of decomposition for each given pixel should be the one that corresponds to producing an acceptable spectral similarity with the original signature. In this work, we make use of the spectral angle distance in (3) as a quantitative indicator to measure the similarity between the original spectral signature and the reconstructed spectral approximation. A threshold similarity angle parameter, t_a, is fixed in advance, so that the number of levels of the wavelet decomposition is adaptively defined (at runtime) for each pixel in order to meet the desired similarity threshold in the reconstructed pixel with regards to the original one. For instance, setting $t_a = 0.1$ (a reasonable similarity threshold as studied in previous work, Plaza et al., 2004) automatically discards level 3 and retains level 2 as the highest acceptable decomposition level to avoid significantly degrading the original signal in the example given in Fig. 3. Note that the number of levels needed to meet the quantitative criterion above varies from one pixel to another, depending on the complexity of its associated spectral signature.

5. Experimental results

In this section we evaluate the impact of including data compression on the scalability of the parallel heterogeneous algorithm described in Section 3. This section is structured as follows. First, we present the hyperspectral image data set that will be used in experiments, along with the analysis results obtained after applying the processing chain in Section 2 to the same data set. Then, we describe the heterogeneous parallel platform used for experimentation. Next, we introduce the motivation for data compression. The section concludes with an experimental assessment of the increase in parallel performance

Fig. 2. Hyperspectral image (left). Location of fires in World Trade Center (right).

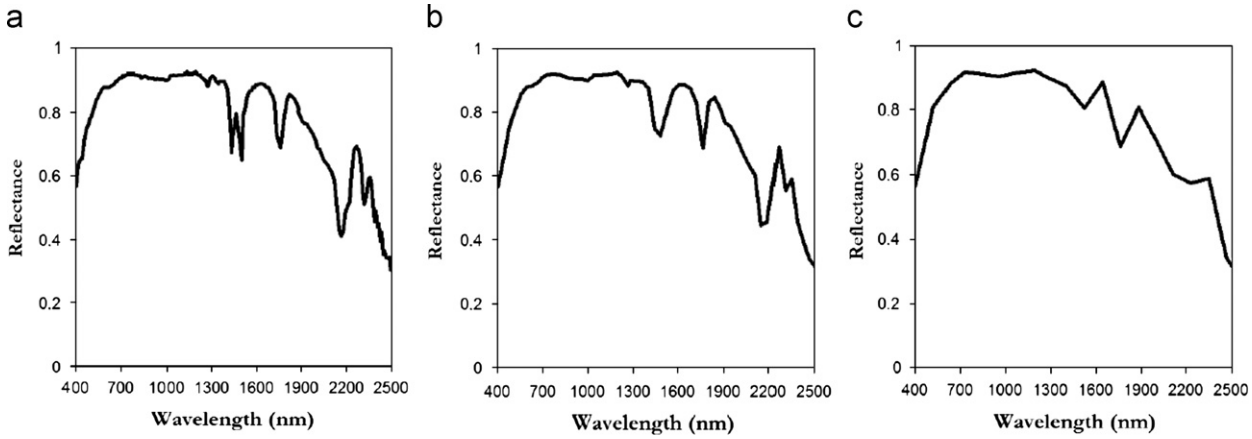

Fig. 3. Example of a mineral spectral signature and different levels of wavelet decomposition for lowpass component. (a) Level 1: 200 bands. (b) Level 2: 100 bands. (c) Level 3: 25 bands.

observed after applying the proposed adaptive data compression strategy.

5.1. Hyperspectral data

The image scene used for experiments in this work was collected by the AVIRIS instrument, flown over the World Trade Center (WTC) area in New York City on September 16, 2001, just five days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex. The data set consists of 614 $\times$ 512 pixels, 200 spectral values per pixel (each coded using 16 bits), for a total size of about 123 MB. Water absorption and noisy channels were removed prior to data processing from the original 224-band data set due to low signal to noise ratio in those channels. Fig. 2 (left) shows a false color composite of the data set selected for experiments. Fig. 2 (right) shows a thermal map centered at the region where the buildings collapsed. The map shows the target locations of the thermal hot spots, used in this work as ground-truth to validate the parallel hyperspectral processing algorithms.

Table 1 evaluates the accuracy of the hyperspectral processing chain described in Section 2 in estimating the sub-pixel abundance of fires in Fig. 2 (right), taking advantage of the information about the area covered by each thermal hot spot available from U.S. Geological Survey (USGS).[2] Since each pixel in the AVIRIS scene has a size of 1.7 square meters, it is inferred that the thermal hot spots are sub-pixel in nature, and thus require spectral unmixing in order to be characterized. Experiments in Table 1 demonstrate that the processing chain can provide accurate estimations of the area covered by thermal hot spots. In particular, the estimations for the thermal hot spots with higher temperature (labeled as 'A' and 'G' in the table) were almost perfect. Here, the processing chain was run using a total of $K=10^4$ skewers, with the cutoff threshold parameter v_c set to the mean of N_{times} scores after 10^4 iterations, and the threshold angle value set to $v_a=0.1$, a reasonable limit of tolerance for this metric. These parameter values are in agreement with those used before in the literature (Plaza et al., 2004). In this experiment, the number of endmembers was set to $E=15$ using the virtual dimensionality concept (Chang, 2003), which automatically calculates the number of endmembers in a scene.

5.2. Heterogeneous platform

The fully heterogeneous network considered in our study consists of 16 different workstations, and four communication

[2] http://speclab.cr.usgs.gov/wtc

Table 1
Comparison of area estimation (in square meters) for each thermal hot spot by parallel implementations of hyperspectral processing chain (USGS reference values are also included).

Thermal hot spot	Latitude (North)	Longitude (West)	Temperature (K)	Area (USGS)	Area (Chain)
'A'	40°42′47.18″	74°00′41.43″	1000	0.56	0.55
'B'	40°42′47.14″	74°00′43.53″	830	0.08	0.06
'C'	40°42′42.89″	74°00′48.88″	900	0.80	0.78
'D'	40°42′41.99″	74°00′46.94″	790	0.80	0.81
'E'	40°42′40.58″	74°00′50.15″	710	0.40	0.45
'F'	40°42′38.74″	74°00′46.70″	700	0.40	0.37
'G'	40°42′39.94″	74°00′45.37″	1020	0.04	0.05
'H'	40°42′38.60″	74°00′43.51″	820	0.08	0.09

Table 2
Specifications of heterogeneous processors.

Processor number	Architecture specification	Cycle-time (s/megaflop)	Memory (MB)	Cache (KB)
p_1	Free BSD–i386 Intel Pentium 4	0.0058	2048	1024
p_2, p_5, p_8	Linux–Intel Xeon	0.0102	1024	512
p_3	Linux–AMD Athlon	0.0026	7748	512
p_4, p_6, p_7, p_9	Linux–Intel Xeon	0.0072	1024	1024
p_{10}	SunOS–SUNW UltraSparc-5	0.0451	512	2048
p_{11}–p_{16}	Linux–AMD Athlon	0.0131	2048	1024

Table 3
Capacity of communication links (time in milliseconds to transfer a one-megabit message).

Processor	p_1–p_4	p_5–p_8	p_9–p_{10}	p_{11}–p_{16}
p_1–p_4	19.26	48.31	96.62	154.76
p_5–p_8	48.31	17.65	48.31	106.45
p_9–p_{10}	96.62	48.31	16.38	58.14
p_{11}–p_{16}	154.76	106.45	58.14	14.05

segments. Table 2 presents the properties of the 16 heterogeneous workstations, where processors $\{p_i\}_{i=1}^{4}$ are attached to communication segment s_1, processors $\{p_i\}_{i=5}^{8}$ communicate through s_2, processors $\{p_i\}_{i=9}^{10}$ are interconnected via s_3, and processors $\{p_i\}_{i=11}^{16}$ share the communication segment s_4. The communication links between the different segments $\{s_j\}_{j=1}^{4}$ only support serial communication. For illustrative purposes, Table 3 also presents the capacity of all point-to-point communications in the heterogeneous network, expressed as the time in milliseconds to transfer a one-megabit message between each processor pair (p_i,p_j) in the heterogeneous system. As noted, the communication network of the fully heterogeneous network consists of four relatively fast homogeneous communication segments, interconnected by three slower communication links with capacities $c^{(1,2)}=29.05$, $c^{(2,3)}=48.31$, $c^{(3,4)}=58.14$ in milliseconds, respectively. Although this is a simple architecture, it is also a quite typical and realistic one as well.

5.3. Motivation for data compression

Table 4 presents the total time spent by the parallel heterogeneous algorithm in computations and communications when processing the hyperspectral image described in Subsection 5.1 on the parallel platform described in Section 5.2. The parameter settings for the parallel algorithm were exactly the same as those used to produce the results in Table 1 (the parallel algorithm produced exactly the same results as the sequential version). For the parallel implementation, we assumed that processor p_3 (the fastest) was always the master. Two types of computation times are analyzed in Table 4, namely, *sequential* (those performed by the root node with no other parallel tasks active in the system), and *parallel* (the rest of computations, i.e. those performed by the root node and/or the workers in parallel).

On the other hand, four types of communications are also reported in Table 4, resulting from a detailed analysis of the communications that take place in our parallel algorithm:

- Resulting from the *workload estimation* step, the master sends each slave an amount of pixel vectors that depends on the processing speed of the slave and also on the capacity of the communication links between the slave and its neighbors.
- In the *skewer generation* step, K random unit vectors with N dimensions (skewers) are broadcast to all the workers. These communications can be partially overlapped with computations in the *extreme projections* step of the parallel algorithm since there are no data dependencies involved. Specifically, each pixel vector in the local partition needs to be projected onto each skewer, and the projection of a skewer to a pixel vector is independent of the projection of the same skewer to a different pixel vector, so the *extreme projections* step can start as soon as the first skewer is received by the worker.
- In the *endmember selection* step, the workers send (in parallel) the spatial coordinates of the pixels locally selected as extremes to the master node.
- Finally, in the *spectral unmixing* step, the final set of endmembers is first broadcast to all the workers. Since each worker needs the full suite of endmembers to estimate the fractional abundances in the local pixels, these communications cannot be overlapped with computations. As a result, this step can only start after the full suite of final endmembers is received by the worker. At the end of this step, the workers send (in parallel) the fractional abundances estimated for each local pixel to the master node.

As described above, in our implementation the communications can only be partially overlapped with computations due to data dependencies. In this regard, it is important to emphasize that the communication times reported on Table 4 actually correspond to the non-overlapped portions of the different types of communications. As a result, the total execution time of the parallel

Table 4
Computation and communication times (seconds) for parallel implementation of hyperspectral processing chain on heterogeneous network (without adaptive compression).

Computations		Communications			
Sequential	Parallel	Workload estimation	Skewer generation	Endmember selection	Unmixing
13.25	61.13	6.21	4.45	0.65	2.89

heterogeneous algorithm in the considered network is the sum of all times reported on Table 4, i.e. 88.58 s, out of which 14.2 s (16.03% of the total execution time) correspond to communications that could not be overlapped with computations.

5.4. Impact of introducing adaptive data compression

Table 5 presents the total time spent by the parallel heterogeneous algorithm in computations and communications when the adaptive data compression strategy described in Section 4 was incorporated into the algorithm. Here, we considered different values of the threshold angle parameter used to control the quality of the reconstruction after data compression, ranging from $t_a=0.05$ (high spectral fidelity in the reconstruction) to $t_a=0.4$ (low spectral fidelity). According to our extensive experiments in Plaza et al. (2004), setting $t_a=0.1$ generally leads to acceptable results in many application domains. Therefore, we recommend this parameter setting in most hyperspectral imaging applications (the main feature of our proposed algorithm is that the number of decomposition levels is automatically estimated in order to satisfy this tolerance threshold in the reconstruction of the data after compression, therefore the algorithm adaptively decides the compression strategy in order to achieve the desired quality in spectral signature reconstruction). It is important to emphasize that, in our considered application, high-quality fractional abundance estimations (within ± 0.3 square meters deviation per pixel with regards to those reported in Table 1) were only achieved for values of $t_a \leq 0.1$. For values of $t_a > 0.1$, the quality of the fractional abundance estimations was significantly lower (above ± 0.9 square meters deviation per pixel with regards to those reported in Table 1). Therefore, from now on we assume that $t_a=0.1$ is the most appropriate reconstruction threshold in terms of both algorithm accuracy and parallel performance. In this case, the total execution time of the heterogeneous algorithm in the considered network was 85.28 s, out of which 7.3 s (8.56% of the total execution time) correspond to communications. If we compare these times with those obtained for the parallel heterogeneous algorithm without adaptive compression (see Table 4), we can infer that the incorporation of adaptive lossy compression reduced the communication time up to 51.4% (from 14.2 to 7.3 s) without significantly reducing sub-pixel analysis accuracy.

As presented by Table 5, the introduction of adaptive run-time compression slightly increased the *sequential* and *parallel* computation times (from 74.38 to 77.98 s for $t_a=0.1$) as a result of the encoding and decoding processes. However, the increase was not very significant, meaning that both the compression processing overhead and the CPU load in compression over various processors was very small. This is due to the high inter-correlation observed in many of the values that comprise the analyzed spectral signatures. This property, characteristic of hyperspectral analysis, tremendously facilitated the execution of the one-dimensional wavelet transform in computationally efficient terms, resulting from the good cache memory system utilization (enhanced by data locality) and the very few accesses to secondary storage. For values of $t_a \leq 0.1$, the one-dimensional wavelet typically required at most 2–3 levels of decomposition per pixel (these are adaptively selected at run-time).

Table 5
Computation and communication times (seconds) for parallel implementation of hyperspectral processing chain on heterogeneous network, with adaptive compression.

t_a	Computations		Communications			
	Sequential	Parallel	Workload estimation	Skewer generation	Endmember selection	Unmixing
0.05	15.91	62.83	3.86	2.68	0.61	1.85
0.1	15.44	62.54	3.12	2.21	0.43	1.54
0.2	14.89	62.33	2.79	1.86	0.56	1.46
0.4	14.71	62.06	2.45	1.54	0.48	1.28

In turn, the use of lossy compression resulted in a significant decrease of communication times, in particular, of those involved in the *workload estimation* (distribution of pixel vectors by the master to the slave processors) and *skewer generation* (broadcast of skewers by the master processor) steps. The communication times were also decreased for the *spectral unmixing* step (broadcast of final endmembers by the master processor). Since the *endmember selection* step only involved communicating the spatial coordinates of a few pixels selected as extreme by each worker, these times always remained very low. Another reason why the *sequential* and *parallel* computation times were not significantly increased was that some of the adaptive compression operations (e.g. returning the local results obtained by the workers to the master in the *spectral unmixing* step) could also be performed in parallel.

To conclude our study, we have analyzed the scalability of the parallel heterogeneous algorithm (with and without adaptive data compression) on the fully heterogeneous network. Here, the concept of scalability used is the one addressed in Llorente et al. (1996) and Grama et al. (2003), i.e. we seek to reduce communication time in order to improve scalability since we have experimentally observed that parallel performance is maintained by scaling the size of the problem (e.g. using larger or smaller hyperspectral data sets). Fig. 4 shows the performance gain of the heterogeneous parallel algorithm (implemented using different values of t_a) as the number of processors was increased on the heterogeneous cluster. Here, we assumed that processor p_3 (the fastest) was always the master, and varied the number of slaves. The construction of speedup plots in heterogeneous environments is not straightforward, mainly because the workers do not have the same relative speed, and therefore the order in which they are added to plot a speedup curve needs to be further analyzed. To substantiate the impact of the order of selection of slaves when constructing the speedup plots, we have tested three different strategies:

(1) First, we used an ordering strategy based on the relative speed of processors in Table 2, i.e., the first case study tested (labeled as '2 processors') consisted of using processor p_3 (the fastest) as the master and processor p_{10} (i.e., the one with lowest relative speed) as the slave; the second case tested (labeled as '3 processors') consisted of using processor p_3 as the master and processors p_{10} and p_{11} (i.e., the two processors with lowest relative speed) as slaves, and so on, until a final case (labeled as '16 processors') was tested, based on using processor p_3 as the master and all remaining 15 processors as slaves.
(2) Second, we used an ordering strategy based on the inverse relative speed of processors in Table 2, i.e., the first case study

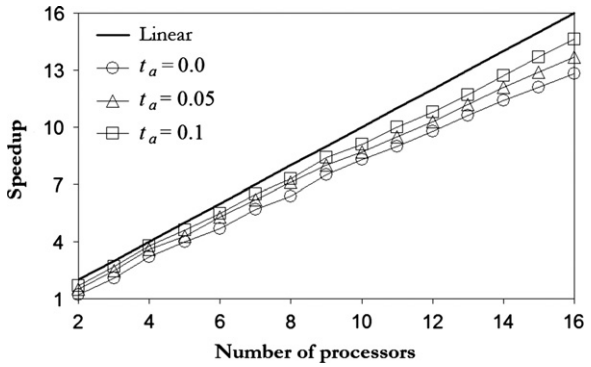

Fig. 4. Performance gain of heterogeneous parallel algorithm, with ($t_a > 0.0$) and without ($t_a = 0.0$) adaptive compression, as number of processors was increased on the heterogeneous network.

tested (labeled as '2 processors') consisted of using processor p_3 (the fastest) as the master and processor p_{10} (i.e., the one with lowest relative speed) as the slave; the second case tested (labeled as '3 processors') consisted of using processor p_3 as the master and processor p_4 (i.e., the second fastest processor) as slave, and so on, until a final case (labeled as '16 processors') was tested, based on using processor p_3 as the master and all remaining 15 processors as slaves.

(3) Finally, we also used a random ordering strategy, i.e., the first case study tested (labeled as '2 processors') consisted of using processor p_3 as the master and a different processor, selected randomly among the remaining processors (say, processor p_i) as the slave; the second case (labeled as'3 processors') consisted of using processor p_3 as the master, processor p_i as the first slave, and a different processor, selected randomly among the remaining processors, as the second slave, and so on, a final case was tested (labeled as '16 processors'), based on using processor p_3 as the master and all remaining 15 processors as slaves.

Since the three tested strategies resulted in very similar speedup curves, we report only one case (corresponding to the first ordering strategy described above) in Fig. 4 for space considerations. The other two strategies, based on the inverse relative speed of processors and random ordering strategy provided very similar results, thus indicating that the proposed parallel implementation scales well regardless of the ordering in which heterogeneous processors are included in order to plot the speedup curve. This fact also reveals that the properties of the heterogeneous communication network are well captured by the parallel implementation, since the processors are communicated via heterogeneous communication links and their relative ordering when plotting the speedup curve is not affected by the heterogeneous communication links.

As shown by Fig. 4, the incorporation of additional processing nodes provided better speedups as the value of t_a was increased, regardless of the relative speed of the nodes. This indicated that the use of adaptive wavelet-based data compression at run-time can increase the scalability of the code by intelligently reducing the information to be transmitted through the heterogeneous network. For instance, the measured speedup improved from 12.82 to 14.63 (for 16 processors) when $t_a = 0.1$, the best observed compromise between sub-pixel analysis accuracy and parallel efficiency. Although better speedup values can be achieved by using values of $t_a > 0.1$, we do not display them in Fig. 4 because the quality of the resulting fractional abundance estimations in those cases is degraded.

Finally, it should be noted that the experimental results in this work have been measured on a distributed network of workstations with slow communication lines. This is mainly because we are interested in analyzing the performance of our parallel algorithms in this type of systems, which are still widely used in relevant agencies and organizations. For instance, the considered parallel system specifications are similar to those used at NASA Jet Propulsion Laboratory's AVIRIS data facility, which currently holds the most relevant repository of high-quality hyperspectral data sets. Of course, upgrading the hardware (e.g. via Gigabit Ethernet cards) could lead to improved parallel performance results. However, we have decided to report results on a parallel platform with slower communication links on purpose, thus trying to address the characteristics of current systems in many international agencies and organizations involved in hyperspectral data processing. In this regard, although the results reported in this work are encouraging, further research is required in order to fully optimize the proposed implementations and extrapolate these results to different heterogeneous parallel environments (e.g. with superior parallel performance) and analysis scenarios.

6. Conclusions and future lines

In heterogeneous networks, the slowness of communication links can be a bottleneck in order to achieve good scalability of parallel applications. A possible method to overcome this issue is to intelligently compress the information to be transmitted through the communication network. If the adopted compression strategy is lossy, then higher compression ratios can be achieved. However, in this case it is very important to substantiate the trade-off between the increase in parallel efficiency and the potential decrease in algorithm accuracy resulting from such loss of information.

In this paper, we have studied the impact of incorporating a wavelet-based lossy compression strategy at run-time in order to reduce the amount of information that is communicated in a parallel and heterogeneous remote sensing application. The proposed approach adaptively determines the most appropriate level of compression in order to avoid degrading the spectral signature associated to each pixel vector in the remotely sensed scene, thus ensuring that the most relevant information for analyzing the data with sub-pixel precision is preserved. Our experimental results, obtained in the context of a fully heterogeneous network of workstations, show that the incorporation of adaptive lossy compression can reduce the communications up to 51.4% without significantly degrading the sub-pixel analysis of a remotely sensed hyperspectral scene. The main benefit of our proposed approach can be summarized as follows: with only a little increase in computation time (associated to the accommodation of the – necessary but fast – compression and decompression operations in the parallel heterogeneous algorithm) the parallel performance of the implementation increases significantly since the amount of data to be communicated throughout the communication network is much smaller. This is a crucial aspect for improving parallel performance of hyperspectral imaging applications, due to the high dimensionality of the data.

In the future, we would like to study the effect of performing the data compression and decompression on the programmable processor available on the network interface, so that these overheads can be eliminated from the host processors. We would also like to investigate performance issues when there are more processes running on each node. Implementation of the proposed framework on specialized hardware devices such as field

programmable gate arrays (FPGAs) rather than desktop computers is also an important extension of this work to be approached in future developments. Finally, we also feel that the applicability of the proposed strategy extends beyond the domain of remotely sensed hyperspectral image processing. This is particularly true for the domains of signal processing and linear algebra applications (Lastovetsky and Reddy, 2007; Benner et al., 2008), which include similar patterns of communication and calculation.

Acknowledgements

This work has been supported by the European Community's Marie Curie Research Training Networks Programme under Reference MRTN-CT-2006-035927, Hyperspectral Imaging Network (HYPER-I-NET). Funding from the Spanish Ministry of Science and Innovation (HYPERCOMP/EODIX Project, Reference AYA2008-05965-C04-02) and funding from Junta de Extremadura (Project Reference: PRI09A110) are also gratefully acknowledged. Last but not least, we would like to take this opportunity to gratefully thank the two anonymous reviewers for their comments and suggestions, which greatly helped us to improve the quality and presentation of our manuscript.

References

Adams, J.B., Smith, M.O., Johnson, P.E., 1986. Spectral mixture modeling: a new analysis of rock and soil types at the viking lander 1 site. Journal of Geophysical Research 91, 8098–8112.

Benner, P., Quintana, E.S., Quintana, G., 2008. Solving linear-quadratic optimal control problems on parallel computers. Optimization Methods & Software 23, 879–909.

Biehl, L., Landgrebe, D., 2002. MultiSpec—a tool for multispectral hyperspectral image data analysis. Computers & Geosciences 28, 1153–1159.

Boardman, J.W., 1993. Automating spectral unmixing of AVIRIS data using convex geometry concepts. In: Summaries of Airborne Earth Science Workshop, JPL(Jet Propulsion Laboratory) Publication 93-26, Pasadena, pp. 11–14.

Chang, C.-I., 2003. Hyperspectral Imaging: Techniques for Spectral Detection and Classification. Kluwer, New York, p. 390.

Dai, H., 2005. Parallel processing of prestack Kirchhoff time migration on a PC cluster. Computers & Geosciences 31, 891–899.

Du, Q., Chang, C.-I., 2004. Linear mixture analysis-based compression for hyperspectral image analysis. IEEE Transactions on Geoscience and Remote Sensing 42, 875–891.

Du, Q., Zhu, W., Yang, H., Fowler, J.E., 2009. Segmented principal component analysis for parallel compression of hyperspectral imagery. IEEE Geoscience and Remote Sensing Letters 6, 713–717.

Elsasser, R., Monien, B., Preis, R., 2002. Diffusion schemes for load balancing on heterogeneous networks. Theory of Computing Systems 35, 305–320.

Goetz, A.F.H., Vane, G., Solomon, J.E., Rock, B.N., 1985. Imaging spectrometry for Earth remote sensing. Science 228, 1147–1153.

Grama, A., Gupta, A., Karypis, G., Kumar, V., 2003. Introduction to Parallel Computing, second ed. Addison Wesley, Essex, England, p. 312.

Green, R.O., Eastwood, M.L., Sarture, C.M., Chrien, T.G., Aronsson, M., Chippendale, B.J., Faust, J.A., Pavri, B.E., Chovit, C.J., Solis, M., Olah, M.R., Williams, O., 1998. Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS). Remote Sensing of Environment 65, 227–248.

Kaewpijit, S., Le Moigne, J., El-Ghazawi, T., 2003. Automatic reduction of hyperspectral imagery using wavelet spectral analysis. IEEE Transactions on Geoscience and Remote Sensing 41, 863–871.

Keshava, N., Mustard, J.F., 2002. Spectral unmixing. IEEE Signal Processing Magazine 19, 44–57.

Kumar, V.S., Nanjundiah, R., Thazhuthaveetil, M.J., Govindarajan, R., 2008. Impact of message compression on the scalability of an atmospheric modeling application on clusters. Parallel Computing 34, 1–16.

Lastovetsky, A., 2003. Parallel Computing on Heterogeneous Networks. Wiley Interscience, New York, p. 380.

Lastovetsky, A., Reddy, R., 2007. Data distribution for dense factorization on computers with memory heterogeneity. Parallel Computing 33, 757–779.

Legrand, A., Renard, H., Robert, Y., Vivien, F., 2004. Mapping and load-balancing iterative computations. IEEE Transactions on Parallel and Distributed Systems 15, 546–558.

Llorente, I.M., Tirado, F., Vazquez, L., 1996. Some aspects about the scalability of scientific applications on parallel architectures. Parallel Computing 22, 1169–1195.

Motta, G., Rizzo, F., Storer, J.A., 2006. Hyperspectral Data Compression. Springer, Berlin, p. 480.

Penn, B.S., 2002. Using simulated annealing to obtain optimal linear end-member mixtures of hyperspectral data. Computers & Geosciences 28, 809–817.

Plaza, A., 2008a. Parallel processing of remotely sensed hyperspectral imagery: full-pixel versus mixed-pixel classification. Concurrency and Computation: Practice & Experience 20, 1539–1572.

Plaza, A., 2008b. Parallel techniques for information extraction from hyperspectral imagery using heterogeneous networks of workstations. Journal of Parallel and Distributed Computing 68, 93–111.

Plaza, A., Chang, C.-I., 2007. High Performance Computing in Remote Sensing. Chapman & Hall, CRC Press, Boca Raton, p. 450.

Plaza, A., Martinez, P., Perez, R., Plaza, J., 2002. Spatial/spectral endmember extraction by multidimensional morphological operations. IEEE Transactions on Geoscience and Remote Sensing 40, 2025–2041.

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

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

Plaza, A., Valencia, D., Plaza, J., 2008. An experimental comparison of parallel algorithms for hyperspectral analysis using homogeneous and heterogeneous networks of workstations. Parallel Computing 34, 92–114.

Ramakhrishna, B., Plaza, A., Chang, C.-I., Ren, H., Du, Q., Chang, C.-C., 2006. Spectral/spatial hyperspectral image compression. In: Motta, G., Rizzo, F., Storer, J.A. (Eds.), Hyperspectral Data Compression. Springer, Berlin, pp. 309–346.

Sourbier, F., Operto, S., Virieux, J., Amestoy, P., L'Excellent, J.-Y., 2009. FWT2D: a massively parallel program for frequency-domain full-waveform tomography of wide-aperture seismic data. Part 2: numerical examples and scalability analysis. Computers & Geosciences 35, 496–514.

Tian, Y., Peters-Lidard, C.D., Kumar, S.V., Geiger, J., Houser, P.R., Eastman, J.L., Dirmeyer, P., Doty, B., Adams, J., 2008. High-performance land surface modeling with a linux cluster. Computers & Geosciences 34, 1492–1504.

Zeng, X., McMechan, G.A., 2002. Load balancing across a highly heterogeneous processor cluster using file status probes. Computers & Geosciences 28, 911–918.