Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 34 (2008) 92–114
PARALLEL COMPUTING
www.elsevier.com/locate/parco

An experimental comparison of parallel algorithms for hyperspectral analysis using heterogeneous and homogeneous networks of workstations

Antonio Plaza *, David Valencia, Javier Plaza

Department of Computer Science, Computer Architecture and Technology Section, University of Extremadura, Avda. de la Universidad s/n, E-10071 Caceres, Spain

Received 5 November 2005; received in revised form 10 May 2007; accepted 20 December 2007
Available online 18 January 2008

Abstract

Imaging spectroscopy, also known as hyperspectral imaging, is a new technique that has gained tremendous popularity in many research areas, including satellite imaging and aerial reconnaissance. In particular, NASA is continuously gathering high-dimensional image data from the surface of the earth with hyperspectral sensors such as the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) or the Hyperion hyperspectral imager aboard NASA's Earth Observing-1 (EO-1) spacecraft. Despite the massive volume of scientific data commonly involved in hyperspectral imaging applications, very few parallel strategies for hyperspectral analysis are currently available, and most of them have been designed in the context of homogeneous computing platforms. However, heterogeneous networks of workstations represent a very promising cost-effective solution that is expected to play a major role in the design of high-performance computing platforms for many on-going and planned remote sensing missions. Our main goal in this paper is to understand parallel performance of hyperspectral imaging algorithms comprising the standard hyperspectral data processing chain (which includes pre-processing, selection of pure spectral components and linear spectral unmixing) in the context of fully heterogeneous computing platforms. For that purpose, we develop an exhaustive quantitative and comparative analysis of several available and new parallel hyperspectral imaging algorithms by comparing their efficiency on both a fully heterogeneous network of workstations and a massively parallel homogeneous cluster at NASA's Goddard Space Flight Center in Maryland.

Keywords: Parallel algorithm design; Heterogeneous computing; Hyperspectral image analysis; Hyperspectral data processing chain; Load balance

1. Introduction

Recent advances in sensor technology have led to the development of so-called hyperspectral imagers, capable of collecting hundreds of images, corresponding to different wavelength channels, for the same area on the

* Corresponding author. Tel.: +34 927 257195; fax: +34 927 257203.
E-mail address: aplaza@unex.es (A. Plaza).

0167-8191/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2007.12.005

surface of the earth [1]. The concept of hyperspectral imaging (also known as imaging spectroscopy) has been historically linked to one of NASA's premier instruments for earth exploration, the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) system [2]. As illustrated by Fig. 1, hyperspectral imagers such as AVIRIS are able to measure reflected radiation in the wavelength region from 0.4 to 2.5 μm using 224 spectral channels, at a nominal spectral resolution of 10 nm. On the other hand, the Hyperion hyperspectral imager aboard NASA's Earth Observing-1 (EO-1) spacecraft has been NASA's first hyperspectral imager to become operational on-orbit. It routinely collects images hundreds of kilometers long with 220 spectral bands in the same spectral range described above. The incorporation of latest-generation sensors such as AVIRIS or Hyperion to airborne and satellite platforms is currently producing a nearly continual stream of high-dimensional data, and this explosion in the amount of collected information has rapidly introduced new processing challenges (it is estimated that NASA collects and sends to earth more than 950 GB of hyperspectral data every day). In particular, the price paid for the wealth of spatial and spectral information available from hyperspectral sensors is the enormous amounts of data that they generate. As a result, the automation of techniques able to transform massive amounts of hyperspectral data into scientific understanding in valid response times is critical for space-based earth science and planetary exploration.

To address the computational requirements introduced by hyperspectral imaging applications, several efforts have been directed towards the incorporation of high performance computing models in remote sensing missions [3–5]. 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 [6,7]. The processing power offered by such commodity systems has been traditionally employed in data mining applications from very large data archives. Although most parallel techniques and systems for image information processing and mining employed by NASA and other institutions during the last decade have been chiefly homogeneous in nature, a current trend in the design of systems for the analysis and interpretation of massive volumes of data, resulting from space-based earth science and planetary exploration missions, relies on the utilization of highly heterogeneous computing platforms [8]. Unlike traditional homogeneous systems, heterogeneous networks are composed of processors running at different speeds. As a result, traditional parallel hyperspectral imaging algorithms, which

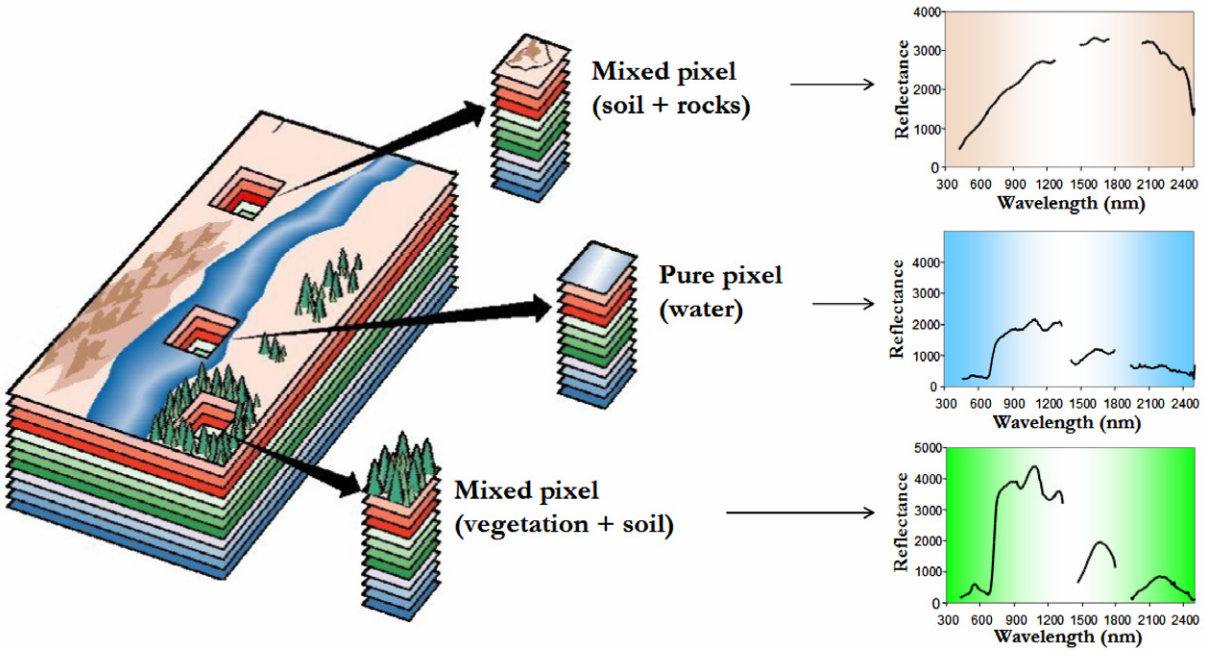


Fig. 1. Concept of hyperspectral imaging using NASA Jet Propulsion Laboratory's AVIRIS sensor.

distribute computations evenly across the different processors, cannot balance the load of different-speed processors in heterogeneous networks as faster processors will quickly perform their portions of computation and will have to wait for slower ones at points of synchronization. Therefore, a natural solution to the problem of heterogeneous computing is to distribute data across processors unevenly, so that each processor performs the volume of computation proportional to its speed. Heterogeneous computing research [9–11] has shown that, with careful job scheduling, off-the-shelf heterogeneous clusters can realize a very high level of aggregate performance. Subsequently, it is expected that these clusters will represent a tool of choice for the scientific community devoted to high-dimensional data analysis in remote sensing and other fields. Due to the recent incorporation of heterogeneous computing to remote sensing-based research [8], significant opportunities to exploit such techniques are available in the analysis of hyperspectral data sets.

The main objective of this paper is to perform an exhaustive comparison, in terms of both parallel efficiency and data interpretation accuracy, of parallel hyperspectral analysis algorithms implemented in both heterogeneous and homogeneous platforms. The remainder of the paper is structured as follows. Section 2 introduces the standard data processing chain used for the interpretation of hyperspectral image data, and describes the impact of data partitioning strategies on the design of parallel techniques utilized by the considered chain. Section 3 describes several parallel hyperspectral algorithms specifically designed to be run in heterogeneous networks of computers. Section 4 assesses the performance of heterogeneous algorithms by evaluating their efficiency on both a fully heterogeneous cluster of workstations at University College Dublin, and a large-scale homogeneous cluster at NASA's Goddard Space Flight Center. Section 5 provides a summary of contributions and hints at plausible future research.

2. Hyperspectral imaging techniques and data partitioning

The underlying assumption governing techniques for the analysis of hyperspectral data is that each spectral signature measures the response of multiple underlying materials at each site [1]. For instance, the pixel vector labeled as "vegetation + soil" in Fig. 1 is actually a mixed pixel which comprises a mixture of vegetation and soil, or different types of soil and vegetation canopies. This situation, often referred to as the "mixture problem" in hyperspectral analysis terminology, is one of the most crucial and distinguishing properties of imaging spectroscopy analysis. Mixed pixels exist for one of two reasons. Firstly, if the spatial resolution of the sensor is not high enough to separate different materials, these can jointly occupy a single pixel, and the resulting spectral measurement will be a composite of the individual spectra. Secondly, mixed pixels can also result when distinct materials are combined into a homogeneous mixture. This circumstance occurs independent of the spatial resolution of the sensor. A hyperspectral image (also referred to as "image cube") is often a combination of the two situations, where a few sites in a scene are pure materials, but many other are mixtures of materials.

To deal with the mixture problem in hyperspectral imaging, spectral unmixing [12] has been proposed as a consolidated analysis procedure in which the measured spectrum of a mixed pixel is decomposed into a collection of spectrally pure constituent spectra, called *endmembers* in the literature, and a set of correspondent fractions or *abundances* that indicate the proportion of each endmember present in the mixed pixel [13]. The procedure for spectral unmixing is given by a well-established hyperspectral data processing chain (schematically described by a diagram in Fig. 2), in which the dimensionality of the input hyperspectral image is first (optionally) reduced and then pure spectral endmembers are automatically extracted from the scene. The last step of the data processing chain is the decomposition of mixed pixels into fractional abundance maps through a linear inversion procedure [1].

During the last decade, several algorithms have been proposed for the purpose of automatically extracting spectral endmembers from hyperspectral image data. One of the most successful algorithms has been the pixel purity index (PPI), originally developed by Boardman et al. [14] and available from Kodak's Research Systems ENVI, one of the most widely used commercial software packages by remote sensing scientists around the world. This algorithm first reduces the dimensionality of the input data and then proceeds by generating a large number of random, N-dimensional (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. Fig. 3 shows a toy example (shown in a 2-D space for simplicity)

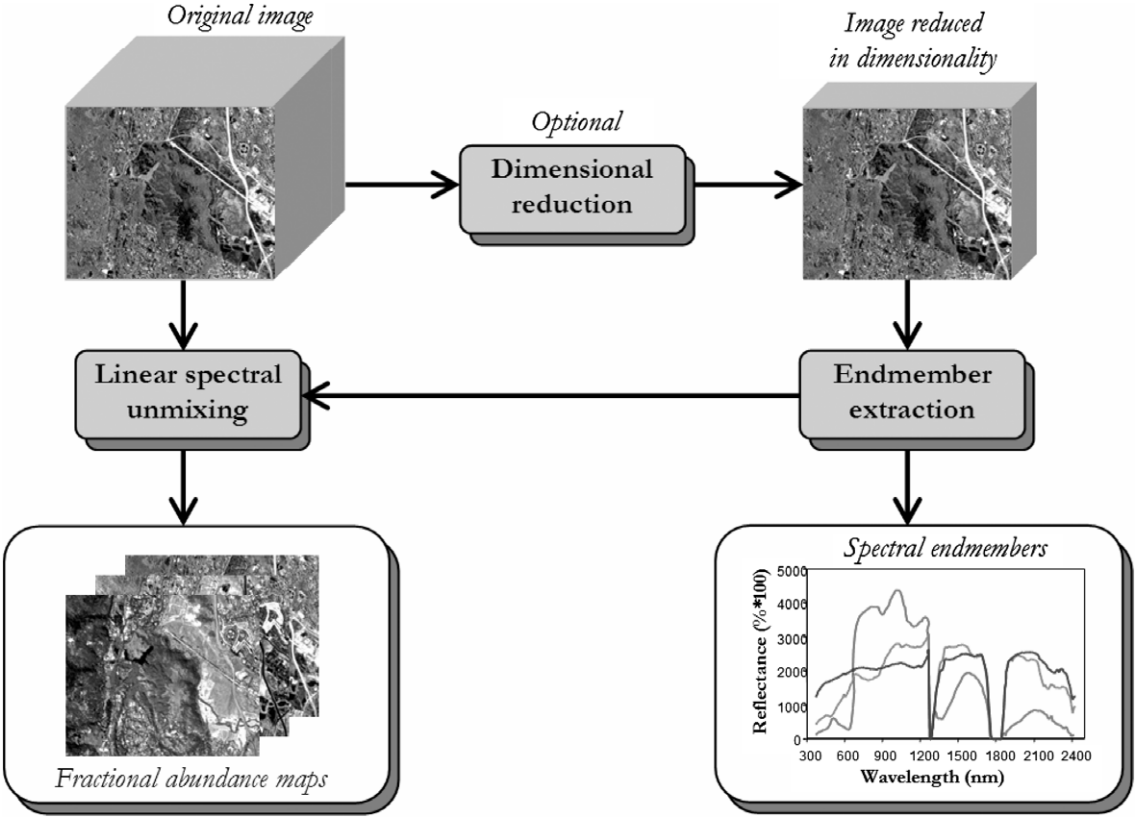


Fig. 2. Diagram representing the standard hyperspectral data processing chain.

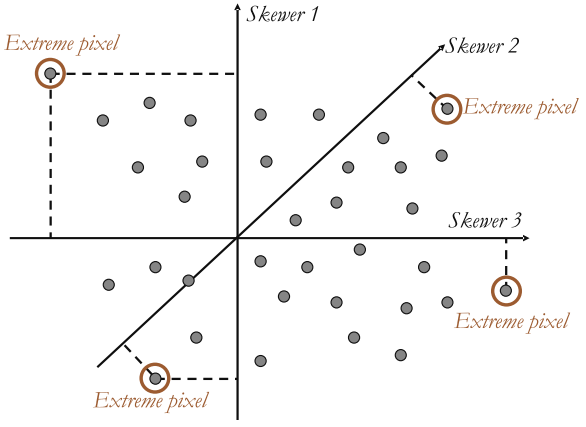
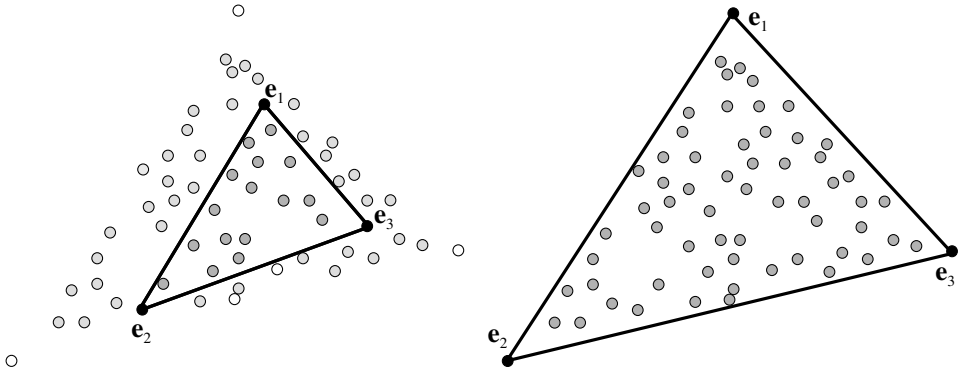


Fig. 3. Toy example illustrating the performance of the PPI algorithm in a 2-D space.

illustrating how skewer projections allow finding extreme pixels in the data cloud. 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.

Another standard technique is the N-FINDR algorithm [15], which aims at identifying the set of pixels which define the simplex with the maximum volume, potentially inscribed within the dataset. After a previous



(a) Initial volume estimation using random pixels (b) Final volume estimation and endmembers

Fig. 4. Toy example illustrating the concept of the N-FINDR endmember algorithm in a 2-D space.

dimensionality reduction step, a set of pixel vectors is first randomly selected, and their corresponding volume is calculated. For illustrative purposes, Fig. 4a shows an example of the above situation, in which 3 randomly selected endmembers (represented as black circles) define a volume which can be used to expressed mixed pixels included in the volume (represented as gray circles) in terms of linear combinations of endmembers. As shown by Fig. 4a, many hyperspectral pixel vectors (represented as white circles) may remain unexplained after the initial random selection. In order to refine the initial estimate and improve the endmember selection procedure, 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. As shown by Fig. 4b, the endmembers obtained at the end of this process will likely define a simplex which encloses most of the pixels in the input hyperspectral data set.

The iterative error analysis (IEA) algorithm [16] can be considered a variation of the N-FINDR algorithm which aims at obtaining the final estimate through an iterative procedure. Here, a single pixel vector (usually the mean spectrum of the data) is chosen to start the process. A linear spectral unmixing in terms of this vector is then performed as described in the standard data processing chain shown in Fig. 2, and an error image, formed by the errors remaining at each pixel after a linear spectral unmixing operation, is then calculated. The spectral vector corresponding to the pixel with the single largest error after the abundance estimation process is labeled as the first endmember. A new linear spectral unmixing operation in terms of this vector is used to find a second endmember, a third endmember, and so on, until a predefined number of endmembers is obtained.

Finally, the automated morphological endmember extraction (AMEE) [17] is the only available technique that integrates the spatial and the spectral information in the search for spectral endmembers. The method is based on the utilization of a kernel or *structuring element* [18] that is moved through all the pixels of the image, defining a spatial search area around each pixel vector. The spectrally purest and the spectrally most highly mixed spectral signatures are respectively obtained at the neighborhood of each pixel by calculating the spectral angle between the purest and most highly mixed pixels. Fig. 5 shows a toy example illustrating the performance of the two morphological operations considered for the selection of pure/mixed pixels. As shown by Fig. 5, morphological dilation expands the purest areas in the scene by selecting the most highly pure pixels (represented as white and dark pixels in Fig. 5) in the structuring element neighborhood. Quite opposite, morphological erosion expands the most highly pure areas in the hyperspectral image by selecting the most highly mixed pixels (represented as gray pixels in Fig. 5) in the same neighborhood. These operations are repeated for all the pixels in the scene until a *morphological eccentricity index* (MEI) score is generated for each pixel vector. The pixels with highest associated MEI scores are assumed to be the final endmembers.

It should be noted that both the identification of image endmembers and the subsequent unmixing process are computationally demanding problems. However, very few research efforts devoted to the design of parallel

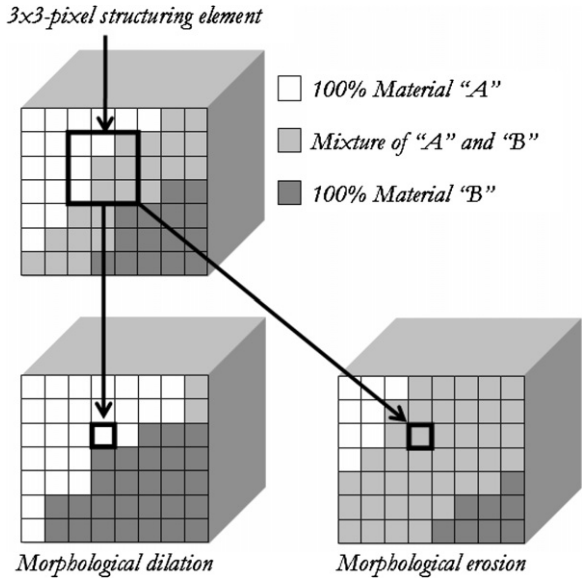


Fig. 5. Performance of morphological dilation and erosion operations used by the AMEE morphological endmember extraction algorithm.

implementations currently exist in the open literature. In particular, some existing parallel hyperspectral imaging techniques are subject to non-disclosure restrictions, mainly due to their use in military and defense applications. Nevertheless, with the recent explosion in the amount and dimensionality of hyperspectral imagery, parallel processing is expected to become a requirement in virtually every remote sensing application. To address this issue, this paper takes a necessary first step toward the comparison of different techniques and strategies for parallel hyperspectral image analysis on heterogeneous platforms.

It is important to reiterate that spectral mixture analysis techniques for hyperspectral imaging focus on analyzing the data based on the properties of spectral signatures, i.e., they utilize the information provided by each pixel vector *as a whole*. This consideration has a significant impact on the design of data partitioning strategies for parallelization. In particular, it has been shown in the literature that domain decomposition techniques provide flexibility and scalability in parallel image processing [19–21]. In hyperspectral imaging, two types of partitioning can be exploited: spectral-domain partitioning and spatial-domain partitioning. Spectral-domain partitioning subdivides the volume into sub-volumes made up of contiguous spectral bands

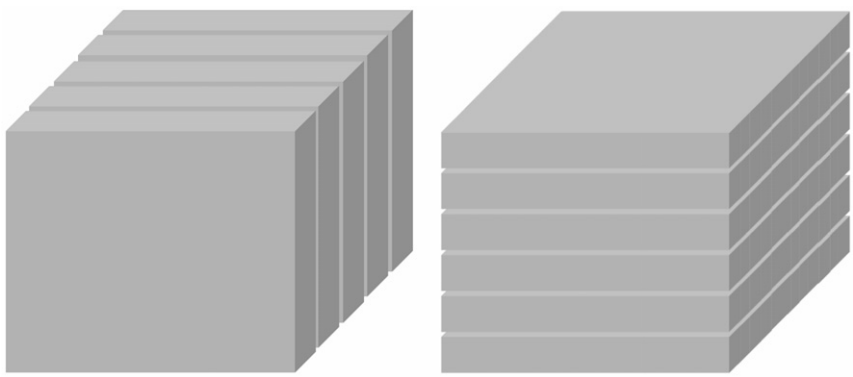


Fig. 6. The two main approaches to volume data partitioning in hyperspectral imaging applications.

(see Fig. 6a), and assigns one or more sub-volumes to each processor. With this model, each pixel vector may be split among several processors and the communication cost associated to the computations based on spectral signatures would be increased [22]. In order to exploit parallelism as much as possible, we have adopted a spatial-domain partitioning approach (see Fig. 6b), in which the data is partitioned in slabs which retain the full spectral information. There are several reasons that justify our decision to incorporate spatial-domain partitioning techniques in our application:

1. First and foremost, spatial-domain partitioning is a natural approach for low-level image processing as many image processing operations require the same function to be applied to a small set of elements around each entire pixel vector in the image volume.
2. A second reason is that, in spectral-domain partitioning, the calculations made for each pixel vector need to originate from several processors and thus require intensive inter-processor communication. This is generally perceived as a shortcoming for parallel design, because the overhead introduced by inter-processor communication would increase linearly with the increase in the number of processing elements, thus complicating the design of parallel algorithms (in particular, in heterogeneous environments [23]).
3. A final major issue is code reusability; to reduce code redundancy and enhance portability, it is desirable to reuse much of the code for the sequential algorithm in the design of its correspondent parallel version and the spatial-domain approach greatly enhances code reuse [22].

As will be shown in the following section, all the parallel algorithms developed in this paper are designed under the assumption that each pixel vector is uniquely represented by its associated spectral signature. Therefore, the introduction of a spectral-domain-based decomposition approach would require additional strategies to combine the partial results from several processing elements.

3. Parallel hyperspectral imaging algorithms

This section describes the parallel algorithms that will be compared in this study. Before introducing the algorithm descriptions, we first formulate a general optimization problem in the context of fully heterogeneous systems. We assume that processing elements in the system can be modeled as a set of computing resources $P = \{p_i\}_{i=1}^{|P|}$, where $|P|$ denotes the total number of processors in the system, and each processor is weighted by its relative speed w_i [23]. In order to estimate processor relative speeds, we use a representative benchmark function which makes use of a core computation directly linked to the considered application domain. We also denote by W the total workload to be performed by a certain hyperspectral imaging algorithm. Such workload depends on the considered algorithm. Since most processing algorithms in hyperspectral imaging applications involve repeated vector product operations, we can measure the workload involved by each considered algorithm in terms of elementary multiplication/accumulation (MAC) operations. With the above assumptions in mind, processor p_i will accomplish a share of $\alpha_i \times W$ of the total workload executed by a certain algorithm, where $\alpha_i \geqslant 0$ for $1 \leqslant i \leqslant |P|$ and $\sum_{i=1}^{|P|} \alpha_i = 1$. An abstract view of the problem can be simply stated in the form of a client–server architecture [24,25], in which a *server* node is responsible for the distribution of work among the $|P|$ nodes, and the *client* nodes operate with the spectral signatures contained in a local partition. The local partitions are updated locally and, depending on the algorithm under consideration, the resulting calculations may also be exchanged between the client processors, or between the server and the clients. The general sequence of operations executed by our server program is summarized below:

Algorithm 1. General server program (GSP)

Input: N-D data cube **F**.
Output: Set of $|P|$ spatial-domain heterogeneous partitions of **F**.

1. Generate necessary system information, including the number of available processors in the system, $|P|$, each processor's $\{p_i\}_{i=1}^{|P|}$ identification number, processor relative speeds $\{w_i\}_{i=1}^{|P|}$, and other data.
2. Set $\alpha_i = \left(w_i / \sum_{i=1}^{|P|} w_i\right)$ for all $i \in \{1, \cdots, |P|\}$.

3. Use $\{\alpha_i\}_{i=1}^{|P|}$ to obtain a set of $|P|$ spatial-domain heterogeneous partitions of $\mathbf{F}$, so that the spectral channels corresponding to the same pixel vector are never stored in different partitions. In order to perform the spatial-domain data partitioning described above, we have adopted a simple methodology which consists of two main steps:
   (a) The hyperspectral data set is partitioned, using spatial-domain decomposition, into a set of vertical slabs which retain the full spectral information at the same partition (see Fig. 6b). The number of rows in each slab is considered to be 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 at the considered processor.
   (b) For each processor, 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, we solve the partitioning problem of a set with remaining pixel vectors over the remaining processors. We recursively apply this procedure until all the pixel vectors in the input data have been assigned.

It should be noted that, with the proposed algorithm description, it is possible that all processors exceed their upper bounds. This situation was never observed in our previous experiments on parallel processing of standard hyperspectral data sets using different platforms [22]. However, if the considered network includes processing units with low memory capacity, this situation could be handled by allocating an amount of data equal to the upper bound to those processors, and then processing the remaining data as an offset in a second algorithm iteration.

In the following, we particularize the client–server architecture described above for each specific parallel method by considering three algorithm sub-categories: dimensionality reduction, endmember extraction and spectral unmixing, where dimensionality reduction is employed by some endmember extraction algorithms (PPI, N-FINDR) to reduce the hyperspectral data volume prior to processing.

3.1. Parallel dimensionality reduction algorithm

In this section, we describe a dimensionality reduction method which is based on the principal component transform (PCT), often used in hyperspectral analysis to summarize and decorrelate the images by reducing redundancy and packing the residual information into a small set of images, termed *principal components*. PCT is a highly compute-intensive algorithm amenable to parallel implementation. Here, we describe an algorithm which is a variant of singular value decomposition of principal components [26].

Algorithm 2. Parallel PCT-based dimensionality reduction (P-PCT)

Input: N-D data cube $\mathbf{F}$, Number of spectral bands to be retained, E.
Output: E-D data cube $\mathbf{G}$.

1. Divide the original data cube $\mathbf{F}$ into $|P|$ heterogeneous partitions using the GSP algorithm, where P is the number of workers.
2. Calculate the N-D mean vector $\bar{\mathbf{f}}$ concurrently, where each component is the average of the pixel values of each spectral band of the input data. This vector is formed at the master once all the workers have finished their parts respective. The total workload involved by this step is $M \times N$, where M is the number of pixels in the hyperspectral image and N is the number of spectral bands.
3. Broadcast vector $\bar{\mathbf{f}}$ to all workers so that each worker p_i computes the local covariance component using $\left(\frac{1}{M-1}\right)\sum_{(x,y)\in\mathbf{F}_i}(\mathbf{f}(x,y)-\bar{\mathbf{f}})\cdot(\mathbf{f}(x,y)-\bar{\mathbf{f}})^{\mathrm{T}}$, where the superscript "T" denotes the matrix transpose operation and $\mathbf{F}_i$ denotes the spatial-domain partition of $\mathbf{F}$ allocated to by the GSP algorithm.
4. The master calculates the covariance matrix using the local matrices calculated in Step 3. It should be noted that Steps 4 and 5 involve M matrix multiplications and a total workload of $M \times N^2$.
5. Obtain a transformation matrix $\mathbf{T}$ by calculating and sorting the eigenvectors of the covariance matrix according to their 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 done sequentially at the master (eigenvectors calculation involves a workload of N^3 [27]).

6. Transform each N-D pixel vector in the original image by using $\mathbf{g}(x,y) = \mathbf{T} \cdot [\mathbf{f}(x,y) - \bar{\mathbf{f}}]$. This step is done in parallel, i.e., all workers transform their respective data portions. The linear transformation process is performed over all the pixels in $\mathbf{F}$ and each computation on a pixel involves a matrix multiplication. Thus, the workload of this step is $M^2 \times N^2$. The results are sent to the master, which retains the first E components of the resulting data cube $\mathbf{G}$ for subsequent processing.

3.2. Parallel endmember extraction algorithms

Three main classes of endmember extraction algorithms are considered in this section [13]: convex geometry, constrained error minimization, and integrated spatial/spectral developments. Two parallel algorithms have been developed to represent the first category: PPI and N-FINDR. Both of them incorporate a previous PCT-based dimensionality reduction step. The second class of algorithms is represented by a parallel version of the IEA algorithm. Finally, a parallel version of the AMEE algorithm is also presented and discussed. This algorithm integrates the spatial and spectral information as opposed to the other discussed algorithms, a feature that introduces additional considerations for its parallelization.

3.2.1. Parallel implementation of pixel purity index algorithm (P-PPI)

The P-PPI algorithm consists of two parallel stages applied sequentially, i.e., PCT-based dimensionality reduction and pixel purity index-based pre-processing. The latter stage is described below:

Algorithm 3. Parallel pixel purity index pre-processing (P-PPI)

Input: E-D cube $\mathbf{G}$, Number of skewers J, Threshold value T.
Output: Set of E final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.

1. Divide the original data cube $\mathbf{G}$ into $|P|$ heterogeneous partitions using the GSP algorithm, where $|P|$ is the number of workers.
2. Generate a set of J random unit N-D vectors called "skewers," denoted by $\{\mathbf{skewer}_j\}_{j=1}^{J}$, and broadcast the entire set to all the workers.
3. For each $\mathbf{skewer}_j$, project all the data sample vectors at each local partition G_i, with $i = 1, \cdots, |P|$, onto $\mathbf{skewer}_j$ using the expression $|\mathbf{skewer}_j \cdot \mathbf{g}(x,y)| = \sum_{k=1}^{E} \mathbf{skewer}(x,y)^{(k)} \times \mathbf{g}(x,y)^{(k)}$, where the superscript "(k)" denotes the k-th spectral component. This allows us to find pixel vectors at extreme positions and form an extreme set for $\mathbf{skewer}_j$, denoted $S^{(i)}(\mathbf{skewer}_j)$. Then, we 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 $N_{\mathrm{PPI}}^{(i)}[\mathbf{g}(x,y)] = \sum_j I_{S^{(i)}(\mathbf{skewer}_j)}[\mathbf{g}(x,y)]$ for each pixel vector $\mathbf{g}(x, y)$ at $\mathbf{G}_i$. Pixels with $N_{\mathrm{PPI}}^{(i)}[\mathbf{g}(x,y)] > T$ are selected and sent to the master.
4. The master collects all the partial results and merges them to form a set of final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.

The total workload involved in the PPI algorithm can be approximated by $M \times J \times E$, where M is the number of pixel vectors in the input scene, J is the number of skewers, and E is the number of spectral bands after PCT-based dimensionality reduction.

3.2.2. Parallel implementation of N-FINDR algorithm (P-FINDR)

The P-FINDR algorithm consists of three main steps: random endmember selection, simplex volume calculation and volume replacement. It should be noted that, due to the lack of detailed implementations of the N-FINDR algorithm in the literature, our parallel version incorporates our own understanding of the algorithm in accordance with high-level algorithmic descriptions provided in [15,28].

Algorithm 4. Parallel N-FINDR-based endmember selection (P-FINDR)

Input: E-D cube $\mathbf{G}$.
Output: Set of E final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.

1. The master selects a random set of E initial pixel vectors $\{\mathbf{e}_e^{(0)}\}_{e=1}^{E}$, and then finds $V(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_E^{(0)})$, i.e., the volume of the simplex defined by $\{\mathbf{e}_e^{(0)}\}_{e=1}^{E}$, denoted by $S(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_E^{(0)})$, as follows:
$$V(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_E^{(0)}) = \frac{\left|\det \begin{bmatrix} \mathbf{1} & 1 & \ldots & \mathbf{1} \\ \mathbf{e}_1^{(0)} & \mathbf{e}_2^{(0)} & \ldots & \mathbf{e}_E^{(0)} \end{bmatrix}\right|}{(E-1)!}$$
2. The workers calculate the volume of E simplexes, $V(\boldsymbol{g}(x,y), \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_E^{(0)}), \ldots, V(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \boldsymbol{g}(x,y))$ in parallel, each of which is formed by replacing one endmember $\mathbf{e}_e^{(0)}$ with the sample vector $\mathbf{g}(x, y)$. Each worker performs replacements using pixels in its local partition, obtained using the GSP algorithm.
3. If none of these E recalculated volumes is greater than $V(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \cdots, \mathbf{e}_E^{(0)})$, then no endmember in $\{\mathbf{e}_e^{(0)}\}_{e=1}^{E}$ is replaced. Otherwise, the master replaces the endmember which is absent in the largest volume among the E simplexes with the vector $\mathbf{g}(x,y)$. Let such endmember be denoted by $\mathbf{e}_l^{(0)}$. A new set of endmembers is produced sequentially at the master by letting $\mathbf{e}_l^{(1)} = \mathbf{g}(x,y)$ and $\mathbf{e}_e^{(1)} = \mathbf{e}_e^{(0)}$ for $e \neq l$.
4. Repeat from step 2 until no replacements occur.

As described in [29], the total workload involved by the N-FINDR algorithm can be approximated by taking into account that the algorithm has to compute the determinant of a $E \times E$ matrix $E \times N$ times. The naive method of implementing an algorithm to compute the determinant is to use Laplace's formula for development by minors [28]. However, this approach is inefficient as it is of order $E!$ for a $E \times E$ matrix. In this work, an improvement to E^3 is achieved by using the LU decomposition for the computation of determinants [30].

3.2.3. Parallel implementation of IEA algorithm (P-IEA)

The IEA algorithm produces a set of endmembers *sequentially* as opposed to PPI and N-FINDR, which generate the final endmember set in single-shot mode. As shown below, parallelization of IEA requires several master–worker communications during the execution.

Algorithm 5. Parallel iterative error analysis (P-IEA)

Input: N-D data cube $\mathbf{F}$, Number of endmembers E.
Output: Set of E final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.

1. Divide the original data cube $\mathbf{F}$ into $|P|$ heterogeneous partitions using the GSP algorithm, where $|P|$ is the number of workers.
2. Calculate the N-D mean vector $\bar{\mathbf{f}}$ concurrently, where each component is the average of the pixel values of each spectral band of the input data. This vector is formed at the master once all the processors have finished their respective parts. The workload involved by this step is $M \times N$, where M is the total number of pixels in the hyperspectral image and N is the number of spectral bands.
3. Broadcast $\bar{\mathbf{f}}$ to all workers so that each worker can form an error image by assuming that all pixels in the local partition are made up of $\bar{\mathbf{f}}$ with 100% abundance.
4. Each worker finds the pixel $\mathbf{f}(x, y)$ in the local partition (obtained using GSP algorithm) with the largest abundance estimation error, i.e., the pixel that has largest error (in least squares sense [12]) if it is represented in terms of a linear mixture of the pixels found by previous iterations. As shown in [1], this is done by multiplying each pixel $\mathbf{f}(x, y)$ by $(\mathbf{M}^{\mathrm{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathrm{T}}$, where $\mathbf{M} = \{\mathbf{e}_e\}_{e=1}^{E}$ and the superscript "T" denotes the matrix transpose operation. The pixel with largest abundance estimation error, denoted by $\mathbf{p}_i$, and its associated error score, denoted by s_i, are sent to the master.
5. The master selects a first endmember $\mathbf{e}_1$ as the pixel $\mathbf{p}_i$ with the maximum associated error score s_k, for $i = 1, \cdots, |P|$ and broadcasts $\mathbf{E} = [\mathbf{e}_1]$ to all the workers.

6. Repeat from step 4 using $\mathbf{E}$ instead of $\bar{\mathbf{f}}$, and repeatedly incorporate a new endmember $\mathbf{e}_2$, $\mathbf{e}_3$, ..., $\mathbf{e}_E$ to $\mathbf{E}$ until $\mathbf{E} = \{\mathbf{e}_e\}_{e=1}^{E}$, in which case the algorithm is terminated.

It should be noted that the IEA is a computationally demanding algorithm, in particular, when the number of endmembers to be detected is very large [16]. In particular, Steps 4 to 6 involve a vector-matrix multiplication with workload of $N^2 \times E$, and three matrix multiplications, each with workload E^3. As the value of E grows, the complexity of the algorithm increases in exponential fashion.

We must also note that there are a number of salient differences between the P-IEA (algorithm 5 above) and the P-PPI (which consists of parallel algorithms 2 and 3) and the P-FINDR (which consists of algorithms 2 and 4). First, the P-IEA does not require a dimensionality reduction step. Further, both the P-PPI and the P-FINDR must research all the endmembers when parameter E is changed, which is a usual practice in hyperspectral studies. In other words, a set of E-1 endmembers is not necessarily a subset of E endmembers generated by the same algorithm. Quite opposite, the P-IEA produces endmembers *in order*, so a set of E generated endmembers always includes the set of previously generated E-1 endmembers.

3.2.4. Parallel implementation of AMEE algorithm (P-AMEE)

To conclude this section, we outline a parallel morphological technique (P-AMEE) that also works with the entire input data cube with no need for previous dimensionality reduction [22]. First, a cumulative distance between each pixel vector $\mathbf{f}(x, y)$ and all pixel vectors in the spatial neighborhood given by a spatial kernel or *structuring element* B is defined as

$$D_B[\mathbf{f}(x,y)] = \sum_s \sum_t \mathrm{SAM}[\mathbf{f}(x,y), \mathbf{f}(s,t)], \text{where} (s,t) \in Z^2(B),$$

where SAM is the spectral angle mapper, given by

$$\mathrm{SAM}[\mathbf{f}(x,y), \mathbf{f}(s,t)] = \cos^{-1}[\mathbf{f}(x,y) \cdot \mathbf{f}(s,t) / \|\mathbf{f}(x,y)\| \cdot \|\mathbf{f}(s,t)\|].$$

Based on the above metric, extended morphological erosion and dilation can be used, respectively, to extract the most highly pure and the most highly mixed pixel in the B-neighborhood as follows [17]:

$$(\mathbf{f} \Theta B)(x,y) = \arg \min_{(s,t) \in Z^2(B)} \{D_B[\mathbf{f}(x+s, y+t)]\} \text{and} (\mathbf{f} \oplus B)(x,y) = \arg \max_{(s,t) \in Z^2(B)} \{D_B[\mathbf{f}(x+s, y+t)]\}.$$

The extended morphological operations defined above (called morphological erosion and dilation, respectively, and illustrated in Fig. 5) are used to design a heterogeneous algorithm as follows [31]:

Algorithm 6. Parallel morphological endmember extraction (P-AMEE)

Input: N-D cube $\mathbf{F}$, Number of iterations $I_{\max}$, 3 × 3-pixel structuring element B, Number endmembers E.
Output: Set of final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.

1. Divide the original data cube $\mathbf{F}$ into $|P|$ heterogeneous partitions, denoted by $\{\mathbf{PSSP}_i\}_{i=1}^{|P|}$ (called parallelizable spatial/spectral partitions) using the GSP algorithm in combination with a data-replication function to avoid accesses outside the local domain of each partition [22]. Then, scatter the partial data structures to each of the $|P|$ workers.
2. Using parameters $I_{\max}$, B and E, each worker executes the sequential AMEE algorithm locally at each processor p_i for the corresponding $\mathbf{PSSP}_i$. The sequential algorithm has been described before [13,17], and consists of the following steps:
   2.1. Set $j = 1$ and initialize a morphological eccentricity index score $\mathrm{MEI}(x, y) = 0$ for each local pixel.
   2.2. Move B(a fixed 3 × 3-pixel structuring element) through all the pixels of the local $\mathbf{PSSP}_i$ and calculate the maximum and the minimum pixels using erosion and dilation operations, respectively. Update the MEI score at each pixel by calculating the SAM between the maximum and the minimum.
   2.3. Set $j = j + 1$. If $j = I_{\max}$ then go to step 2.4. Otherwise, replace $\mathbf{PSSP}_i$ by its dilation using B and go to 2.2.
   2.4. Select the set of E pixel vectors in the local partition with the highest MEI scores.

3. The master gathers all the individual endmember sets provided by the workers and forms a unique set $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the SAM for all possible pairs.

The workload involved by the first two steps of the AMEE algorithm can be approximated by $M \times B \times I_{\max} \times N$, where M is the number of pixel vectors in the input image, B is the size in pixels of the structuring element (set to a constant $B = 3 \times 3$ in this work), $I_{\max}$ is the number of iterations, and N is the number of spectral bands. In order to form a unique set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$, an additional number of $E^2 \times N$ operations is required.

We emphasize that Step 1 of P-AMEE algorithm uses a function that replicates border data such that the intersection between two adjacent partitions is non-empty. This allows processing of local border pixels via the considered 3×3-pixel structuring element without additional communications (otherwise, pixel vectors in a different partition should be communicated to complete the calculations for border pixel vectors at the local partition). Such replication-based strategy has been demonstrated in previous work to be more efficient than allowing data communication for such pixels [22], mainly due to the large volume of information to be communicated in hyperspectral imagery as a result of the high-dimensional nature of pixel vectors. A similar strategy is provided in the user transparent software architecture in [32].

It is also worth noting that our implementation of P-AMEE makes use of a constant 3×3-pixel structuring element through iterations. As a result, instead of increasing the size of the structuring element, we replace the original cube $\mathbf{F}$ (or, equivalently, a local partition in parallel processing) by the resulting cube after a dilation operation using B. This allows us to perform multi-scale analysis without increasing the overlap border size between iterations, thus minimizing the amount of redundant computations.

3.3. Parallel spectral unmixing algorithm

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 [1], i.e., the linear spectral unmixing (LSU) technique that can be briefly described as follows. Suppose that there are E endmembers $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_E$ in a hyperspectral image scene, and let $\mathbf{m}$ be a mixed pixel vector. LSU assumes that the spectral signature of m can be represented by a linear mixture of $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_E$ with appropriate abundance fractions specified by $a_1, a_2, \ldots, a_E$. Then, we can model the spectral signature of an image pixel $\mathbf{m}$ by a linear regression form $m = \mathbf{e}_1 \cdot a_1 + \mathbf{e}_2 \cdot a_2 + \cdots + \mathbf{e}_E \cdot a_E$. Two constraints should be imposed on this model to produce adequate solutions. These are the abundance sum-to-one constraint, that is, $\sum_{e=1}^{E} a_e = 1$, and the abundance non-negativity constraint, that is, $a_e \geqslant 0$ for $1 \leqslant e \leqslant E$, where the two constraints above can be imposed as described in [12]. 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.

Algorithm 7. Parallel linear spectral unmixing (P-LSU)

Input: N-D data cube $\mathbf{F}$, Set of final endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$.
Output: Set of fractional abundances $\{a_e(x,y)\}_{e=1}^{E}$ for each pixel $\mathbf{f}(x, y)$.

1. Divide the original data cube $\mathbf{F}$ into $|P|$ heterogeneous partitions using the GSP algorithm, where $|P|$ is the number of workers.
2. Broadcast the set $\{\mathbf{e}_e\}_{e=1}^{E}$ to all the workers.
3. For each pixel $\mathbf{f}(x, y)$ in a local partition, obtain a set of abundance fractions specified by $a_1(x,y), a_2(x,y), \ldots, a_E(x, y)$ using $\{\mathbf{e}_e\}_{e=1}^{E}$, so that $\mathbf{f}(x,y) = \mathbf{e}_1 \cdot a_1(x,y) + \mathbf{e}_2 \cdot a_2(x,y) + \cdots + \mathbf{e}_E \cdot a_E(x, y)$, and taking into account the abundance sum-to-one and abundance non-negativity constraints [12]. This is done by multiplying each pixel $\mathbf{f}(x, y)$ by $(\mathbf{M}^{\mathrm{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathrm{T}}$ [1], where $\mathbf{M} = \{\mathbf{e}_e\}_{e=1}^{E}$. As mentioned above, this step involves a vector-matrix multiplication with workload of $N^2 \times E$, and three matrix multiplications, each with E^3.
4. The master collects all the individual sets of fractional abundances $\{a_e^{(i)}(x,y)\}_{e=1}^{E}$ calculated for the pixels at every individual partition i, with $i = 1, \cdots, P$, and forms a final set of fractional abundances designated by $\{a_e(x,y)\}_{e=1}^{E} = \cup_{i=1}^{P}\{a_e^{(i)}(x,y)\}_{e=1}^{E}$.

As a final note, we reiterate that the proposed hyperspectral data processing chain consists of a sequence of three steps, i.e., dimensionality reduction (optional), endmember extraction, and spectral unmixing (see Fig. 2), each of which has been implemented in parallel in this work. Although implementations for parallel dimensionality reduction [26,27] and endmember extraction algorithms [22,33] are available in the literature (in homogeneous form), the heterogeneous implementations described in this section represent highly innovative contributions. Performance data for all considered parallel algorithms are given in the following section.

4. Experimental results

This section provides an assessment of the effectiveness of the parallel algorithms described in Section 3. Before describing our study on performance analysis, we first describe the parallel computing architectures used for evaluation purposes.

4.1. Parallel computing architectures

Two different parallel computing platforms have been used in this work for experimental assessment. Table 1 shows the specifications of processors in a fully heterogeneous cluster of computers available at Heterogeneous Computing Laboratory, University College Dublin.[1] It is made up of 16 nodes from Dell, IBM, and HP, with Celeron, Pentium 4, Xeon, and AMD processors ranging in speed from 1.8 to 3.6 Ghz. Accordingly, architectures and parameters such as cache and main memory all vary. Two machines have SCSI hard drives while the rest have SATA. Operating Systems used are Fedora Core 4 (11 nodes) and Debian (5). The network hardware consists of two Cisco 24 + 4 port Gigabit switches. Each node has two Gigabit ethernet ports.

For illustrative purposes, Table 1 also reports the relative speed of each processor in the heterogeneous cluster, estimated using a benchmark function representative of hyperspectral imaging algorithms. At this point, we reiterate that the nature of the benchmark function used as a baseline for estimating the relative speed of processors is crucial for the success of the general optimization problem described in Section 3. On the one hand, this function should be truly representative of the underlying application. On the other hand, the computations involved in such function should be small enough to give an accurate approximation of the processing power in a short time.

In this work, we have adopted as representative benchmark computation for all described hyperspectral image processing algorithms the dot product between two spectrally distinct N-dimensional vectors (spectral signatures). A dot product calculation between two different pixel vectors, say, $\mathbf{f}(x, y)$ and $\mathbf{f}(s, t)$, is given by $\sum_{k=1}^{N} \mathbf{f}(x,y)^{(k)} \times \mathbf{f}(s,t)^{(k)}$, where the superscript "(k)" denotes the k-th spectral component of the pixel vector. There are several reasons that justify our selection of the dot product as representative benchmark function in the context of our application:

1. First and foremost, the projection of an N-dimensional vector to another one is a baseline computation included in all the hyperspectral image processing algorithms described in Section 3. Therefore, the use of this regular core computation is truly representative of every algorithm, as indicated by the fact that the workload performed by each algorithm can be expressed in terms of elementary MAC operations.
2. Secondly, the proposed benchmark function allows for simple modelling of memory parameters relative to processing nodes in the considered heterogeneous architecture. In order to model the impact of local memory on the proposed benchmark function, we simply assume that the amount of data allocated to a single processor in the heterogeneous cluster is a full AVIRIS hyperspectral cube with 614×512 pixels and 224 spectral bands (each stored using 2 bytes). This is the standard image size produced by AVIRIS in each pass. Since AVIRIS is the most advanced hyperspectral instrument currently available, it provides a highly relevant case study for the definition of the benchmark function. In particular, our benchmark function assumes a realistic scenario in which each processor may be forced to make use of reallocation/paging

[1] See http://hcl.ucd.ie/Hardware for additional details.

Table 1
Specifications of processors in a heterogeneous cluster at University College Dublin, and relative speed of each processor, estimated using a benchmark function tuned for hyperspectral imaging applications

#	Model	Processor	O/S	CPU (Ghz)	Mem. (MB)	Cache (KB)	HDD1	HDD2	Relative speed
0, 1	Dell Poweredge SC1425	Intel Xeon	Fedora Core 4	3.6	256	2048	240 GB SCSI	80 GB SCSI	7.91
2, 3, 4, 5, 6, 7	Dell Poweredge 750	Intel Xeon	Fedora Core 4	3.4	1024	1024	80 GB SATA	N/A	7.22
8	IBM E-server 326	AMD Opteron	Debian	1.8	1024	1024	80 GB SATA	N/A	2.76
9	IBM E-server 326	AMD Opteron	Fedora Core 4	1.8	1024	1024	80 GB SATA	N/A	2.61
10	IBM X-Series 306	Intel Pentium 4	Debian	3.2	512	1024	80 GB SATA	N/A	6.15
11	HP Proliant DL 320 G3	Intel Pentium 4	Fedora Core 4	3.4	512	1024	80 GB SATA	N/A	6.84
12	HP Proliant DL 320 G3	Intel Celeron	Fedora Core 4	2.9	1024	256	80 GB SATA	N/A	3.55
13	HP Proliant DL 140 G2	Intel Xeon	Debian	3.4	1024	1024	80 GB SATA	N/A	7.61
14	HP Proliant DL 140 G2	Intel Xeon	Debian	2.8	1024	1024	80 GB SATA	N/A	3.39
15	HP Proliant DL 140 G2	Intel Xeon	Debian	3.6	1024	2048	80 GB SATA	N/A	8.72

mechanisms due to cache misses. This approach allows us to realistically model the relative speed of each heterogeneous processor by simply running a standardized core computation for hyperspectral image processing algorithms, assuming that the processor running the algorithm stores a full hyperspectral data cube in its local memory.

3. Finally, we must note that the considered core computation is very small compared to the other computations performed by any hyperspectral algorithm (execution of the benchmark function used to produce the relative speeds reported in Table 1 took less than 0.015 seconds in all cases).

In order to analyze issues of algorithm scalability on larger-scale parallel platform, we have also experimented with Thunderhead, a 568-processor Beowulf cluster located at NASA's Goddard Space Flight Center in Maryland. Thunderhead can be seen as an evolution of the HIVE (Highly Parallel Virtual Environment) project [6], started in 1997 to build a homogeneous commodity cluster to be used in a wide range of scientific applications. The Thunderhead consists of 268 Xeons which results in a total peak performance of 2.5728 Tflops (http://thunderhead.gsfc.nasa.gov). Each of the nodes has 1 GB of main memory and 80 GB of local disk space. Despite the computational power offered by Thunderhead, a current design trend at Goddard and other NASA centers is to exploit heterogeneous, massively parallel computing platforms able to operate in large-scale distributed environments. To explore the scalability of the proposed algorithms in available massively parallel platforms, performance data on Thunderhead will also be given in this section.

4.2. Performance analysis

The parallel algorithms were applied to a hyperspectral scene collected by an AVIRIS flight over the Cuprite mining district in Nevada, which consists of 614×512 pixels and 224 bands (137 MB in size). The site is well understood mineralogically, and has several exposed minerals of interest. Fig. 7a shows a spectral band of the image, and Fig. 7b plots the spectra of five minerals measured in the field by US Geological Survey (USGS). These signatures will be used to substantiate endmember extraction accuracy. Fractional abundance maps derived by the USGS Tetracorder method[2] will also be used to evaluate abundance estimation accuracy. Since the AVIRIS data are available online,[3] people interested in the proposed parallel algorithms can reproduce our results and conduct their own experiments.

An experiment-based cross-examination of endmember extraction and abundance estimation accuracy is first presented in Table 2, which tabulates the SAM scores obtained after comparing five USGS library spectra with the corresponding endmembers extracted by the parallel algorithms. The smaller the SAM values across

[2] http://speclab.cr.usgs.gov/spectral-lib.html.
[3] http://aviris.jpl.nasa.gov/html/aviris.freedata.html.

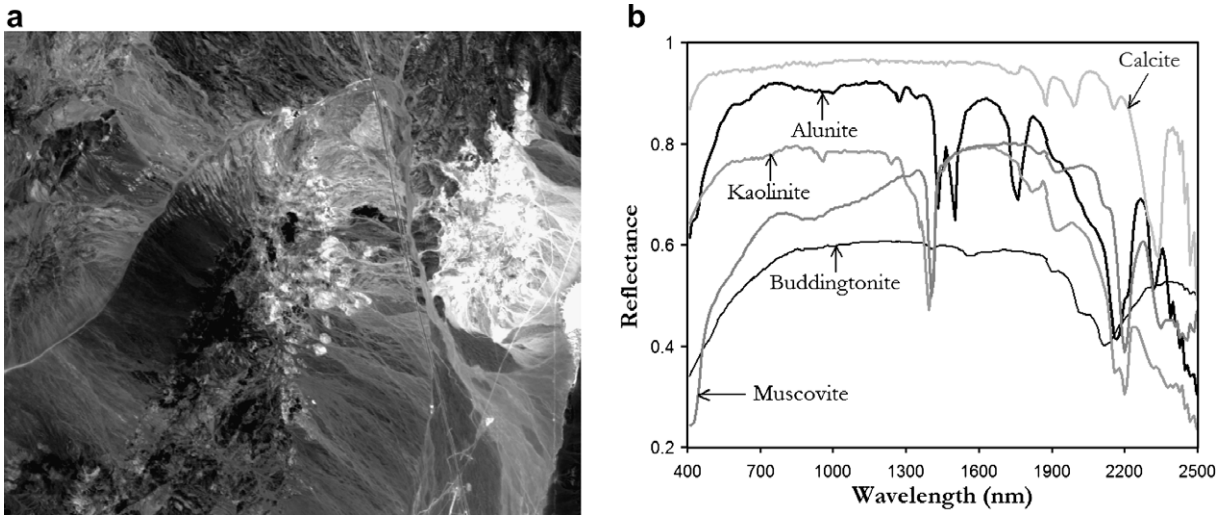


Fig. 7. (a) Spectral band at 587 nm wavelength of the AVIRIS scene comprising mineral features at the Cuprite mining district, Nevada. (b) USGS ground-truth mineral spectra.

Table 2
SAM-based similarity scores and RMSE-based abundance estimation errors (in bold typeface) between USGS signatures and algorithm-derived endmembers

Algorithm	Alunite	Buddingtonite	Calcite	Kaolinite	Muscovite
P-PPI	0.084	0.106	0.105	0.136	0.108
	0.164	**0.193**	**0.186**	**0.217**	**0.195**
P-FINDR	0.094	0.052	0.065	0.105	0.098
	0.194	**0.163**	**0.178**	**0.205**	**0.186**
P-IEA	0.091	0.103	0.093	0.078	0.081
	0.123	**0.134**	**0.119**	**0.116**	**0.131**
P-AMEE	0.073	0.071	0.084	0.064	0.077
	0.114	**0.105**	**0.116**	**0.103**	**0.112**

the five minerals considered in Table 2, the better the results. It should be noted that Table 1 only displays the smallest SAM scores of all endmembers with respect to each USGS signature for each algorithm. In previous work, we have experimentally tested that spectral similarity scores below 0.1 can be considered as widely acceptable [13]. As a result, Table 2 reveals that most endmember extraction algorithms tested can provide signatures which are very similar, in spectral terms to ground-truth references. Table 2 also reports (in bold typeface) the root mean square error (RMSE) between the abundances estimated by using the P-LSU algorithm in combination with the endmembers provided by the different methods, using the following expression

$$\mathrm{RMSE}(\mathbf{e}_e) = \left((1/M) \sum_{x=1}^{X} \sum_{y=1}^{Y} [a_e(x,y) - \hat{a}_e(x,y)]^2 \right)^{1/2},$$

where $\hat{a}_e(x,y)$ denotes the abundance estimated by P-LSU for endmember $\mathbf{e}_e$ at the pixel $\mathbf{f}(x, y)$, $a_e(x, y)$ denotes the abundance measured by USGS for that endmember at the same pixel, and $M = X \times Y$ is the total number of pixels in the input hyperspectral image. Overall, abundance estimation results in Table 2 reveal errors of about 10% for most considered algorithms. This figure is considered to be widely acceptable in the context of remote sensing applications [13].

We must also note that the number of endmembers to be extracted, E, was set to 16 for all parallel methods tested after using the virtual dimensionality (VD) concept, which is used to automatically calculate the number of components that should be retained after applying a PCT transform for explaining most of the variance of

the data using a limited number of components [1]. However, only five endmembers were used for evaluation due to limited ground-truth data availability. Prior to a full examination and discussion of results, it is important to outline parameter values used for the P-PPI and P-AMEE methods, bearing in mind that the other tested parallel methods only require as input the number of endmembers to be found, E. For the P-PPI, parameter T was set to the mean of N_{PPI} scores after $J = 10^3$ skewer iterations. Previous studies on our P-AMEE algorithm revealed that high quality endmembers were found by setting $I_{max} = 7$ iterations [22]. We emphasize that the parameter settings above are in agreement with those used in previous studies [13]. The output provided by the parallel algorithms was verified using not only our sequential implementations, but also the original versions of the algorithms (using the same parameter settings) as well. In all cases, we experimentally tested that our versions provided exactly the same results as those found by the original algorithms.

4.2.1. Performance analysis on the heterogeneous cluster

To investigate the parallel properties of the tested algorithms using the AVIRIS scene in Fig. 7a, their performance was first tested by timing the programs on the heterogeneous cluster of workstations described in Table 1. For that purpose, Table 3 reports the execution times (in seconds) measured for the parallel algorithms in each of the processors of the heterogeneous cluster. For comparative purposes, Table 4 reports the processing times measured for real sequential versions of the tested algorithms, executed in only one processor of the same heterogeneous cluster. We selected processor #10 in Table 1 for testing the sequential implementations because its relative speed (6.15) was the closest one to the average of all relative speeds reported in Table 2.

As shown by Table 3, the heterogeneous algorithms seemed to be able to adapt efficiently to the heterogeneous computing environment where they were run. In particular, after comparing the times reported on Table 3 to those reported on Table 4, one can see that the heterogeneous algorithms, executed on the heterogeneous cluster, were always several times faster than the equivalent sequential algorithms executed on processor #10 of the heterogeneous cluster. It is also worth noting that the fastest algorithm on the heterogeneous

Table 3
Execution times (in seconds) of the parallel algorithms in each of the processors of the heterogeneous cluster (processor #10 is used as the master)

Processor #	P-PCT	P-PPI	P-FINDR	P-IEA	P-AMEE	P-LSU
0	136.67	304.21	47.02	278.45	145.63	152.23
1	136.91	304.69	47.19	278.60	145.64	152.49
2	136.75	304.05	47.06	278.52	145.65	152.40
3	136.73	304.02	47.02	278.47	145.64	152.35
4	136.42	303.78	46.94	278.12	145.56	152.16
5	136.48	303.89	46.97	278.22	145.58	152.19
6	136.12	303.79	46.81	278.03	144.39	151.94
7	136.29	303.99	46.64	278.09	145.56	152.03
8	136.39	304.04	46.90	278.12	145.60	151.82
9	136.92	304.78	47.27	278.34	145.72	152.58
10	150.27	332.04	52.19	302.21	147.63	155.37
11	136.55	304.53	47.00	278.39	145.61	152.20
12	136.52	304.48	46.97	278.33	145.59	152.19
13	136.53	304.51	46.99	278.35	145.59	152.21
14	136.50	304.50	46.90	278.30	145.58	152.18
15	136.45	304.47	46.92	278.28	145.58	152.15

Table 4
Execution times (in seconds) of real sequential versions of the considered algorithms measured in processor #10 of the heterogeneous cluster

P-PCT	P-PPI	P-FINDR	P-IEA	P-AMEE	P-LSU
1540	3669	539	3393	2149	2296

cluster was the P-FINDR, which was the only one able to provide a response below 1 min, while both P-PPI and P-IEA resulted in the highest processing times (above 5 min).

In order to further compare the performance gain of heterogeneous algorithms as compared to their respective sequential versions in more detail, we have conducted a thorough study of scalability on the heterogeneous cluster [34]. For that purpose, Fig. 8 shows the performance gain of heterogeneous algorithms with regards to their respective sequential versions as the number of processors was increased on the heterogeneous cluster. Here, we assumed that processor #10 was always the master and varied the number of slaves. The construction of speedup plots in heterogeneous environments is not straightforward [35], mainly because the workers do not have the same relative speed, and therefore the order in which they are added to plot the speedup curve needs to be further analyzed. In order to evaluate the impact of the order of selection of slaves, we have tested three different ordering strategies:

1. First, we used an ordering strategy based on increasing the number of processors according to their processor numbers in Table 2, i.e., the first case study tested (labeled as "2 CPUs" in Fig. 8) consisted of using processor #10 as the master and processor #0 as the slave; the second case tested (labeled as "3 CPUs" in Fig. 8) consisted of using processor #10 as the master and processors #0 and #1 as slaves, and so on, until a final case (labeled as "15 CPUs" in Fig. 8) was tested, based on using processor #10 as the master and all remaining 15 processors as slaves.
2. Second, we used an ordering strategy based on the relative speed of processors in Table 2, i.e., the first case study tested consisted of using processor #10 as the master and processor #9 (i.e., the one with lowest relative speed) as the slave; the second case tested consisted of using processor #10 as the master and processors #9 and #8 (i.e., the two processors with lowest relative speed) as slaves, and so on, until a final case was tested, based on using processor #10 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 consisted of using processor #10 as the master and a different processor, selected randomly among the remaining processors (say, processor p_i) as the slave; the second case consisted of using processor #10 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, based on using processor #10 as the master and all remaining 15 processors as slaves.

Since the three tested scenarios resulted in almost identical speedups, Fig. 8 only reports the results obtained for the first ordering strategy tested. As shown by Fig. 8, the workers provided a linear performance increase (regardless of their relative speed) when incorporated as additional processing nodes in every considered algorithm. The above results indicate that the proposed heterogeneous algorithms were able to distribute

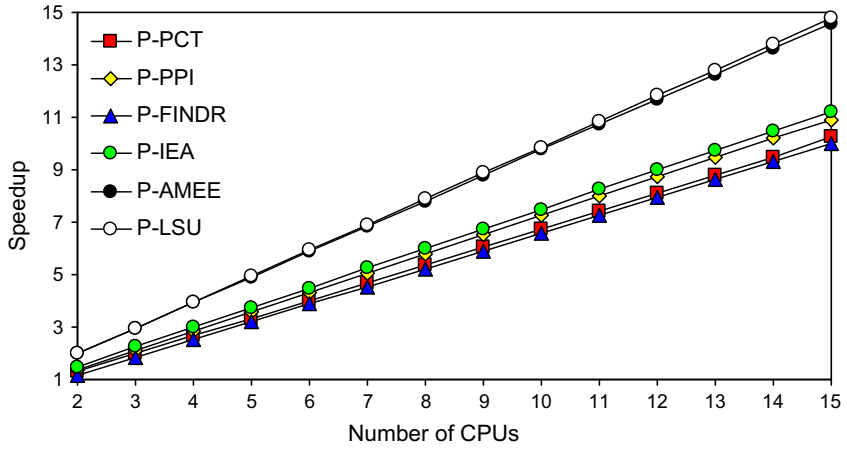


Fig. 8. Speedup achieved by the proposed parallel algorithms on the heterogeneous cluster (processor #10 is used as the master).

the workload effectively between the slaves, regardless of the order in which new heterogeneous processors were added.

For the sake of completeness, we also conducted additional experiments based on selecting other processors (instead of processor #10) to assume the role of master. In all cases, the results obtained were very similar to those reported in Table 3 (i.e., a very slight increase was observed in the processing times measured for the master, while the processing times measured for the slaves were almost identical). One can also see that, although P-FINDR was the algorithm that resulted in the lowest execution times in Table 3, it was also one of the algorithms with lowest speedup factors in Fig. 8. Quite opposite, both P-LSU and P-AMEE performed satisfactorily in terms of scalability and closely approached linear speedup.

In order to explore the parallel properties of the considered algorithms in more detail, an in-depth analysis of computation and communication times achieved by the different methods is also highly desirable. For that purpose, Table 5 shows the total time spent by the tested algorithms in communications and computations in the test case reported on Table 3, i.e., using processor #10 as the master and all 15 remaining processors as slaves. Here, two types of computation times were analyzed, namely, sequential (those performed by the master node *with no other parallel tasks active in the system*) and parallel (the rest of computations, i.e., those performed by the master node and/or the workers *in parallel*). The latter includes the times in which the workers remain idle.

It can be seen from Table 5 that sequential computations were particularly significant (when compared to parallel computations) for the P-PCT algorithm. They were also relevant for the P-PPI and P-FINDR. 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., covariance matrix calculations in the P-PCT or 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, although the P-AMEE is the only technique that introduces redundant information (expected to slow down computations *a priori*), Table 5 reveals that the amount of sequential computations introduced by this algorithm is very low when compared to that introduced by the other endmember extraction algorithms. As a result, the ratio of computations to communications for this method is much higher. This comes at no surprise, since P-AMEE is a windowing type algorithm as opposed to the other methods and, therefore, it is expected to scale better. This property allowed P-AMEE to process the full AVIRIS scene (137 MB in size) in only 156 s on the heterogeneous cluster, while other methods such as P-IEA and P-PPI required twice as much computing time to complete their calculations in the same environment. Finally, it can also be seen from Table 5 that the cost of parallel computations clearly dominated that of communications in all the considered algorithms.

For completeness, Table 6 shows the load imbalance scores [36] achieved by the considered algorithms on the heterogeneous cluster. The imbalance is defined as $D = R_{max}/R_{min}$, where R_{max} and R_{min} are the maxima

Table 5
Computation/communication times (in seconds) achieved by the parallel algorithms on the heterogeneous cluster

Algorithm	Communications	Sequential computations	Parallel computations
P-PCT	11.05	16.11	123.11
P-PPI	7.85	21.12	303.07
P-FINDR	7.08	6.54	38.57
P-IEA	10.21	9.96	282.04
P-AMEE	6.73	5.69	135.21
P-LSU	5.21	6.07	144.09

Table 6
Maxima and minima execution times and load balancing rates for the parallel algorithms executed on the heterogeneous cluster

Algorithm	P-PCT	P-PPI	P-FINDR	P-IEA	P-AMEE	P-LSU
R_{max}	150.27	332.04	52.19	302.21	147.63	155.37
R_{min}	136.12	303.79	46.64	278.03	144.39	151.82
D_{All}	1.104	1.093	1.119	1.087	1.022	1.023
D_{Minus}	1.005	1.003	1.013	1.002	1.009	1.005

and minima processor run times, respectively. Therefore, perfect balance is achieved when $D = 1$. In the table, we display the imbalance considering all processors, D_{All}, and also considering all processors but the root, D_{Minus}. As we can see from Table 6, the P-LSU and P-AMEE were able to provide values of D_{All} very close to 1 in the considered platform. It is also clear from Table 6 that the two algorithms above provided almost the same results for both D_{All} and D_{Minus} while, for the other tested methods, load balance was slightly better when the root processor was not included. Despite the fact that conventional hyperspectral imaging algorithms generally do not take into account the spatial information explicitly into the computations (which has traditionally been perceived as an advantage for the development of parallel implementations), and taking into account that P-AMEE introduces redundant information expected to slow down the computation *a priori*, experimental results in Table 6 seem to indicate that the P-AMEE is more effective in terms of workload distribution than other parallel endmember extraction methods which appeared to be more "pleasingly parallel" *a priori*. In particular, the combination of P-AMEE followed by P-LSU provided a well-balanced analysis result in about 300 s, while slightly higher imbalance scores were obtained for the P-PPI and P-FINDR (which are combined with P-PCT prior to data processing).

Before concluding this subsection, we would like to emphasize the importance of incorporating memory considerations in the benchmark function used to estimate processor relative speeds in the heterogeneous cluster. For perspective, Fig. 9 shows the values of $R_{\max}$, $R_{\min}$ and D obtained for the considered parallel algorithms using a benchmark function which only modeled the processing power of heterogeneous processors (i.e., using the vector dot product as the core computation) but without assuming that each processor stores a full hyperspectral data cube. In this case, reallocation and paging mechanisms due to cache misses are not included in the computation of the benchmark function. As shown by Fig. 9, disregarding virtual memory paging and cache effects in the definition of the benchmark function often leads to higher imbalance scores.

4.2.2. Performance analysis on the Beowulf cluster

To analyze scalability issues in a larger computing platform, this subsection provides parallel performance results using the Thunderhead Beowulf cluster as the baseline computing architecture [2]. It should be noted that results in this subsection should not be regarded as indicative of the efficiency of the proposed heterogeneous parallel implementations since, in this parallel platform, our GSP algorithm assigns an equal amount of workload to each processor. As a result, the experimental evaluation conduced in this subsection is only intended to provide a preliminary assessment of the scalability of the proposed algorithms in a parallel platform with a much higher number of processors. For that purpose, Fig. 10 plots the speedups achieved by a multi-processor run of each of the proposed parallel algorithms over the execution of a real sequential version of each algorithm on a single Thunderhead node (a maximum of 400 processors were available to us at the time of experiments).

Results in Fig. 10 reveal that the performance drop from linear speedup in the considered parallel algorithms was more relevant as the number of processors was increased. In particular, the P-AMEE scaled significantly better than the other endmember extraction methods tested, specifically, when the number of

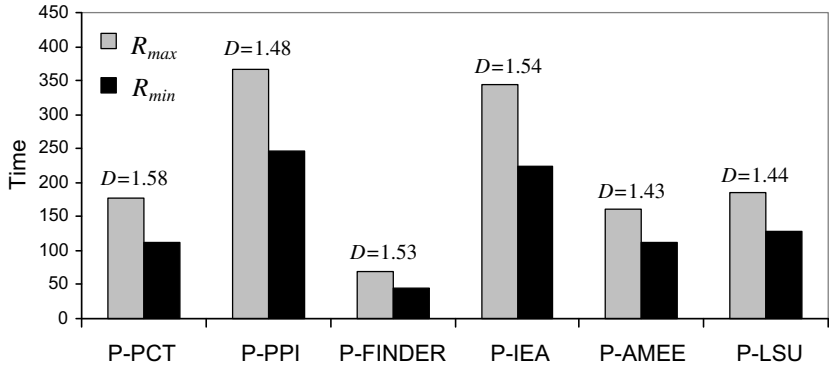


Fig. 9. Maxima/minima execution times (seconds) and load balancing rates for a case study in which memory considerations are not included in the definition of the benchmark function used to estimate processor relative speeds in the heterogeneous cluster.

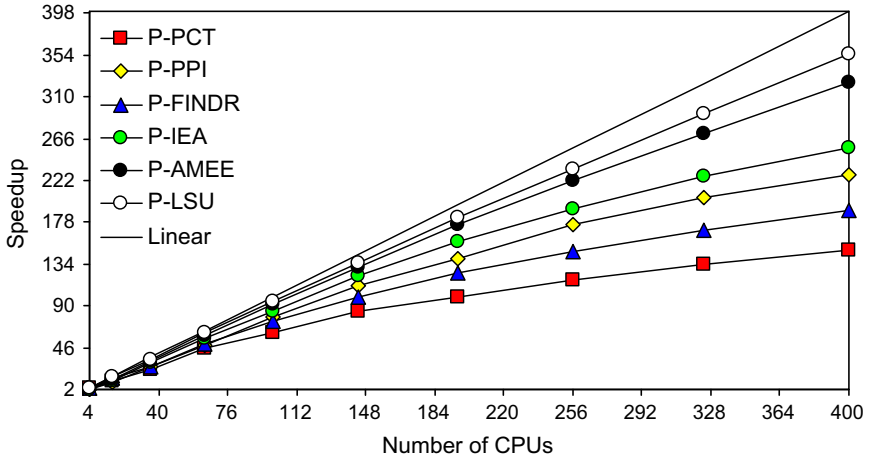


Fig. 10. Scalability of the proposed parallel algorithms on NASA's Thunderhead Beowulf cluster.

processors was very large. It is also clear that P-LSU is the algorithm that scaled better on Thunderhead, a fact that is not surprising given the straightforward parallelization of the LSU technique. On the other hand, the performance decrease experienced by all parallel algorithms as the number of processors was increased can be explained by the nature of our MPI implementation, in which a master processor sends the partitions of the original image, using the `MPI_Scatter` operation, to a group of processors, each of which processes one partition. The partitions are processed in parallel and then returned to the master processor with the `MPI_Gather` operation. This framework requires an optimization of collective communications, which in our application mostly take place by aggregating global values on the server and then communicating them to the slaves.

Although fine-tuning of the proposed MPI-based parallel algorithms for efficient performance in large-scale commodity clusters is out of the scope of this paper, this problem can be approached by resorting to a recently proposed model for many-to-one operations such as those involved in our `MPI_Gather` [37]. This model differentiates between small, medium and large messages by introducing two threshold parameters, T_1 and T_2. For small messages (of size below T_1), the execution time is shown to respond linearly to the increase of message size. For medium-sized messages (of size between T_1 and T_2), a very significant degradation in performance due to non-linear effects is found empirically. Finally, it is shown that for large messages (of size larger than T_2), the execution time resumes linear predictability for increasing message sizes. In our application we have experimentally tested that, when a small number of processors is used, the message size fits into the area of large messages. Quite opposite, as the number of processors is increased, the message size fits the area of medium-sized messages, which ultimately results in a significant increase in execution times of the single `MPI_Gather` operation.

In order to address this relevant issue, we can redesign our parallel algorithms as follows. If we replace the single `MPI_Gather` operation used to gather medium-sized messages by an equivalent sequence of `MPI_Gather` operations, each gathering messages with a size that fits the range of small messages, then each medium-sized partition can be communicated as a sequence of small-sized sub-partitions. To achieve this goal, one option is to calculate the number of sub-partitions s of a partition of medium size, S, so that $\frac{S}{s} \leqslant T_1$ and $\frac{S}{s-1} > T_1$. An alternative option to avoid the congestion region relies on the fact that the system settings for MPI and the TCP/IP flags can be modified to allow a reservation mode to begin for smaller messages. If larger messages are required, then the non-buffered mode can be selected for MPI by setting the system buffer size to a smaller value. In this case, sporadic non-linear behavior may still be observed, but it is likely to occur at negligible probability [37]. Although we are planning on carefully exploring the above-mentioned issues in future work, our current access restrictions to configuration parameters and settings for the Thunderhead system, a US Government facility, prevented us from properly implementing the two above-mentioned optimizations at this point.

Table 7
Execution times for parallel algorithms using different numbers of processors on Thunderhead (column "1" reports the times obtained by real sequential versions of the algorithms run on a single node)

Algorithm	1	4	16	36	64	100	144	196	256	324	400
P-PCT	1246	366	111	51	26	18	14	11	10	9	8
P-PPI	2745	1013	251	99	52	34	24	18	15	13	12
P-FINDR	695	199	46	24	13	9	6	5	4	3	3
P-IEA	2865	952	234	88	49	33	23	18	14	13	11
P-AMEE	1874	580	132	53	30	20	14	11	8	7	6
P-LSU	2163	577	142	61	34	22	16	12	9	7	6

To conclude this subsection, Table 5 reports the measured execution times for the considered parallel algorithms, using different numbers of processors on Thunderhead. Results in Table 5 reveal that some of the tested algorithms were able to obtain highly accurate hyperspectral analysis results (in light of Table 2), but also quickly enough for practical use. For instance, using 400 processors the P-AMEE algorithm (followed by P-LSU) provided highly accurate abundance estimations in only 12 s, while the combination of P-PCT for dimensionality reduction, P-FINDR for endmember extraction, and P-LSU for spectral unmixing was able to provide a result in 20 s. The P-IEA was able to produce a response in 11 s, with the slight advantage over P-AMEE that this parallel approach does not need to re-calculate all the endmembers again in case the number of desired endmembers to be extracted by the algorithm is changed by the user. The above results indicate significant improvements over commonly used processing strategies for this kind of high-dimensional data sets, which can take up to more than one hour of computation for the considered problem size, as indicated by the column labeled as "1" in Table 7, which reports the execution times measured for real sequential versions of the considered parallel algorithms, executed in a single Thunderhead node.

Summarizing, experimental results in this study reveal that parallel algorithms able to efficiently distribute the workload among a set of heterogeneous processors offer a simple, platform-independent and moderately scalable solution in the context of realistic hyperspectral imaging applications. Although scalability issues have been identified (in particular, for a very large number of processors) and should be addressed in future work, our quantitative and comparative assessment of different parallel strategies for efficiently implementing a well-consolidated hyperspectral data processing chain provides several interesting findings. For instance, our study reveals that joint spatial/spectral techniques are indeed effective for parallel implementation, not only due to the window-based nature of such algorithms, but also because they can reduce sequential computations at the master node and involve only minimal communication between the parallel tasks, namely, at the beginning and ending of such tasks. We also feel that the applicability of the proposed parallel methods may extend beyond the domain of hyperspectral data processing. This is particularly true for the domains of signal processing and linear algebra applications, which include similar patterns of communication and calculation.

5. Conclusions and future work

The aim of this paper has been the examination of different parallel strategies for hyperspectral analysis on heterogeneous and homogeneous computing platforms, with the purpose of evaluating the possibility of obtaining results in valid response times and with adequate reliability in heterogeneous computing environments where these techniques are intended to be applied. In particular, this paper provided a detailed discussion on the effects that platform heterogeneity has on degrading parallel performance of hyperspectral analysis algorithms. An interesting finding by experiments is that spatial/spectral heterogeneous approaches with redundant computations offer a surprisingly simple, yet effective and scalable solution for solving the endmember extraction problem, which is the most computationally demanding task in the entire hyperspectral data analysis chain. Our experimental results revealed important algorithmic aspects that may be of great importance for designing and adapting existing high-performance hyperspectral imaging applications (often developed in the context of homogeneous computing platforms) to fully heterogeneous computing environments, which are currently the tool of choice in many remote sensing and earth exploration missions. Combining this readily available computational power with last-generation sensor and parallel processing technology may

introduce substantial changes in the systems currently used by NASA and other agencies for exploiting earth and planetary remotely sensed data. As future work, we plan to incorporate optimizations to enhance scalability of the proposed algorithms when implemented in massively parallel systems. We also aim at implementing the parallel algorithms on other massively parallel computing architectures, such as NASA's Project Columbia or CEPBA's Mare Nostrum supercomputer. Finally, we are also working towards field programmable gate array (FPGA)-based and graphic processing unit (GPU)-based implementations, which may allow us to fully accomplish the goal of near real-time processing of hyperspectral image data, with potential applications in exploitation-based on-board hyperspectral data compression and analysis.

Acknowledgements

This research was supported by the European Commission through the Marie Curie project entitled "Hyperspectral Imaging Network," (MRTN–CT–2006–035927). The authors would like to thank Drs. John E. Dorband, James C. Tilton and J. Anthony Gualtieri for many helpful discussions. They would also like to state their appreciation for Profs. Mateo Valero and Francisco Tirado. The first author acknowledges support received from the Spanish Ministry of Education and Science (Fellowship PR2003–0360), which allowed him to conduct postdoctoral research at NASA's Goddard Space Flight Center and University of Maryland in 2004.

References

[1] C.-I. Chang, Hyperspectral Imaging: Techniques for Spectral Detection and Classification, Kluwer Academic Publishers, New York, 2003.
[2] R.O. Green et al., Imaging spectroscopy and the airborne visible/infrared imaging spectrometer AVIRIS, Remote Sensing of Environment 65 (1998) 227–248.
[3] A. Plaza, C.-I. Chang, High Performance Computing in Remote Sensing, Chapman & Hall/CRC Press, Boca Raton, FL, 2007.
[4] G. Aloisio, M. Cafaro, A dynamic earth observation system, Parallel Computing 29 (2003) 1357–1362.
[5] P. Wang, K.Y. Liu, T. Cwik, R.O. Green, MODTRAN on supercomputers and parallel computers, Parallel Computing 28 (2002) 53–64.
[6] J. Dorband, J. Palencia, U. Ranawake, Commodity computing clusters at Goddard Space Flight Center, Journal of Space Communication 1 (2003) (available online: <http://satjournal.tcom.ohio.edu/pdf/Dorband.pdf>).
[7] R. Brightwell, L.A. Fisk, D.S. Greenberg, T. Hudson, M. Levenhagen, A.B. Maccabe, R. Riesen, Massively parallel computing using commodity components, Parallel Computing 26 (2000) 243–266.
[8] K.A. Hawick, P.D. Coddington, H.A. James, Distributed frameworks and parallel algorithms for processing large-scale geographic data, Parallel Computing 29 (2003) 1297–1333.
[9] A. Lastovetsky, Parallel Computing on Heterogeneous Networks, Wiley-Interscience, Hoboken, NJ, 2003.
[10] A. Kalinov, A. Lastovetsky, Y. Robert, Heterogeneous computing, Parallel computing 31 (2005) 649–652.
[11] A. Lastovetsky, R. Reddy, On performance analysis of heterogeneous algorithms, Parallel Computing 30 (2004) 1195–1216.
[12] D. Heinz, C.-I. Chang, Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery, IEEE Transactions on Geoscience and Remote Sensing 39 (2001) 529–545.
[13] A. Plaza, P. Martínez, R. Pérez, J. Plaza, A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data, IEEE Transactions on Geoscience and Remote Sensing 42 (2004) 650–663.
[14] 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.
[15] M.E. Winter, N-FINDR: an algorithm for fast autonomous spectral endmember determination in hyperspectral data, in: Proceedings of SPIE Imaging Spectrometry Conference, vol. 3753, pp. 266–277, 1999.
[16] R.A. Neville, K. Staenz, T. Szeredi, J. Lefebvre, P. Hauff, Automatic endmember extraction from hyperspectral data for mineral exploration, in: Proceedings 21st Canadian Symposium on Remote Sensing, Ontario, Canada, pp. 21–24, 1999.
[17] A. Plaza, P. Martinez, R.M. Perez, J. Plaza, Spatial/spectral endmember extraction by multidimensional morphological operations, IEEE Transactions on Geoscience and Remote Sensing 40 (2002) 2025–2041.
[18] P. Soille, Morphological Image Analysis: Principles and Applications, second ed., Springer, Berlin, 2003.
[19] C. Nicolescu, P. Jonker, Data and task parallel image processing environment, Parallel Computing 28 (2002) 945–965.
[20] K. Sano, Y. Kobayashi, T. Nakamura, Differential coding scheme for efficient parallel image composition on a PC cluster system, Parallel Computing 30 (2004) 285–299.
[21] B. Veeravalli, S. Ranganath, Theoretical and experimental study on large size image processing applications using divisible load paradigm on distributed bus networks, Image and Vision Computing 20 (2003) 917–935.
[22] A. Plaza, D. Valencia, J. Plaza, P. Martinez, Commodity cluster-based parallel processing of hyperspectral imagery, Journal of Parallel and Distributed Computing 66 (2006) 345–358.

[23] T.D. Braun, H.J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D.A. Hensgen, R.F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (2001) 810–837.
[24] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, Scheduling strategies for master–slave tasking on heterogeneous processor platforms, IEEE Transactions on Parallel and Distributed Systems 15 (2004) 319–330.
[25] V.E. Bazterra, M. Cuma, M.B. Ferraro, J.C. Facelli, A general framework to understand parallel performance in heterogeneous clusters: analysis of a new adaptive parallel genetic algorithm, Journal of Parallel and Distributed Computing 65 (2005) 48–57.
[26] T. Achalakul, S. Taylor, A distributed spectral-screening PCT algorithm, Journal of Parallel and Distributed Computing 63 (2003) 373–384.
[27] T. El-Ghazawi, S. Kaewpijit, J. Le Moigne, Parallel and adaptive reduction of hyperspectral data to intrinsic dimensionality, in: Proceedings of the 2001 IEEE International Conference on Cluster Computing (Cluster'01), IEEE Computer Society, 2001.
[28] M.E. Winter, A proof of the N-FINDR algorithm for the automated detection of endmembers in a hyperspectral image, in: Proceedings of SPIE, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery X, vol. 5425, pp. 31–41, 2004.
[29] C.I. Chang, C.-C. Wu, W.-M. Liu, Y.-C. Ouyang, A new growing method for simplex-based endmember extraction algorithm, IEEE Transactions on Geoscience and Remote Sensing 44 (2006) 2804–2819.
[30] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., Baltimore, Johns Hopkins, 1996.
[31] A. Plaza, J. Plaza, D. Valencia, Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data, Journal of Supercomputing 40 (2007) 81–107.
[32] F.J. Seinstra, D. Koelma, J.M. Geusebroek, A software architecture for user transparent parallel image processing, Parallel Computing 28 (2002) 967–993.
[33] A. Plaza, D. Valencia, J. Plaza, C.-I. Chang, Parallel implementation of endmember extraction algorithms from hyperspectral data, IEEE Geoscience and Remote Sensing Letters 3 (2006) 334–338.
[34] X.-H. Sun, Scalability versus execution time in scalable systems, Journal of Parallel and Distributed Computing 62 (2002) 173–192.
[35] A. Kalinov, Scalability analysis of matrix–matrix multiplication on heterogeneous clusters, in: Proceedings of ISPDC'2004/ HeteroPar'04, IEEE Computer Society, 2004.
[36] M.J. Martín, D.E. Singh, J.C. Mouriño, F.F. Rivera, R. Doallo, J.D. Bruguera, High performance air pollution modeling for a power plan environment, Parallel Computing 29 (2003) 1763–1790.
[37] A. Lastovetsky, M. O'Flynn, A performance model of many-to-one collective communications for parallel computing, in: Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, California, IEEE Computer Society, 2007.