Parallel Processing of Remotely Sensed Hyperspectral Images On Heterogeneous Networks of Workstations Using HeteroMPI

David Valencia, Alexey Lastovetsky, Maureen O'Flynn, Antonio Plaza and Javier Plaza

International Journal of High Performance Computing Applications 2008; 22; 386
DOI: 10.1177/1094342007088377

PARALLEL PROCESSING OF REMOTELY SENSED HYPERSPECTRAL IMAGES ON HETEROGENEOUS NETWORKS OF WORKSTATIONS USING HETEROMPI

David Valencia[1]
Alexey Lastovetsky[2]
Maureen O'Flynn[2]
Antonio Plaza[1]
Javier Plaza[1]

Abstract

The development of efficient techniques for transforming massive volumes of remotely sensed hyperspectral data into scientific understanding is critical for space-based Earth science and planetary exploration. Although most available parallel processing strategies for information extraction and mining from hyperspectral imagery assume homogeneity in the underlying computing platform, heterogeneous networks of computers (HNOCs) have become a promising cost-effective solution, expected to play a major role in many on-going and planned remote sensing missions. In this paper, we develop a new morphological parallel algorithm for hyperspectral image classification using HeteroMPI, an extension of MPI for programming high-performance computations on HNOCs. The main idea of HeteroMPI is to automate and optimize the selection of a group of processes that executes a heterogeneous algorithm faster than any other possible group in a heterogeneous environment. In order to analyze the impact of many-to-one (gather) communication operations introduced by our proposed algorithm, we resort to a recently proposed collective communication model. The parallel algorithm is validated using two heterogeneous clusters at University College Dublin and a massively parallel Beowulf cluster at NASA's Goddard Space Flight Center.

Key words: heterogeneous parallel computing, hyperspectral image processing, HeteroMPI, mathematical morphology, performance evaluation

1 Introduction

Hyperspectral imaging identifies materials and objects in the air, land and water on the basis of the unique reflectance patterns that result from the interaction of solar energy with the molecular structure of the material (Chang 2003). Most applications of this technology require timely responses for swift decisions which depend upon high computing performance of algorithm analysis. Examples include target detection for military and defense/security deployment, urban planning and management, risk/hazard prevention and response including wild-land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination. The concept of hyperspectral imaging was introduced when NASA's Jet Propulsion Laboratory developed the Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) system, which covers the wavelength region from 0.4 to 2.5 μm using 224 spectral channels (see Figure 1). This imager is able to continuously produce snapshot image cubes of tens or even hundreds of kilometers long, each of them with hundreds of MB in size, and this explosion in the amount of collected information has rapidly introduced new processing challenges (Plaza et al. 2006).

Although most dedicated parallel machines for remote sensing data analysis employed by NASA and other institutions during the last decade have been chiefly homogeneous in nature (Dorband, Palencia, and Ranawake 2003), computing on heterogeneous networks of computers (HNOCs) has soon become a viable alternative to expensive parallel computing systems (Lastovetsky 2003). These networks enable the use of existing resources and provide incremental scalability of hardware components. At the same time, HNOCs can achieve high communication speed at low cost, using switch-based networks such as ATMs, as well as distributed service and support, especially for large file systems.

Despite the growing interest in hyperspectral imaging research, only a few consolidated parallel techniques for analyzing this kind of data currently exist in the open literature. However, with the recent explosion in the amount and dimensionality of hyperspectral data, parallel processing is expected to become a requirement in most ongoing and planned remote sensing missions. As a result, this paper takes a necessary first step toward the development of parallel hyperspectral imaging techniques on HNOCs.

[1]DEPARTMENT OF TECHNOLOGY OF COMPUTERS AND COMMUNICATIONS, TECHNICAL SCHOOL OF CÁCERES, UNIVERSITY OF EXTREMADURA, E-10071 CÁCERES, SPAIN

[2]HETEROGENEOUS COMPUTING LABORATORY, SCHOOL OF COMPUTER SCIENCE AND INFORMATICS, UNIVERSITY COLLEGE DUBLIN, BELFIELD, DUBLIN 4, IRELAND (ALEXEY.LASTOVETSKY@UCD.IE)

The International Journal of High Performance Computing Applications, Volume 22, No. 4, Winter 2008, pp. 386–407
DOI: 10.1177/1094342007088377

Figure 6 appears in color online: http://hpc.sagepub.com

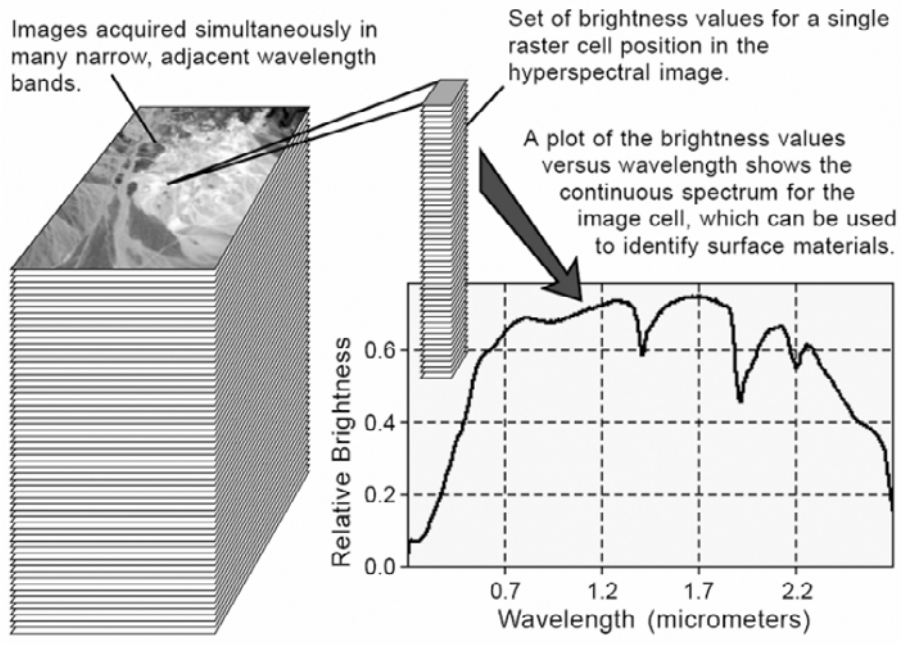

Fig. 1 The concept of hyperspectral imaging using NASA/Jet Propulsion Laboratory's AVIRIS system.

Although the standard MPI (Dongarra et al. 1996) has been widely used to implement parallel algorithms for HNOCs in the past, it does not provide specific means to address some additional challenges posed by these networks, including the distribution of computations and communications unevenly, taking into account the computing power of the heterogeneous processors and the bandwidth of the communications links. To achieve the above goals, HeteroMPI was developed as an extension of MPI which allows the programmer to describe the performance model of a parallel algorithm in generic fashion (Lastovetsky and Reddy 2006). This is a highly desirable feature in hyperspectral imaging applications, in which the main features of the underlying parallel algorithm have an essential impact on execution performance.

In this paper, our main goal is to develop an advanced heterogeneous parallel algorithm for hyperspectral image processing using HeteroMPI. The paper is structured as follows. Section 2 first describes previous efforts concerned with the design of parallel hyperspectral imaging algorithms in the literature, and then outlines the main features of HeteroMPI. Section 3 describes the hyperspectral-imaging algorithm considered in this study, which performs joint spatial and spectral analysis of the data in combined fashion, and further develops HeteroMPI-based parallel implementations of the algorithm. Section 4 assesses the performance of the parallel heterogeneous algorithm by analyzing its accuracy and parallel properties on a heterogeneous cluster and also on a homogeneous one. This section also evaluates the communication framework adopted for the proposed parallel code and, in particular, the impact of many-to-one communications on the overall performance, in light of a recently proposed collective communication model (Lastovetsky, Mkwawa, and O'Flynn 2006). Section 5 addresses the effectiveness of the HeteroMPI-based implementation. Finally, Section 6 concludes with some remarks and hints at plausible future research.

2 Related Work

This section first provides an overview of previous research in the area of parallel hyperspectral imaging. As will be shown by our overview, most available parallel approaches have been specifically designed for homogeneous clusters. In order to address the need for efficient implementations in heterogeneous platforms, this section also provides an outline of HeteroMPI, a heterogeneous version of MPI that will be used to develop a new heterogeneous algorithm for hyperspectral image classification.

2.1 Parallel Hyperspectral Algorithms

Despite the growing interest in hyperspectral imaging, only a few research efforts devoted to the design of paral-

lel implementations currently exist in the open literature. It should be noted that some available parallel techniques are subject to non-disclosure restrictions, mainly because of their use in military and defense applications. However, with the recent explosion in the amount of hyperspectral imagery, parallel processing has now become a requirement in most remote sensing applications.

The utilization of parallel systems hyperspectral imaging applications has become increasingly widespread in recent years. The idea of using COTS (commercial off-the-shelf) computer equipment, clustered together to work as a computational team (Brightwell et al. 2000), was first explored to address the extremely high computational requirements introduced by Earth observation applications. This strategy, often referred to as Beowulf-class cluster computing, has already offered access to greatly increased computational power, but at a low cost (commensurate with falling commercial PC costs) in a number of remote sensing applications (Kalluri et al. 2001; Wang et al. 2002; Le Moigne, Campbell, and Cromp 2002; Plaza and Chang 2007). In particular, NASA is actively supporting massively parallel clusters for remote sensing studies including those involving hyperspectral imagery. An example is Thunderhead, a 512-processor homogeneous Beowulf cluster at NASA's Goddard Space Flight Center in Maryland (see http://thunderhead.gsfc.nasa.gov for details). Another example is the Columbia supercomputer at NASA Ames Research Center, a 10,240-CPU SGI Altix supercomputer, with Intel Itanium-2 processors, 20 terabytes of total memory and heterogeneous interconnects including InfiniBand network and 10-gigabit Ethernet.

Several hyperspectral imaging algorithms have been implemented in the system described above using MPI as a standard development tool. Examples include the distributed spectral-screening principal component transform algorithm (S-PCT; Achalakul and Taylor 2003), which makes use of the principal component transform (PCT) to summarize and decorrelate the images by reducing redundancy and packing the residual information into a small set of images, termed *principal components*. The algorithm uses a standard master–slave decomposition technique, where the master coordinates the actions of the workers, gathers the partial results from them and provides the final result. Another example of Beowulf cluster-based parallel algorithm in the literature is D-ISODATA (Dhodhi et al. 1999), designed as the first parallel approach able to deal with the entire high-dimensional volume directly, thereby preserving all the spectral information in the data. It should be noted that the ISODATA classification procedure is widely regarded as a benchmark for most unsupervised classification algorithms (Richards and Jia 2005).

A shortcoming of both S-PCT and D-ISODATA is that these algorithms rely on using the spectral information alone, without taking into account the spatial arrangement of pixels. In contrast, the hierarchical image segmentation algorithm (HSEG; Tilton 2005) has been recently proposed as a hybrid method able to use the spatial and the spectral information in the analysis of multichannel images. To counteract the extremely high computational complexity of the algorithm, a computationally efficient recursive approximation of HSEG (called RHSEG) was first developed and later transformed into an efficient MPI-based implementation by regularly allocating processing tasks among available CPUs (Tilton 2007). Most recently, a morphological approach for classification of hyperspectral images has been developed. The algorithm, called automated morphological classification (AMC), takes into account both the spatial and the spectral information in the analysis in a combined fashion, as opposed to HSEG which first uses spectral information to produce an initial segmentation and then refines the segmentation using spatial context. An MPI-based parallel version of AMC has been developed and tested on NASA's Thunderhead cluster, showing parallel performance results superior to those achieved by other parallel hyperspectral algorithms in the literature (Plaza et al. 2006).

An important limitation in the above-mentioned parallel techniques is that they assume that the number and location of nodes are known and relatively fixed. However, the commercial availability of high-performance networking hardware has made it possible to develop distributed parallel systems made up of networked groups of machines distributed among different locations. Current remote sensing applications, constrained by the ever-growing dimensionality and size of the collected image data, can greatly benefit from this concept of distributed computing on HNOCs. In this regard, HeteroMPI offers an excellent tool to develop parallel algorithms specifically adapted to heterogeneous platforms, and also to transform available parallel hyperspectral algorithms into efficient implementations for these systems.

2.2 Outline of HeteroMPI

The standard MPI specification provides communication and group constructors which allow the application programmer to create a group of processes explicitly chosen from an ordered set (Dongarra et al. 1996). This approach is feasible when the application is run on a homogeneous distributed-memory computer system. However, selection of a group for execution on HNOCs must take into account the computing power of the heterogeneous processors and the speed/bandwidth of communication links between each processor pair (Lastovetsky and Reddy 2006). This feature is of particular importance in applications dominated by large data volumes such as hyperspectral image analysis, but is also quite difficult to accomplish from the viewpoint of the programmer.

The main idea of HeteroMPI is to automate and optimize the selection of a group of processes that executes a heterogeneous algorithm faster than any other possible group. For this purpose, HeteroMPI provides a small and dedicated definition language for the specification of such performance model. This language is a subset of mpC, defined by Lastovetsky (2002), and allows the programmer to explicitly define an *abstract* network and distribute data, computations and communications over the network. Then, HeteroMPI automatically maps (at run time) the abstract network to a *real* execution network by dynamically adapting the performance model to specific network parameters such as the computing power of processors or the capacities of communication links in the real environment. By means of a compiler, the description of a performance model is translated into a set of functions that make up an algorithm-specific part of the HeteroMPI runtime system. Below, we provide a brief outline of the most important HeteroMPI functions which have been used to implement the proposed parallel algorithm. Detailed information about these and other HeteroMPI functions is given in Lastovetsky and Reddy (2006).

A typical HeteroMPI application starts with the initialization of the runtime system using the operation:

```
HeteroMPI_Init(int argc, char **argv)
```

This routine must be called once by all the processes running in the application. After the initialization, application programmers can call any other HeteroMPI routines. For instance, the following function is used to create a group that will execute the heterogeneous algorithm faster than any other group of processes:

```
HeteroMPI_Group_create(HeteroMPI_Group *gid,
    const HeteroMPI_Model *perf_model,
    const void *model_parameters,
    int param_count)
```

This function returns a handle `gid` to the group of MPI processes. Here, `perf_model` encapsulates the features of the performance model; `model_parameters` are the actual parameters of the performance model; and `param_count` is the total number of parameters. After the execution of this function, the performances `opt_speeds` can be obtained by using the HeteroMPI group accessor function shown below:

```
HeteroMPI_Group_performances(&gid, opt_speeds)
```

It is important to emphasize at this point that the accuracy of the performance model depends heavily on the accuracy of the estimation of the actual speeds of the processors. For that purpose, HeteroMPI provides a function to dynamically update the estimation of processor speeds at runtime:

```
HeteroMPI_Recon(HeteroMPI_Benchmarkfunction b,
    const void *input_p, int num_of_parameters,
    void *output_p)
```

where all the processors execute the benchmark function `b` in parallel. This is a collective operation and must be called by all the processes in the group associated with a predefined communication universe `HeteroMPI_COMM_WORLD` of HeteroMPI. A similar comment applies to the group destructor operation provided by HeteroMPI:

```
HeteroMPI_Group_free(HeteroMPI_Group *gid)
```

where `gid` is the HeteroMPI handle to the group of MPI processes. Again, this is a collective operation that must be called by all members of this group. There are no analogs of other group constructors of MPI such as the set-like operations on groups and the range operations on groups in HeteroMPI. This is because: 1) HeteroMPI does not guarantee that groups composed using these operations can execute a logical unit of parallel algorithm faster than any other group of processes; and 2) it is relatively straightforward for application programmers to perform such group operations by obtaining the groups associated with the MPI communicator given by the `Hetero_MPI_Get_comm` operation shown below:

```
const MPI_Comm* HeteroMPI_Get_comm
    (const HeteroMPI_Group *gid)
```

which returns an MPI communicator with a communication group of MPI processes defined by `gid`. This is a local operation not requiring inter-process communication. Application programmers can use this communicator to call the standard MPI communication routines during the execution of the parallel algorithm. This communicator can safely be used in other MPI routines. In order to finalize the runtime system, the following operation is used:

```
HeteroMPI_Finalize(int exitcode)
```

3 Parallel Hyperspectral Algorithm

This section describes a parallel heterogeneous version of the AMC algorithm for automated morphological analysis of hyperspectral image data on HNOCs. The section is organized as follows. First, we describe the standard morphological algorithm. Next, we outline important aspects of its parallel implementation such as data partitioning and communication issues.

3.1 Morphological Algorithm

Morphological analysis has been successfully used in previous research to analyze hyperspectral data sets (Soille 2003; Plaza et al. 2005). The morphological algorithm selected in this work as a representative case study takes into account both the spatial and spectral information of the data in simultaneous fashion. Such spatial/spectral, hybrid techniques represent the most advanced generation of hyperspectral imaging algorithms currently available. Before describing our proposed approach, let us first denote by $\boldsymbol{f}$ a hyperspectral data set defined on an L-dimensional (L-D) space, where N is the number of channels or spectral bands. The main idea of the algorithm is to impose an ordering relation in terms of spectral purity in the set of pixel vectors lying within a spatial search window or *structuring element* (SE) around each image pixel vector (Plaza et al. 2005). To do so, we first define a cumulative distance between one particular pixel $\boldsymbol{f}(x, y)$, where $\boldsymbol{f}(x, y)$ denotes an L-D vector at discrete spatial coordinates $(x, y) \in Z^2$, and all the pixel vectors in the spatial neighborhood given by a SE denoted by B (B-neighborhood) as follows:

$$D_B[\boldsymbol{f}(x, y)] = \sum_i \sum_j \text{SAM}[\boldsymbol{f}(x, y), \boldsymbol{f}(i, j)],$$

where (i, j) are the spatial coordinates in the B-neighborhood and SAM is the spectral angle mapper (Chang 2003):

$$\text{SAM}(\boldsymbol{f}(x, y), \boldsymbol{f}(i, j)) = \cos^{-1}\left(\frac{\boldsymbol{f}(x, y) \cdot \boldsymbol{f}(i, j)}{\|\boldsymbol{f}(x, y)\| \cdot \|\boldsymbol{f}(i, j)\|}\right)$$

Based on the distance above, we calculate the extended morphological erosion of $\boldsymbol{f}$ by B (Plaza et al. 2002) for each pixel in the input data scene as follows:

$$(\boldsymbol{f} \ominus B)(x, y) = \arg\min_{(i, j)}\{D_B[\boldsymbol{f}(x + i, y + j)]\}$$

where the argmin operator selects the pixel vector that is most highly similar, spectrally, to all the other pixels in the B-neighborhood. On the other hand, the extended morphological dilation of $\boldsymbol{f}$ by B (Plaza et al. 2002) is calculated as follows:

$$(\boldsymbol{f} \oplus B)(x, y) = \arg\max_{(i, j)}\{D_B[\boldsymbol{f}(x + i, y + j)]\}$$

where the argmax operator selects the pixel vector that is most spectrally distinct to all the other pixels in the B-neighborhood. With the above definitions in mind, we provide below an unsupervised classification algorithm for hyperspectral imagery based on extended morphological operations. One of the main features of the algorithm above is regularity in the computations. As shown in previous work (Plaza et al. 2006), its computational complexity is $O(p_f \times p_B \times I_{max} \times N)$, where p_f is the number of pixels in $\boldsymbol{f}$ and p_B is the number of pixels in B. This results in high computational cost in real applications. However, an adequate parallelization strategy can greatly enhance the computational performance of the algorithm, as outlined in the following subsection. The inputs to our algorithm, called automated morphological classification (AMC), are a hyperspectral data cube $\boldsymbol{f}$, a morphological SE with constant size of 3 × 3 pixels, B, a number of classes, c, and a number of iterations, I_{max}. The output is a 2-D matrix which contains a classification label for each pixel vector $\boldsymbol{f}(x, y)$ in the input image. The AMC algorithm can be summarized by the following steps:

1. Set $i = 1$ and initialize a morphological eccentricity index score $\text{MEI}(x, y) = 0$ for each pixel.
2. Move B through all the pixels of $\boldsymbol{f}$, defining a local spatial search area around each $\boldsymbol{f}(x, y)$, and calculate the maximum and the minimum pixels at each B-neighborhood using dilation and erosion, respectively. Update the MEI at each pixel using the SAM between the maximum and the minimum.
3. Set $i = 1 + 1$. If $i = I_{max}$ then go to step 4. Otherwise, replace $\boldsymbol{f}$ by its dilation using B, and go to step 2.
4. Select the set of c pixel vectors in $\boldsymbol{f}$ with higher associated score in the resulting MEI image and estimate the sub-pixel abundance $\alpha_i(x, y)$ of those pixels at $\boldsymbol{f}(x, y)$ using the standard linear mixture model described in (Chang 2003).
5. Obtain a classification label for each pixel $\boldsymbol{f}(x, y)$ by assigning it to the class with the highest sub-pixel fractional abundance score in that pixel. This is done by comparing all estimated abundance fractions $\{\alpha_1(x, y), \alpha_2(x, y), \ldots, \alpha_c(x, y)\}$ and finding the one with the maximum value, say $\alpha_{i^*}(x, y)$, with

$$i^* = \arg\left\{\max_{1 \le i \le c}\{\alpha_i(x, y)\}\right\}.$$

3.2 Data Partitioning

Two types of parallelism can be exploited in hyperspectral image analysis algorithms: spatial-domain parallelism and spectral-domain parallelism (Plaza et al. 2006). Spatial-domain parallelism subdivides the image into multiple blocks made up of entire pixel vectors, and assigns one or more blocks to each processor. Spectral-domain parallelism subdivides the hyperspectral data into blocks made up of contiguous spectral bands (sub-volumes), and assigns one or more sub-volumes to each

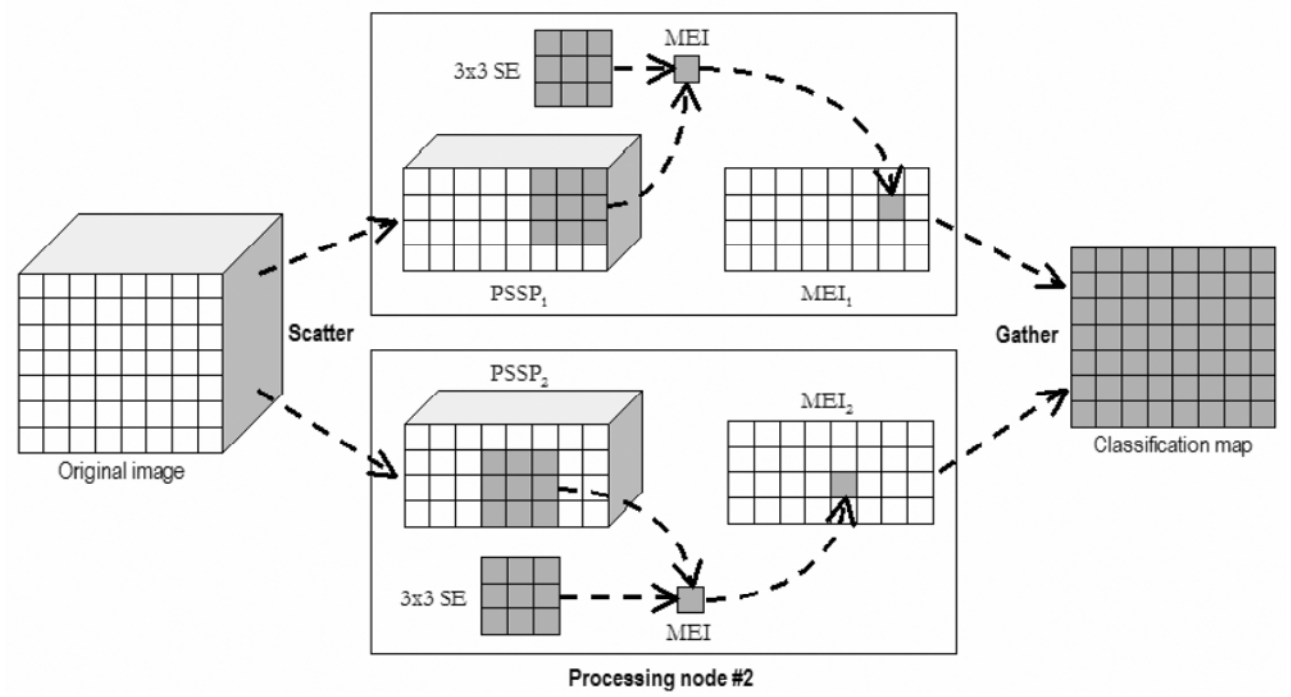

Fig. 2 Example structuring element (SE)-based morphological computation performed using two processing units.

processor. The latter approach breaks the spectral identity of the data because each pixel vector is split amongst several processing units, and operations such as morphological erosion and dilation would need to originate from several processors, thus requiring intensive inter-processor communication. In this work, we use spatial-domain parallelism in order to preserve the entire spectral information of each image pixel (see Figure 2). This is a natural approach for low-level image processing, as many operations require the same function to be applied to a small set of elements around each data element present in the image data structure.

With the above ideas in mind, the main goal of our parallelization framework for the AMC algorithm is to use a low-level image processing-oriented approach, in which each heterogeneous processor will be able to process a spatial/spectral data partition *locally*. In previous work, we have defined the concept of parallelizable spatial/spectral partition (PSSP) as a hyperspectral data partition that can be processed independently at each processing node (Plaza et al. 2006). Here, we use the concept of PSSP above to define a virtual processor grid organization in which processors apply the AMC algorithm locally to each partition, thus producing a set of *local* classification outputs which are then combined to form a *global* classification output. In order to adequately exploit the concept of PSSP introduced above, two important issues need to be taken into account:

1. An important issue in SE-based morphological image processing operations is that accesses to pixels outside the spatial domain of the input image are possible. This is particularly so when the SE is centered on a pixel located in the border of the original image. In sequential implementations, it is common practice to redirect such accesses according to a predefined border handling strategy. In our application, a border handling strategy is adopted when the location of the SE is such that some of the pixel positions in the SE are outside the input image domain (see Figure 3). In this situation, only those pixels inside the image domain are read for the MEI calculation. This strategy is equivalent to the common mirroring technique used in digital image processing applications, but slightly faster since fewer pixels are involved in the SE calculation.
2. Apart from the border handling strategy above, a communication overhead is introduced when the SE computation is split amongst several different processing nodes (see Figure 4). It should be

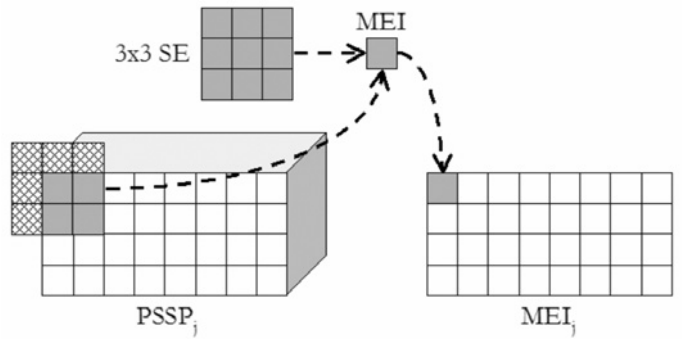

Fig. 3 Border-handling strategy implemented on a PSSP when pixels lying outside the input image domain are required for the SE-based morphological computation.

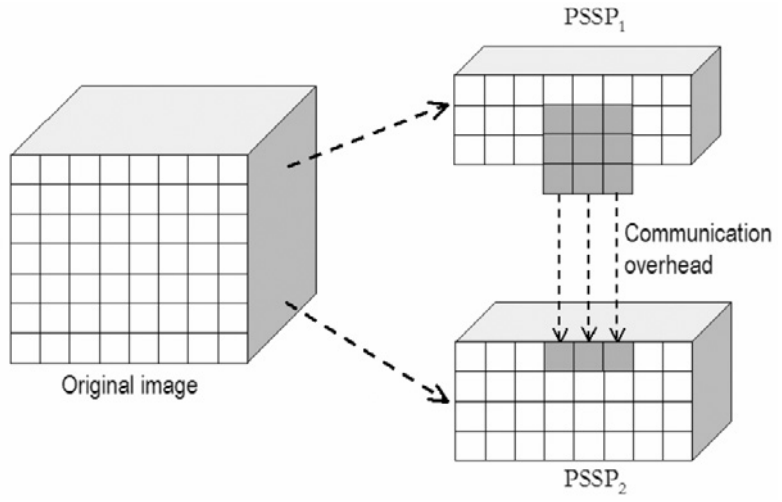

Fig. 4 Communication overhead introduced for SE-based operation split between two adjacent processing nodes.

noted that Figure 4 gives a simplified view. Depending on how many adjacent PSSPs are involved in the parallel computation of a SE, it may be necessary to introduce additional communication patterns. In this regard, it is important to emphasize that the amount of redundant information introduced at each partition after the communication depends on the size of B, the SE used in the morphological operations.

Our implementation of the AMC algorithm always uses a constant 3×3-pixel SE through the different iterations (see Section 3.1). In other words, instead of increasing the size of the SE to consider a larger spatial neighborhood, we replace the original image cube $\boldsymbol{f}$ or, equivalently, the local PSSP in parallel processing, by the resulting cube after applying a dilation operation using B (see step 3 of the AMC algorithm). This allows us to perform multi-scale analysis of the data without increasing significantly the communication overhead (Soille 2003; Plaza et al. 2005).

One of the main challenges for the design of a heterogeneous version of the parallel algorithm described above is to find an optimal mapping of PSSPs on the virtual grid of processors such that the size of the resulting partitions is in accordance with the computing power of heterogeneous processors. Another challenge is to efficiently handle interprocessor communications. These aspects will be accomplished through the definition of a suitable performance model, as indicated by the following section.

4 Heterogeneous Parallel Implementation

This section describes the heterogeneous parallel implementation of the AMC algorithm developed in the previous section using HeteroMPI. The section is organized as follows. First, we define a performance model for the proposed algorithm and provide a detailed description of the core benchmark which constitutes the basis of such performance model. Our discussion includes considerations about how memory-related parameters are incorporated into the benchmark function. Then, we provide a description of the communication framework adopted for the parallel implementation. The section concludes with an overview of the HeteroMPI implementation of the AMC algorithm, outlining the most relevant aspects of the parallel code.

4.1 Performance Model and Definition of a Benchmarking Function

The considered application case study is an example of a regular problem decomposition in which the whole program can be decomposed into a large set of small equivalent programs, running in parallel and interacting via message passing. The main idea for efficiently solving a regular problem is to reduce it to an irregular problem, the structure of which is determined by the irregularity of underlying hardware rather than the irregularity of the problem itself. In order to implement the parallel morphological algorithm outlined above using HeteroMPI, the first step is to define a performance model able to capture the data partitioning and communication framework described in the previous subsection (Valencia, Lastovetsky, and Plaza 2006). In the following, we summarize the most important fragments of the mpC-based code that describes the adopted performance model, which has several input parameters:

- Parameter `m` specifies the number samples of the data cube.
- Parameter `n` specifies the number of lines.
- Parameters `se_size` and `iter` respectively denote the size of the SE and the number of iterations executed by the algorithm.
- Parameters `p` and `q` indicate the dimensions of the computational grid (in columns and rows, respectively), which are used to map the spatial coordinates of the individual processors within the processor grid layout.
- Finally, parameter `partition_size` is an array that indicates the size of the local PSSPs (calculated automatically using the relative estimated computing power of the heterogeneous processors using the `benchmark` function).

```
algorithm amc_perf (int m, int n, int se_size,
    int iter, int p, int q,
    int partition_size[p*q]) {
coord I=p, J=q;//p is the number of columns;
    q is the number of rows
node { I>=0 && J>=0:
    benchmark*((partition_size[I*q+J]*iter);};
parent[0,0];
}
```

It should be noted that some of the definitions have been removed from the code above for simplicity. However, the most representative sections are included. Keyword **`algorithm`** begins the specification of the performance model followed by its name and the list of parameters. The **`coord`** section defines the mapping of individual abstract processors performing the algorithm onto the grid layout using variables `I` and `J`. The **`node`** primitive defines the amount of computations that will be performed by each processor, which depends on its spatial coordinates in the grid as indicated by `I` and `J` and the computing power of the individual processors as indicated by `partition_size`, which is controlled by a `benchmark` function. Finally, the **`parent`** directive simply indicates the spatial localization of the master processor.

An important consideration in the performance model `amc_perf` described above is the nature of the `benchmark` function used as a baseline for the model definition. On the one hand, this function should be truly representative of the underlying application. On the other hand, the computations involved in such a function should be small enough to give an accurate approximation of the processing power in a very short time (which of course depends on the particular application). In this work, we have adopted as benchmark the computation of the MEI index for a 3 × 3 SE (as described in Figure 2), which means that our benchmark function is step 2 of the AMC algorithm described in the previous section. The main reasons for this decision are as follows:

1. First and foremost, it should be noted that the proposed AMC algorithm is based on repeatedly computing step 2 (parameter I_{max} controls the total number of iterations) and then assigns a classification label based on the estimated MEI score to each hyperspectral image pixel. Therefore, the use of the core computations involved in step 2 are truly representative of the algorithm.
2. Secondly, we emphasize that the computation of the MEI index for a 3 × 3 SE prevents the inclusion into the performance model of border-handling routines such as those depicted in Figures 3 and 4, which are only implemented for certain pixels and thus are not fully representative of the algorithm's performance.

3. Thirdly, it should be noted that the full computation of a 3 × 3 structuring element prevents the inclusion into the performance model of optimization aspects, such as the possible presence in cache memory of pixels belonging to a certain SE neighborhood – centered, say, around a hyperspectral image pixel $\boldsymbol{f}(x, y)$ – and which would also be present in the SE neighborhoods centered around pixels which are spatially adjacent to $\boldsymbol{f}(x, y)$ following eight-neighbor connectivity (Plaza et al. 2006).
4. Finally, in order to properly model memory considerations associated to hyperspectral imaging applications, we assume in the computation of the benchmark function that the amount of data allocated to a single processor in the cluster is a full AVIRIS hyperspectral cube with 614 × 512 pixels. The amount of data produced by the instrument in each pass is fixed (to 614 × 512 pixels with 224 spectral bands, each stored using 12 bits). Since AVIRIS is the most advanced instrument of its kind, we have adopted it as a highly representative case study for the definition of the benchmark function. Therefore, our function assumes an unfavorable scenario in which each processor is forced to make use of reallocation/paging mechanisms due to cache misses. This approach allows us to realistically model the relative speed of heterogeneous processors by simply running a standardized core computation in hyperspectral image processing.

With the above considerations in mind, the performance model introduced for AMC can be regarded as generic since it can be used to model any hyperspectral image processing algorithm which makes use of a sliding-window approach to perform local computations in each pixel's neighborhood. In Section 4.2 we describe the communication pattern adopted for this type of algorithm.

4.2 Communication Framework

Once a heterogeneous set of data partitions has been obtained using the performance model described in the previous subsection, a communication framework among heterogeneous processors needs to be established. Figure 5 provides a graphical description of the communication framework adopted in our proposed application. As Figure 5(a), shows, the processors are arranged in a virtual grid which, in the considered example, comprises 16 heterogeneous processors allocated into a 4 × 4 processor grid. The communication framework depicted in Figure 5 can be summarized by the following cases:

1. Processors located at the leftmost column of the grid: these first send all their overlap borders to processors at the column located immediately to the right. Then, these processors wait for the overlap borders that will be provided by processors at the column immediately to the right (see Figure 5b).
2. Processors located at an intermediate column of the grid: these first send all their overlap borders to processors at the columns located immediately to the left and to the right. Then, these processors wait for the overlap borders that will be provided by processors at the columns located immediately to the left and to the right (see Figure 5c).
3. Processors located at the rightmost column of the grid: these first wait for the overlap borders that will be provided by processors at the column immediately to the left. Then, these processors send all their overlap borders to processors at the column located immediately to the left (see Figure 5d).

The heterogeneity in communication patterns addressed above has been adopted on purpose in order to evaluate the best possible pattern to reduce communication times. From Figure 5, it can be seen that an alternative communication pattern may consist of having processors located at even columns first send the overlap borders to processors located at odd columns and then wait for the overlap borders provided by processors at such columns, while processors located at odd columns could first wait for the overlap borders provided by processors located at even columns and then send the overlap borders to processors located at even columns. Specifically, we have adopted the proposed communication pattern above bearing in mind the heterogeneous nature of the underlying hardware platform and the algorithm itself. The idea is that each processor communicates its part of the overlap border to the processors located at neighboring columns. There are several reasons for the above decision:

- Firstly, the proposed communication framework simplifies the parallel algorithm design and alleviates the need to impose strong conditionals in the main loop of the parallel code in order to estimate the size of each particular communication for each particular processor. For instance, processor P_9 in Figure 5(a) would have to send its leftmost overlap border in chunks of different sizes to processors P_4, P_5 and P_6, respectively, and all these communications should be based on the processing power of each particular processor.
- Secondly, the proposed communication framework simplifies and enhances dynamic reconfiguration during execution, e.g. by updating the processing power of each processor using the HeteroMPI_Recon operation. This way, the amount of data to be computed and/or commu-

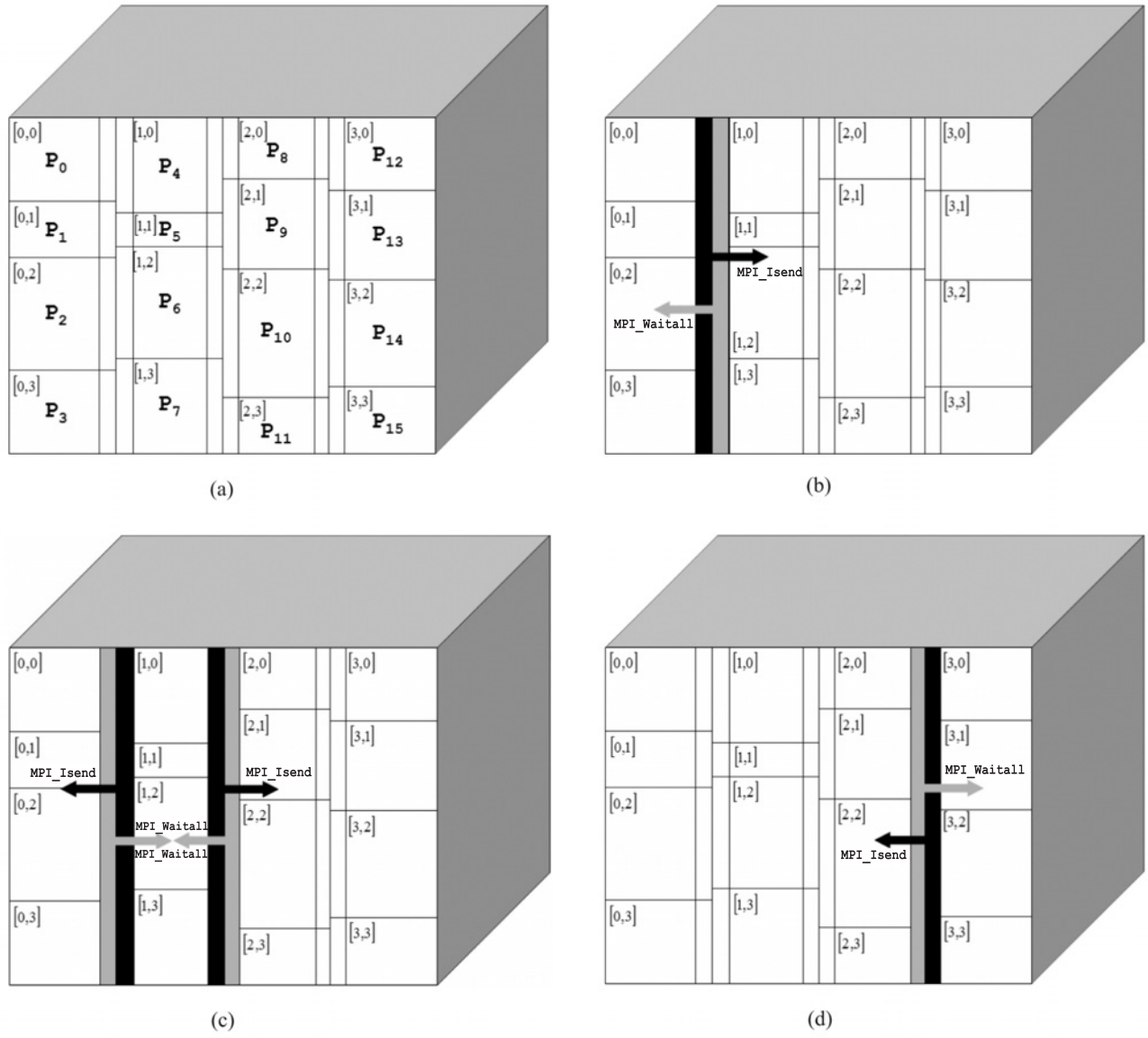

Fig. 5 Communication framework for the proposed AMC algorithm. (a) Assignment of data partitions to a set of heterogeneous processors arranged in a 4 × 4 virtual processor grid; (b) Processors in leftmost column (column 0) first send their overlap borders to processors in column 1 and then wait for the overlap borders of processors in that column; (c) Processors in middle column (column 1) first send their overlap borders to processors in columns 0 and 2 and then wait for the overlap borders of processors in those columns; (d) Processors in rightmost column (column 3) first wait for the overlap borders of processors in column 2 and then send their overlap borders to processors in that column.

nicated at each heterogeneous partition can be dynamically adjusted.

- Finally, although other alternatives are indeed possible, the proposed approach favors the balance of communications and computations.

4.3 HeteroMPI-Based Implementation

Once a performance model for the parallel algorithm has been defined, implementation using the standard HeteroMPI in Section 2 is quite straightforward (Valencia et al. 2006), as shown by the `main` program below which

```
main(int argc, char *argv[]){
HeteroMPI_Init(&argc,&argv);
if (HeteroMPI_Is_member(HMPI_COMM_WORLD_GROUP)){
     HeteroMPI_Recon(benchmark, dims, 15, &output);
}
HeteroMPI_Group_create(&gid, &MPC_NetType_amc_perf, modelp, num_param);
if (HeteroMPI_Is_free()){
     HeteroMPI_Group_create(&gid, &MPC_NetType_hpamc_rend, NULL, 0);
}
 if (HeteroMPI_Is_free()){
      HeteroMPI_Finalize(0);
 }
 if (HeteroMPI_Is_member(&gid)){
     Tinit = MPI_Wtime();
     Communicator = *(MPI_Comm *)HMPI_Get_comm(&gid);
     if (&Communicator == NULL){
          HeteroMPI_Finalize(0);
     }
     if (HeteroMPI_Group_coordof(&gid,&dim,&coord) == HMPI_SUCCESS){
          HeteroMPI_Group_performances(&gid, speeds);
          Read_image(name,image,lin,col,bands,data_type,init);
          for (i=imax; i>1; i=i--){
               AMC_algorithm(image,lin,col,bands,sizeofB,res);
               if (coord[0] == 0) { //First column in the virtual grid
               //MPI_Isend to send border to rightmost col
               //MPI_Waitall to receive border from rightmost col
               } else {
               if (coord[0] == p-1) { //Last col in the grid
               //MPI_Waitall to receive border from left col
               //MPI_Isend to send border to left col
               } else { //Any other case
               //MPI_Isend to send border to left col
               //MPI_Isend to send border to right col
               //MPI_Waitall to receive border from left col
               //MPI_Waitall to receive border from right col}
          }
          if (HeteroMPI_Is_member(&gid)){
               free(image);}
          HeteroMPI_Group_free(&gid);
          HeteroMPI_Finalize(0);}
     }
}
```

represents the most interesting fragment of the HeteroMPI-based code of our parallel implementation.

As shown by the piece of code above, the HeteroMPI runtime system is initialized using operation **HeteroMPI_Init**. Then, operation **HeteroMPI_Recon** updates the estimation of performances of processors. This is followed by the creation of a group of processes using operation **HeteroMPI_Group_create**. The members of this group then execute the parallel algorithm. At this point, control is handed over to MPI. HeteroMPI and MPI are interconnected by the operation **HeteroMPI_Get_comm**, which returns an MPI communicator with communication group of MPI processes dened by `gid`. This is a local operation not requiring inter-process communication. The communicator is used to call standard MPI communication routines such as **MPI_Isend** and **MPI_Waitall**, following the communication pattern described in Figure 5. This is followed by freeing the group using operation **HeteroMPI_Group_free**, and the finalization of the HeteroMPI runtime system using operation **HeteroMPI_Finalize**.

To conclude this section, we emphasize that HeteroMPI allows us to map our parallel application on a heterogeneous environment in accordance with the computational

resources available from every single node. As a result, the amount of work in the AMC algorithm is distributed unequally among heterogeneous processors to balance load. In other words, a HeteroMPI application is like any other MPI application and can be deployed to run in any environment where MPI applications are used (HeteroMPI applications can be run in environments where batch queuing and resource management systems are used). However, HeteroMPI uses its own measurements and performance models of the underlying system for running parallel applications efficiently. In this regard, it is important to note that the `benchmark` function used to measure the processing power of the processors in **`HeteroMPI_Recon`** is essential, mainly because a poor estimation of the power and memory capacity of processors may result in load balancing problems (Plaza, Plaza, and Valencia 2007). The effectiveness of our HeteroMPI-based implementation is addressed via experiments in the following section.

5 Experimental results

5.1 Parallel Computing Architectures

Two types of parallel computers have been used in this work for experimental assessment: heterogeneous and homogeneous. Table 1 shows the specifications of processors in a heterogeneous cluster composed of 11 Linux/SunOS workstations (15 processors) at the Heterogeneous Computing Laboratory (HCL), University College Dublin (UCD). From now on, we will refer to this platform as HCL-1. The processors in Table 1 are interconnected via 100 Mbit Ethernet communication network with a switch enabling parallel communications among the processors. Although this is a simple configuration, it is also a quite typical and realistic one as well. For illustrative purposes, Table 1 also reports the relative speeds of the heterogeneous processors in the cluster.

Another heterogeneous cluster designated by HCL-2 was also used in experiments (see Table 2). It is made up of 16 nodes from Dell, IBM, and HP, with Celeron, Pentium 4, Xeon, and AMD processors ranging in speeds 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 and the bandwidth of each port can be configured to meet any value between 8 Kb/s and 1 Gb/s (see http://hcl.ucd.ie/Hardware for additional details). Table 2 also reports the relative speed of each processor measured with the benchmark function (which took only 0.036 seconds to be executed in all cases).

Finally, in order to test the scalability of the proposed parallel algorithm on a larger-scale parallel platform, we have also experimented with Thunderhead, a 568-processor Beowulf cluster located at NASA's Goddard Space Flight Center (see http://thunderhead.gsfc.nasa.gov for additional details). Thunderhead can be seen as an evolution of the HIVE (Highly Parallel Virtual Environment) project (Dorband et al. 2003), started in 1997 to build a homogeneous commodity cluster to be used in a wide range of scientific applications. The system is composed of 268 dual 2.4 GHz Intel Xeon nodes, each with 1 GB of main memory and 80 GB of disk space. The theoretical peak performance of the system is 2.5728 Tflops.

5.2 Hyperspectral Image Data

Figure 6(a) shows the Indian Pines AVIRIS hyperspectral data set considered in experiments. The scene was col-

Table 1
Specifications of heterogeneous processors in HCL-1 heterogeneous cluster.

Processor number	Name (Processors)	Architecture description	CPU (MHz)	Memory (MB)	Cache (KB)	Relative speed
0,1 2,3 4,5 6,7	Pg1cluster01(2) Pg1cluster02(2) Pg1cluster03(2) Pg1cluster04(2)	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1024	512	70
8 9 10 11 12 13 14	csultra01(1) csultra02(1) csultra03(1) csultra05(1) csultra06(1) csultra07(1) csultra08(1)	SunOS 5.8 sun4u sparc SUNW, Ultra-5_10	440	512	2048	30

Table 2
Specifications of heterogeneous processors in HCL-2 heterogeneous cluster.

Proc. no.	Model description	Processor description	Operating system	CPU (GHz)	Mem (MB)	Cache (KB)	HDD 1	HDD 2	Rel. speed
0,1	Dell Poweredge SC1425	Intel Xeon	Fedora Core 4	3.6	256	2048	240 GB SCSI	80 GB SCSI	7.93
2–7	Dell Poweredge 750	Intel Xeon	Fedora Core 4	3.4	1024	1024	80 GB SATA	N/A	7.20
8	IBM E-server 326	AMD Opteron	Debian	1.8	1024	1024	80 GB SATA	N/A	2.75
9	IBM E-server 326	AMD Opteron	Fedora Core 4	1.8	1024	1024	80 GB SATA	N/A	2.75
10	IBM X-Series 306	Intel Pentium 4	Debian	3.2	512	1024	80 GB SATA	N/A	6.13
11	HP Proliant DL 320 G3	Intel Pentium 4	Fedora Core 4	3.4	512	1024	80 GB SATA	N/A	6.93
12	HP Proliant DL 320 G3	Intel Celeron	Fedora Core 4	2.9	1024	256	80 GB SATA	N/A	3.40
13	HP Proliant DL 140 G2	Intel Xeon	Debian	3.4	1024	1024	80 GB SATA	N/A	7.73
14	HP Proliant DL 140 G2	Intel Xeon	Debian	2.8	1024	1024	80 GB SATA	N/A	3.26
15	HP Proliant DL 140 G2	Intel Xeon	Debian	3.6	1024	2048	80 GB SATA	N/A	8.60

lected by the AVIRIS sensor, and is characterized by very high spectral resolution (224 narrow spectral bands in the range 0.4–2.5 μm) and moderate spatial resolution (614 samples, 512 lines and 20 m pixels). It was gathered over the Indian Pines test site in Northwestern Indiana, a mixed agricultural/forested area, early in the growing season. As shown by Figure 6(a), the data set represents a very challenging classification problem. The primary crops of the area, mainly corn and soybeans, were very early in their growth cycle with only about 5% canopy cover. Discriminating between the major crops under these circumstances can be very difficult, a fact that has made this scene a universal and extensively used benchmark to validate classification accuracy of hyperspectral imaging algorithms. Fortunately, extensive ground-truth (reference) information is available for the area. Figure 6(b) shows a ground-truth map, given in the form of a class assignment for each labeled pixel with 30 mutually exclusive ground-truth classes.

5.3 Assessment of the Parallel Algorithm

This section develops an extensive evaluation of the proposed heterogeneous parallel AMC algorithm from the viewpoint of classification accuracy and load balance, using the two heterogeneous clusters at HCL/UCD. The section concludes by analyzing the scalability of the algorithm on the Thunderhead Beowulf cluster at NASA.

5.3.1 Study of classification accuracy The parallel AMC algorithm was applied to the AVIRIS Indian Pines scene in Figure 6 using a fixed, 3 × 3-pixel SE and seven different values for parameter I_{max}, which defines the number of iterations executed by the algorithm (ranging from 1 to 7 in experiments). Table 3 shows the classification accuracies (in percentage of correctly classified pixels) obtained using the seven considered numbers of iterations, along with the single-processor execution times (in minutes) measured in a Linux workstation with Intel Xeon processor at 2 GHz, 1 GB of RAM and 512 KB of cache.

As shown by Table 3, the AMC algorithm was able to achieve very high classification accuracies, especially for $I_{max} = 7$ (above 90%), but the measured processing times were extremely high and generally unacceptable in remote sensing applications in which a response in (near) real-time is often required.

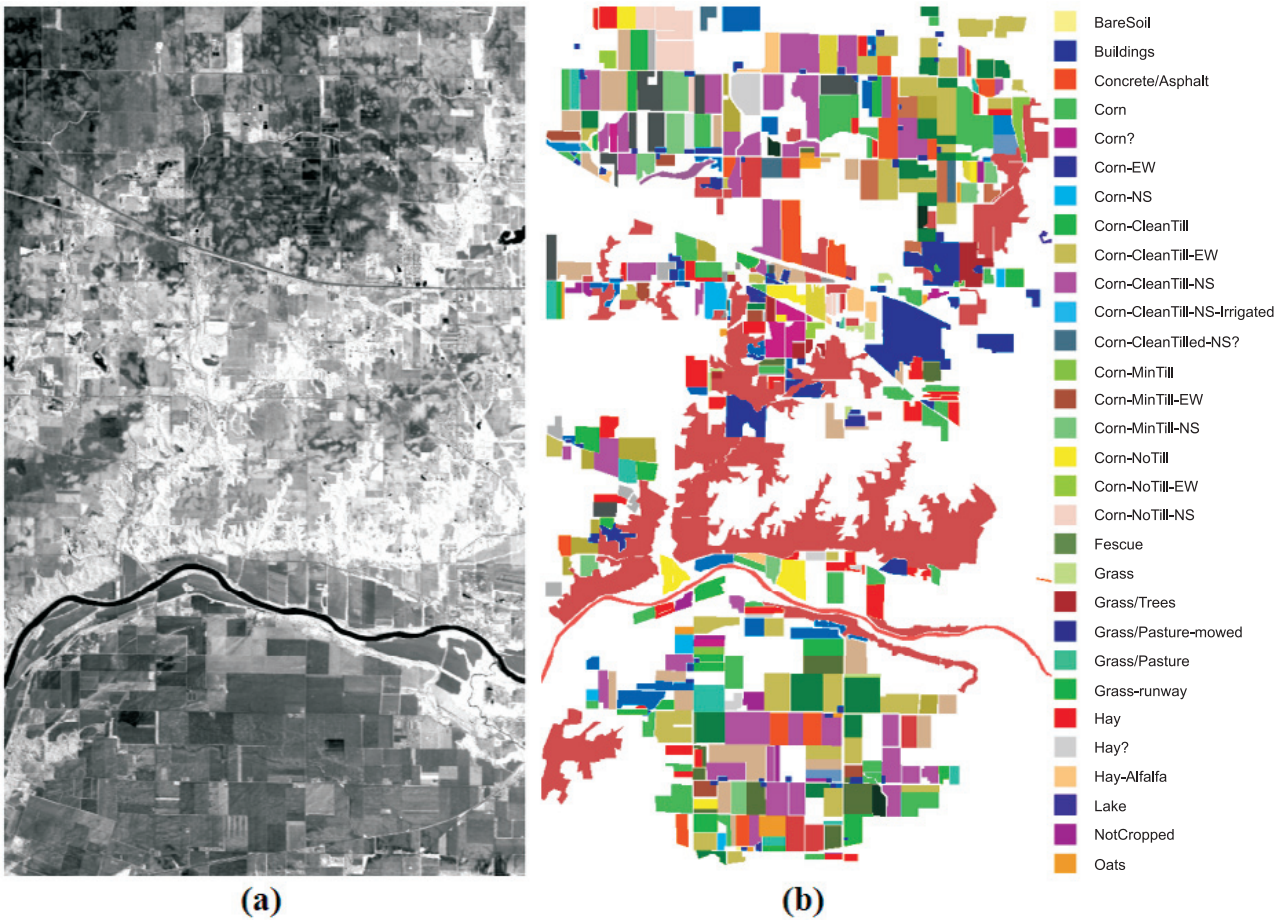

Fig. 6 (a) Spectral band at 587 nm wavelength of an AVIRIS scene comprising agricultural and forest features at Indian Pines, Indiana. (b) Ground-truth map with 30 mutually exclusive classes.

Table 3
Classification accuracies and single-processor times for the AMC algorithm.

Iterations	1	2	3	4	5	6	7
Accuracy (%)	75.23	78.43	81.94	83.99	87.95	88.79	90.02
Time (min)	9.54	19.56	27.82	37.06	46.91	54.68	64.79

5.3.2 Performance evaluation on the HCL-1 heterogeneous cluster. To investigate the parallel properties of the considered HeteroMPI-based algorithm, it was first implemented on the HCL-1 cluster at UCD (see Table 1). Before reporting the timing results, we emphasize that the relative speeds of the heterogeneous processors were first estimated for different problem sizes (i.e. number of iterations ranging from $I_{max} = 1$ to $I_{max} = 7$) by incorporating the core computations of the morphological algorithm (as defined by our adopted `benchmark` function) to the `amc_perf` performance model. In order for such estimation to be accurate, it was necessary to include memory management considerations in the benchmark function to avoid disregarding important aspects such as virtual memory paging and cache considerations. As mentioned above, in our particular implementation, we used an

Table 4
Execution times (in seconds) of the HeteroMPI-based algorithm in each of the heterogeneous processors of HCL-1 for different numbers of iterations.

Iterations	1	2	3	4	5	6	7
0	46.86	91.25	140.69	186.46	226.06	285.51	337.49
1	47.05	90.74	141.49	183.66	228.06	288.77	328.88
2	47.32	92.15	138.23	187.38	227.75	287.96	325.31
3	47.09	92.96	134.46	180.55	226.68	274.10	317.73
4	50.01	95.57	149.55	199.20	237.06	300.94	340.53
5	50.59	94.95	148.70	197.76	235.17	309.22	345.14
6	48.32	99.48	139.15	188.48	246.55	291.75	329.67
7	48.26	91.82	143.86	191.09	246.61	294.96	333.94
8	48.90	101.28	141.44	188.25	250.61	290.83	322.06
9	50.48	98.63	152.04	200.33	238.35	304.19	358.36
10	51.07	98.48	154.39	197.50	238.12	308.83	358.06
11	46.43	92.69	139.80	180.44	227.03	274.77	321.50
12	47.12	93.24	141.40	183.85	229.87	282.43	328.16
13	46.54	92.35	137.60	184.44	231.65	288.52	315.20
14	46.85	94.47	137.70	186.32	235.26	288.67	326.25

Table 5
Load balancing rates for the HeteroMPI-based algorithm executed on HCL-1 with different numbers of iterations.

Iterations	1	2	3	4	5	6	7
R_{max}	46.43	90.74	134.46	180.44	226.06	309.22	358.36
R_{min}	51.07	101.28	154.39	200.33	250.61	274.10	315.20
D	1.09	1.11	1.14	1.11	1.10	1.12	1.13

approach which assumes that each heterogeneous processor has memory capacity sufficient to work with the entire hyperspectral data set locally. Based on previous work (Plaza et al. 2006), this is a reasonable assumption in most hyperspectral imaging scenarios. Further, this provides us with a means to effectively model memory hierarchy-related parameters by simulating a largely unfavorable scenario in which each processor is forced to make use of reallocation/paging mechanisms due to cache misses.

With the above assumptions in mind, Table 4 shows the execution times (in seconds) of the HeteroMPI-based parallel morphological algorithm in each of the processors of the heterogeneous cluster. As shown by Table 4, the heterogeneous algorithm was able to adapt efficiently to the heterogeneous computing environment where it was run. In particular, one can see that the heterogeneous algorithm executed on HCL-1 was always about eleven times faster than the equivalent sequential algorithm executed on a Linux workstation which is almost identical to the **csultra** nodes in the considered heterogeneous cluster (see Table 1). Most importantly, we experimentally tested that the mean processing times in the eight Pg1cluster processors were almost identical to the mean processing times in the seven **csultra** nodes (for all considered problem sizes). This fact reveals that the slight differences in the execution times reported in Table 4 are due to the intrinsic characteristics of the parallel problem, and not to platform heterogeneity which is accurately modeled by HeteroMPI.

In order to measure load balance, Table 5 shows the imbalance scores achieved by the parallel heterogeneous algorithm on the considered HNOC. The imbalance is defined as $D = R_{max}/R_{min}$, where R_{max} and R_{min} are the maxima and minima processor run times, respectively.

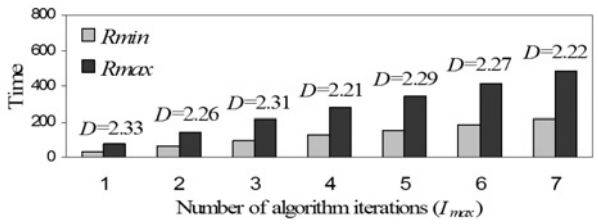

Fig. 7 Load-balancing rates for a parallel version of the algorithm executed on HCL-1 without memory considerations.

Therefore, perfect balance is achieved when $D = 1$. The load balancing rates in Table 5 are superior to those reported in (Plaza et al. 2006) for standard, spectral-based hyperspectral analysis algorithms executed in homogeneous computing platforms.

Before concluding this subsection, we would like to emphasize the importance of incorporating considerations about memory capacity of the different nodes in the benchmark function used to define the performance model `amc_perf`. For illustrative purposes, Figure 7 shows the values of R_{max}, R_{min} and D obtained on the considered HNOC for a parallel version of the proposed algorithm in which the benchmark function only modeled the processing power of heterogeneous processors and did not take into account memory-related parameters. The imbalance scores are also reported for completeness. Overall, Figure 7 shows that disregarding memory considerations in the HeteroMPI performance model results in higher imbalance scores.

5.3.3 Performance evaluation on the HCL-2 heterogeneous cluster. The proposed parallel heterogeneous algorithm has also been implemented in the HCL-2 heterogeneous cluster, which provides a more heterogeneous environment than HCL-1 for experiments. Table 6 reports the execution times (including both computations and communications) measured at each heterogeneous processor of the HCL-2 cluster after running the HeteroMPI-based version of the AMC algorithm with $I_{max} = 1$, and considering an increasing number of spectral bands for the 350 × 350-pixel scene (ranging from 20 bands to the maximum number of available bands in the scene,

Table 6
Processing times (in seconds) measured at each processor of the HCL-2 cluster for an execution of the parallel AMC algorithm with I_{max} = 1 and different numbers of spectral bands in the considered AVIRIS hyperspectral scene.

	Number of spectral bands in the hyperspectral image								
Processor	20	40	60	80	100	120	140	160	180
0	3.41	5.03	6.93	9.69	11.49	14.02	16.73	19.09	21.45
1	3.46	5.32	7.05	9.16	13.33	14.02	16.74	19.09	21.46
2	3.25	5.03	6.92	9.12	13.30	13.98	16.67	19.03	21.39
3	3.27	5.10	7.17	9.34	13.34	14.03	16.74	19.11	21.47
4	3.45	5.27	7.14	9.67	13.29	13.98	16.69	19.04	21.41
5	3.45	5.31	7.16	9.69	13.32	14.01	16.72	19.10	21.46
6	3.44	5.28	7.15	9.67	13.31	13.99	16.70	19.05	21.41
7	3.46	5.32	7.17	9.70	13.34	14.03	16.74	19.11	21.47
8	3.26	5.02	6.91	9.14	13.32	13.99	16.72	19.08	21.42
9	3.24	5.01	6.91	9.12	13.29	13.97	16.67	19.02	21.39
10	3.26	5.04	6.93	9.13	13.31	14.00	16.70	19.07	21.44
11	3.24	4.98	6.90	9.10	13.28	13.95	16.65	19.00	21.37
12	2.08	2.48	2.81	3.24	3.67	12.72	16.71	19.06	21.40
13	2.09	2.64	2.99	3.35	3.67	12.71	16.68	19.04	21.42
14	2.10	2.47	2.82	3.26	3.68	12.72	16.70	19.07	21.44
15	2.09	2.47	2.81	3.24	3.66	12.71	16.68	19.05	21.40

i.e., 180). In this example, we have considered the minimum possible number of algorithm iterations in order to reduce the ratio of computations to communications as much as possible, and thus be able to evaluate the impact of the communication pattern adopted for the algorithm.

As shown by Table 6, the execution times reported for processors P_{12}, P_{13}, P_{14} and P_{15} are lower than those measured for processors P_0 to P_{11} when the number of spectral bands is 120 or below. This is partly because these processors, located at the rightmost column of a 4 × 4 virtual processor grid, exhibit a different communication pattern (as reported in Figure 5), i.e. these processors first wait for the overlap borders to be provided by processors at the column immediately to the left and then send their overlap borders, as opposed to processors P_0 to P_{11} which first send their overlap borders and then wait for the overlap borders to be provided by neighboring processors. Since at some point their neighbors will also be sending (at the same time), processors P_{12}, P_{13}, P_{14} and P_{15} cannot start receiving until one or several of them have finished sending their messages. Thus, a sequential pattern is introduced in communications leading to an overall increase in the measured time as shown in Table 6.

Interestingly, when the number of bands is 120 or higher, the execution times reported for processors P_{12}, P_{13}, P_{14} and P_{15} increase significantly and a linearity of the execution times is observed. In order to interpret this phenomenon, we first roughly estimate the average size of messages for our 350 × 350-pixel scene with 120 spectral bands as (350/4 overlap border pixels × 120 bands × 16 bits per value = 21,000 bytes) – the number of pixels in a column of the image is roughly divided by 4 to reflect our arrangement of processors into a 4 × 4 virtual grid. As shown by Lastovetsky and O'Flynn (2007), the probability of congestion measured for the HCL-2 cluster for a message size of 21 KB is 100%, which explains the escalation observed for the processing times measured for processors P_{12}, P_{13}, P_{14} and P_{15} in Table 6 when the number of spectral bands goes from 100 to 120, as well as the linear behavior observed for all processors in HCL-2 when the number of bands is above 120.

It should also be noted that the processing of hyperspectral images using the AMC algorithm and $I_{max} = 1$ for only 120 out of 180 spectral bands represents a moderately interesting case study in hyperspectral imaging since the AMC algorithm has been shown to provide better classification results as the number of algorithm iterations is increased (see Table 3). In real applications, it is often desirable to use the full spectral information available in the hyperspectral data in order to be able to separate the classes more effectively, thus improving the final classification results. In this case, the average size of messages for our 350 × 350 pixel scene with 180 spectral bands can be roughly estimated as (350/4 overlap border pixels × 180 bands × 16 bits per value = 31,500 bytes). Therefore, we are aware that experimental results reported in this paper can be further improved in future developments since the average size of messages used to communicate the overlap borders in our application is located in the congestion region described by Lastovetsky and O'Flynn (2007) and also in the range of message sizes with high probability of congestion. However, we believe that our results are encouraging since the achieved load balance (from the viewpoint of both communications and computations) is very good, in particular, for a large number of spectral bands.

In order to further substantiate the above remark, we need to analyze the performance of the proposed parallel AMC algorithm using the full spectral information available and different numbers of iterations. For that purpose, Table 7 reports the processing times (measured in seconds) at each processor of the HCL-2 cluster using $1 < I_{max} \leq 7$ iterations and all available spectral bands (180) in the considered AVIRIS Indian Pines scene. As shown by Table 7, the processing times reported for the 16 processors are well balanced in all cases, as was also observed in experiments in the HCL-1 cluster (see Tables 4 and 5). It should be noted that both HCL-1 and HCL-2 are composed of a limited number of processors and, hence, experiments on parallel platforms with a higher number of processing units are highly desirable.

5.3.4 Scalability analysis on the Thunderhead Beowulf cluster. To analyze scalability issues in a larger computing platform, we provide parallel performance results of a homogeneous version of the proposed AMC algorithm using NASA's Thunderhead Beowulf cluster as the baseline computing architecture (Plaza et al. 2006). Although results in this subsection are not indicative of the efficiency of the proposed heterogeneous parallel implementation, they provide a preliminary assessment of the scalability of a homogeneous parallel version of the AMC algorithm in a fully homogeneous cluster. It should be noted that the homogeneous algorithm was derived using a standard MPI implementation (Plaza et al. 2006).

Figure 8 plots the speedups achieved by AMC (using different values of I_{max}) as a function of the number of processors on Thunderhead. For illustrative purposes, the performance of two additional parallel hyperspectral algorithms, S-PCT (Achalakul and Taylor 2003) and D-ISODATA (Dhodhi et al. 1999), is also reported. Results in Figure 8 reveal that the performance drop from linear speedup in both S-PCT and D-ISODATA algorithms increases significantly as the number of processors increase. A similar effect is also observed for the proposed AMC parallel algorithm. This comes as no surprise since we already expected that our application would encounter performance problems in large-scale platforms when

Table 7
Processing times (in seconds) measured at each processor of the HCL-2 cluster for an execution of the parallel AMC algorithm with different numbers of iterations (I_{max}) and the full spectral information available (180 spectral bands) in the considered AVIRIS hyperspectral scene.

	Number of iterations					
Processor	2	3	4	5	6	7
0	42.53	63.98	84.84	107.74	130.67	145.63
1	42.59	63.88	84.80	107.68	130.71	145.64
2	42.58	63.83	84.99	107.67	130.65	145.65
3	42.56	63.77	84.74	106.42	130.72	145.64
4	42.56	62.86	84.80	107.68	130.72	145.56
5	42.49	63.84	84.85	107.74	130.59	145.58
6	42.61	63.81	84.77	107.73	130.66	144.39
7	42.60	63.97	84.95	107.74	130.67	145.56
8	42.54	63.81	83.88	107.67	130.65	145.60
9	42.52	63.82	84.79	107.70	128.88	145.52
10	42.60	63.80	84.78	107.69	130.71	145.63
11	42.53	63.84	84.84	107.71	130.64	145.61
12	42.61	63.80	84.77	107.66	130.64	145.59
13	42.52	63.88	84.77	107.69	130.63	145.59
14	42.59	63.83	84.78	107.63	130.66	145.58
15	42.59	63.88	84.95	107.73	130.70	145.58

the number of processors is increased, a situation in which the partition sizes decrease significantly. If for a small number of processors the partition size fits into the area of large messages, then an increase in the number of processors will quickly move it into the area of medium-sized messages, i.e. $M_1 < M \leq M_2$. It should be noted that M_1 is a threshold below which the execution time is always linearly proportional to the message size, while M_2 is a threshold above which a return to a deterministic linear behavior is observed in the execution time after a non-linear and non-deterministic behavior in between the two thresholds (Lastovetsky and O'Flynn 2007). This may result in a significant increase in the execution times of the single `MPI_Gather` operation used to develop our MPI-based parallel code for the homogeneous platform. Although fine-tuning of the proposed MPI-based parallel algorithm for efficient performance in large-scale commodity clusters is out of the scope of this paper, we can address this issue by redesigning the parallel algorithm as follows. If we replace the single `MPI_Gather` gathering 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 would be communicated as a sequence of small-sized sub-partitions. In order to accomplish the above-mentioned goal, we should properly calculate the number of sub-partitions m of a partition of the medium size M so that $(M/m) < M_1$ and $M/(m-1) > M_1$.

On the other hand, Figure 8 also reveals that, although the proposed AMC algorithm introduces inter-processor communications expected to slow down the computation a priori, it is important to note that the measured speed-ups tend to be higher for large values of I_{max}, a fact that reveals that the proposed scheme scales better as the size of the problem increases. As a result, we can conclude that the dominant issue in the proposed morphological algorithm is problem size, which makes the algorithm very appealing for high-dimensional imaging applications. Although for a high number of nodes the speedup graphs flatten out a little, due to the issues discussed above, they clearly outperform those obtained for the other tested parallel algorithms.

For comparative purposes, Table 8 reports the algorithm processing times of the parallel homogeneous algorithms on Thunderhead. It can be seen in the table that the single-processor implementations of both S-PCT and D-ISODATA require much more computation time than

Table 8
Execution times (seconds) for parallel algorithms using different numbers of processors on Thunderhead.

No. of CPUs	S-PCT	D-ISODATA	AMC (I_{max} = 1)	AMC (I_{max} = 3)	AMC (I_{max} = 5)	AMC (I_{max} = 7)
1	41,239	49,912	1523	3265	4876	7028
4	13,521	21,330	562	1107	1563	2176
16	4314	5907	139	290	369	494
36	1759	2428	55	112	156	210
64	884	1299	29	60	86	121
100	572	865	19	39	56	79
144	392	630	13	27	39	56
196	314	444	10	21	30	42
256	265	386	6	16	22	29

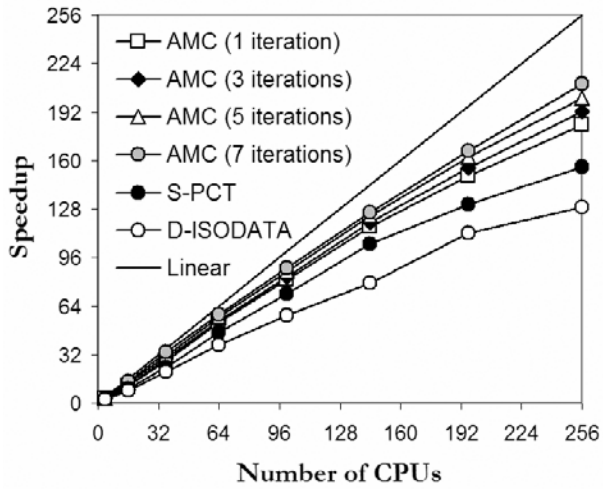

Fig. 8 Scalability of the proposed AMC and other parallel hyperspectral algorithms on NASA's Thunderhead cluster.

any of the four variations of AMC tested in experiments. This comes as no surprise, since AMC is a windowing type algorithm as opposed to the other tested algorithms which involve internal synchronization steps and additional communications. Interestingly, the table also reveals that the utilization of a high number of processors on Thunderhead allows near real-time processing of the considered AVIRIS scene. Since the cross-track scan line in AVIRIS is quite fast (8.3 ms), a full image cube frame with 512 × 614 pixels would have to be processed in about 5 s in order to fully achieve real-time. Although we closely approach this figure using the AMC algorithm with I_{max} = 1, further work is required in order to achieve similar results with a higher number of algorithm iterations, thus leading to improved classification results.

Finally, Table 9 reports the load-balancing scores for the best case in the sense of higher speedups (I_{max} = 7) of our MPI-based implementation of the AMC algorithm in the homogeneous cluster. As can be seen after comparing results in Tables 5 and 7 with those in Table 9, the load-balancing results obtained in the homogeneous platform are not better than those reported for the two considered heterogeneous platforms, revealing that our HeteroMPI-based implementation competes (in terms of load balance and parallel efficiency) with the equivalent homogeneous implementation of the same algorithm. For illustrative purposes, the load-balancing scores achieved by both S-PCT and D-ISODATA are also reported in Table 9.

Table 9
Load-balancing scores for the MPI-based S-PCT, D-ISODATA and AMC (I_{max} = 7) on Thunderhead.

No. of CPUs	4	16	36	64	100	144	196	256
S-PCT	1.08	1.05	1.05	1.04	1.03	1.04	1.04	1.03
D-ISODATA	1.13	1.08	1.10	1.04	1.07	1.05	1.07	1.06
AMC	1.04	1.02	1.04	1.03	1.01	1.02	1.03	1.01

Summarizing, experimental results in our study reveal that heterogeneous algorithms offer an attractive and efficient solution to the problem of extracting useful information and knowledge from hyperspectral image data set in distributed fashion. Contrary to common perception that spatial/spectral information extraction algorithms are too computationally demanding for practical use, our results demonstrate that such combined approaches may indeed be very appealing for parallel design and implementation, not only because of the window-based nature of such algorithms, but also because their communication patterns can be effectively described via robust communication models. Our experimental results also revealed several important algorithmic aspects that may be of great importance for adapting existing hyperspectral imaging techniques to heterogeneous platforms, which currently represent a cost-effective parallel computing platform for many scientific and engineering applications. We also feel that the applicability of the proposed parallel heterogeneous method may extend beyond the domain of hyperspectral image analysis. This is particularly true for the domains of signal processing and linear algebra applications, which include similar patterns of communication and calculation.

6 Conclusions and Future Lines of Research

The aim of this paper has been the examination of parallel strategies for hyperspectral analysis on heterogeneous 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 of the effects that platform heterogeneity has on degrading parallel performance of hyperspectral analysis algorithms. To achieve the above goals, we have resorted to HeteroMPI (a recently proposed extension of MPI for programming high-performance computations on heterogeneous platforms) to develop a heterogeneous version of a spatial/spectral morphological classification algorithm, selected as a case study throughout the paper. Another contribution of the paper has been the consideration of a collective communication model in order to evaluate the communication framework adopted by the proposed parallel algorithm.

An interesting finding of experiments in this paper is that spatial/spectral heterogeneous approaches offer a surprisingly simple, yet effective and highly scalable solution for hyperspectral image classification. Despite the fact that conventional hyperspectral imaging algorithms do not take into account the spatial information explicitly in the computations (which has traditionally been perceived as an advantage for the development of parallel implementations), experimental results in this work suggest that the proposed HeteroMPI-based parallel algorithm is effective in terms of workload distribution, load-balancing rates, required inter-processor communications, and execution times. Our experimental results also revealed important algorithmic aspects that may be of great importance for designing and adapting existing high-performance hyperspectral imaging applications (mostly 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 latest-generation sensor and parallel processing technology may introduce substantial changes in the systems currently used by NASA and other agencies for exploiting the sheer volume of Earth and planetary remotely sensed data collected on a daily basis.

Our future work will be mainly directed towards the integration of the collective communication model described by Lastovetsky and O'Flynn (2007) with our proposed HeteroMPI-based code (i.e. using the model to redesign our parallel implementation). We will also pursue the implementation of parallel hyperspectral imaging algorithms on other massively parallel and distributed computing architectures, including grid computing environments. Finally, we are also working towards specialized hardware implementations of hyperspectral imaging algorithms on field programmable gate arrays (FPGAs) and graphics processing units (GPUs), which may allow us to fully accomplish the goal of real-time processing of hyperspectral imagery, with potential applications in onboard hyperspectral data compression and analysis.

Author Biographies

David Valencia is an assistant professor at the University of Extremadura, Spain, where he is currently pursuing his Ph.D. degree. His research interests are in the development of parallel implementations of algorithms for high-dimensional data analysis, with particular emphasis on commodity cluster-based (homogeneous and heterogeneous) systems and hardware-based architectures, including systolic arrays and FPGAs. He is also involved in the design and testing of large-scale distributed heterogeneous computing platforms.

Alexey Lastovetsky received his Ph.D. degree from the Moscow Aviation Institute in 1986, and his Doctor of Science degree from the Russian Academy of Sciences in 1997. His main research interests include algorithms, models and programming tools for high performance heterogeneous computing. He is the author of C[], a parallel

language for vector and superscalar processors, and mpC, the first parallel programming language for heterogeneous networks of computers. He designed HeteroMPI, an extension of MPI for heterogeneous parallel computing, and SmartNetSolve, an extension of NetSolve aimed at higher performance of scientific computing on global networks. He made contributions to heterogeneous data distribution algorithms and modeling the performance of processors in heterogeneous environments. He has published over 80 technical papers in refereed journals, edited books and international conferences. He authored the monograph *Parallel computing on heterogeneous networks* (Wiley, 2003). He is currently a senior lecturer in the School of Computer Science and Informatics at University College Dublin, National University of Ireland. At UCD, he also created and leads the Heterogeneous Computing Laboratory. He is an editor of the research journals *Parallel Computing* (Elsevier) and *Programming and Computer Software* (Springer).

Maureen O'Flynn is a Ph.D. candidate in the School of Computer Science and Informatics at University College Dublin, National University of Ireland. Her research interests include communication performance models for parallel computing on heterogeneous platforms.

Antonio Plaza received his Ph.D. degree in computer science from the University of Extremadura, Spain, in 2002, where he is currently an associate professor with the Computer Science Department. He has also been a visiting researcher with the University of Maryland, NASA Goddard Space Flight Center and Jet Propulsion Laboratory. His main research interests include the development and efficient implementation of high-dimensional data algorithms on parallel homogeneous and heterogeneous computing systems and hardware-based computer architectures such as FPGAs and GPUs. He has authored or coauthored more than 150 publications including journal papers, book chapters and peer-reviewed conference proceedings, and currently serves as regular manuscript reviewer for more than 15 highly cited journals in the areas of parallel and distributed computing, computer architectures, pattern recognition, image processing and remote sensing. He is editing a book on *High-Performance Computing in Remote Sensing* (with Prof. Chein-I Chang) for Chapman &Hall/CRC Press and a special issue on *Architectures and Techniques for Real-Time Processing of Remotely Sensed Images* for the Journal of Real-Time Image Processing. He is the coordinator of the *Hyperspectral Imaging Network*, an FP6 Marie Curie Research Training Network European project, and currently serves as Associate Editor for the IEEE Transactions on Geoscience and Remote Sensing journal in the areas of *Hyperspectral Image Analysis and Signal Processing*.

Javier Plaza received his Ph.D. degree in computer science from the University of Extremadura, Spain, in 2008, where he is currently an assistant professor. His current research work is focused on the development of efficient implementations of neural network-based algorithms for analysis and classification of hyperspectral scenes. He is also involved in the design and configuration of homogeneous and fully heterogeneous parallel computing architectures for high-performance scientific applications. Other major research interests include telecommunications, networking and configuration and training of neural network architectures for specific applications. He has been a reviewer for several highly cited journals in the areas of image processing, neural networks and remote sensing.

References

Achalakul, T. and Taylor, S. (2003). A distributed spectral-screening PCT algorithm, *Journal of Parallel and Distributed Computing* **63**(3): 373–384.

Brightwell, R., Fisk, L. A., Greenberg, D. S., Hudson, T., Levenhagen, M., Maccabe, A., and Riesen, R. (2000). Massively parallel computing using commodity components, *Parallel Computing* **26**(2–3): 243–266.

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

Dhodhi, M. K., Saghri, J. A., Ahmad, I., and Ul-Mustafa, R. (1999). D-ISODATA: A distributed algorithm for unsupervised classification of remotely sensed data on network of workstations, *Journal of Parallel and Distributed Computing* **59**(2): 280–301.

Dongarra, J., Huss-Lederman, S., Otto, S., Snir, M., and Walker, D. (1996). *MPI: The complete reference*, Cambridge, MA: MIT Press.

Dorband, J., Palencia, J., and Ranawake, U. (2003). Commodity computing clusters at Goddard Space Flight Center, *Journal of Space Communication* **1**(3), 23–35.

Kalluri, S., Zhang, Z., JaJa, J., Liang, S., and Townshend, J. (2001). Characterizing land surface anisotropy from AVHRR data at a global scale using high performance computing, *International Journal of Remote Sensing* **22**(11): 2171–2191.

Lastovetsky, A. (2002). Adaptive parallel computing on heterogeneous networks with mpC, *Parallel Computing* **28**(10): 1369–1407.

Lastovetsky, A. (2003). *Parallel computing on heterogeneous networks*, Hoboken, NJ: Wiley-Interscience.

Lastovetsky, A. and Reddy, R. (2006). HeteroMPI: Towards a message-passing library for heterogeneous networks of computers, *Journal of Parallel and Distributed Computing* **66**(2): 197–220.

Lastovetsky, A., Mkwawa, I., and O'Flynn, M. (2006). An accurate communication model for a heterogeneous cluster based on a switched-enabled Ethernet network. In: *Proceedings of the 12th International Conference on Parallel and Distributed Systems*, Vol. 2: 6 pp.

Lastovetsky, A. and O'Flynn, M. (2007). A performance model of many-to-one collective communications for parallel computing. In: *Proceedings of the 21st IEEE International parallel and Distributed processing symposium (IPDPS 2007)*, 26–30 March 2007, Long Beach, California, USA, CD-Rom/Abstract Proceeding, IEEE Computer Society.

Le Moigne, J., Campbell, W. J., and Cromp, R. F. (2002). An automated parallel image registration technique based on the correlation of wavelet features, *IEEE Transactions on Geoscience and Remote Sensing* **40**(8): 1849–1864.

Plaza, A. and Chang, C.-I (2007). *High-performance computing in remote sensing*, Boca Raton, FL: Chapman & Hall/CRC Press.

Plaza, A., Martinez, P., Plaza, J., and Perez, R. (2002). Spatial-spectral endmember extraction by multidimensional morphological operations, *IEEE Transactions on Geoscience and Remote Sensing* **40**(9): 2025–2041.

Plaza, A., Martinez, P., Plaza, J., and Perez, R. (2005). Dimensionality reduction and classification of hyperspectral image data using sequences of extended morphological transformations, *IEEE Transactions on Geoscience and Remote Sensing* **43**(3): 466–479.

Plaza, A., Valencia, D., Plaza, J., and Martinez, P. (2006). Commodity cluster-based parallel processing of hyperspectral imagery, *Journal of Parallel and Distributed Computing* **66**(3): 345–358.

Plaza, A., Plaza, J., and 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**(1): 81–107.

Richards, J. and Jia, X. (2005). *Remote sensing digital image analysis*, 4th edition, Berlin: Springer.

Soille, P. (2003). *Morphological image analysis: Principles and applications*, 2nd edition, Berlin: Springer.

Tilton, J. C. (2005). Method for implementation of recursive hierarchical segmentation on parallel computers, U.*S. Patent Office, Washington, DC, Pending Published Application 09/839147*. Online: http://www.fuentek.com/technologies/rhseg.htm

Tilton, J. C. (2007). Parallel implementation of the recursive approximation of an unsupervised hierarchical segmentation algorithm. In: *High-performance computing in remote sensing*, edited by A. Plaza and C.-I Chang. Boca Raton, FL: Chapman & Hall/CRC Press.

Valencia, D., Lastovetsky, A., and Plaza, A. (2006). Design and implementation of a parallel heterogeneous algorithm for hyperspectral image analysis using HeteroMPI. In: *Proceedings of the 5th International Symposium on Parallel and Distributed Computing* Vol. 1 pp. 301–308.

Wang, P., Liu, K. Y., Cwik, T., and Green, R. O. (2002). MODTRAN on supercomputers and parallel computers, *Parallel Computing* **28**(1): 53–64.