J Supercomput (2007) 40: 81–107
DOI 10.1007/s11227-006-0015-2

Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data

Antonio Plaza · Javier Plaza · David Valencia

Published online: 24 February 2007

Abstract The main objective of this paper is to describe a realistic framework to understand parallel performance of high-dimensional image processing algorithms in the context of heterogeneous networks of workstations (NOWs). As a case study, this paper explores techniques for mapping hyperspectral image analysis techniques onto fully heterogeneous NOWs. Hyperspectral imaging is a new technique in remote sensing that has gained tremendous popularity in many research areas, including satellite imaging and aerial reconnaissance. 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. Using an evaluation strategy which is based on comparing the efficiency achieved by an heterogeneous algorithm on a fully heterogeneous NOW with that evidenced by its homogeneous version on a homogeneous NOW with the same aggregate performance as the heterogeneous one, we develop a detailed analysis of parallel algorithms that integrate the spatial and spectral information in the image data through mathematical morphology concepts. For comparative purposes, performance data for the tested algorithms on Thunderhead (a large-scale Beowulf cluster at NASA's Goddard Space Flight Center) are also provided. Our detailed investigation of the parallel properties of the proposed morphological algorithms provides several intriguing findings that may help image analysts in selection of parallel techniques and strategies for specific applications.

Keywords Heterogeneous computing · Parallel algorithm design · Cluster computing · Performance analysis · Hyperspectral imaging · Mathematical morphology

A. Plaza (✉) · J. Plaza · D. Valencia
Department of Computer Science, University of Extremadura Polytechnic Institute of Caceres, Avda. de la Universidad s/n, E-10071, Caceres, Spain
e-mail: aplaza@unex.es

1 Introduction

Heterogeneous networks of workstations (NOWs) are a very promising type of distributed-memory parallel architecture [1]. Unlike traditional homogeneous parallel platforms, heterogeneous NOWs are composed of processors running at different speeds. This heterogeneity is seldom planned, arising as a result of technology evolution over time, or computer market sales and trends. Traditional parallel algorithms, which distribute computations evenly across the different processors, cannot balance the load of different-speed processors in heterogeneous NOWs, as faster processors will quickly perform their portions of computation and will have to wait for slower ones at points of synchronization. In heterogeneous computing, it is generally assumed that the processors are non-decomposable, i.e., multiple algorithms cannot be executed on the same processor simultaneously. This assumption is based on the fact that, in general terms, heterogeneous systems are operated as capacity computing resources in which a scheduler assigns different jobs to processors according to prescribed rules, which try to balance resource allocation while maximizing resource utilization. As a result, a *natural* solution to the load balancing problem in heterogeneous computing environments is to distribute data across processors unevenly so that each processor performs an amount of computation which is proportional to its speed. In particular, heterogeneous computing research has shown that, with careful job scheduling, heterogeneous collections of computing resources can usually outperform comparable homogeneous resource sets when the application set places varied demands on the computing nodes and the interconnection networks (see, for example, [2–5]). However, the problem of optimal heterogeneous data distribution has been proven to be NP-complete for fully heterogeneous NOWs, i.e., for systems where both the processing units and the communication links are heterogeneous in nature. As a result, most practical heterogeneous algorithms are sub-optimal [6].

Heterogeneous computing has recently started to play a major role in remotely sensed image processing applications [7, 8]. The large dimensionality and volume of image data sets collected in such applications have soon introduced the need for highly scalable processing platforms, coupled with efficient algorithms able to produce analysis results quickly enough for practical use [9]. For instance, data sets available for some areas such as satellite imaging and aerial reconnaissance continue to increase in size, in their spatial resolution, in the number of spectral channels, and in the ever growing sequence of images collected over time [10]. Latest-generation hyperspectral imagers, such as the Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) operated by NASA's Jet Propulsion Laboratory [11], are now able to measure reflected radiation in hundreds of spectral channels (see Fig. 1). It is estimated that NASA collects and sends to Earth more than 950 Gb of hyperspectral data on a daily basis.

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 [12, 13]. Although most parallel image processing systems employed by NASA and other institutions during the last

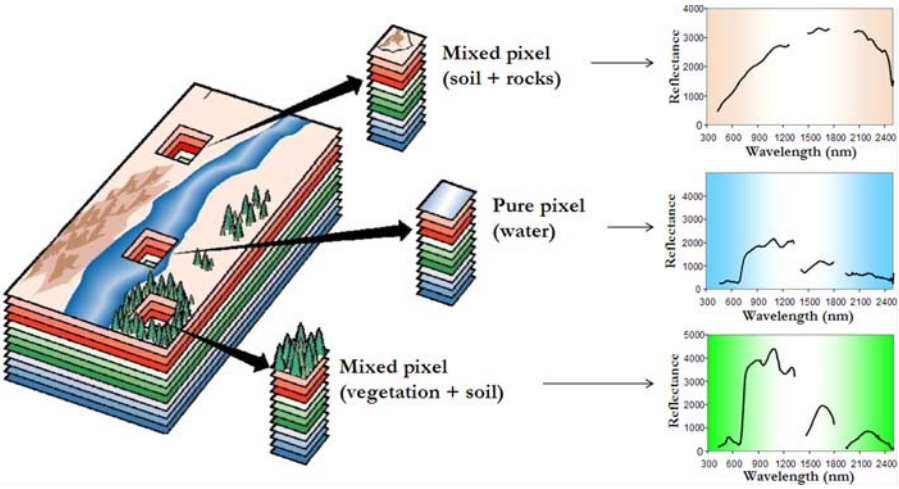


Fig. 1 The concept of hyperspectral imaging using Jet Propulsion Laboratory's AVIRIS system

decade have chiefly been homogeneous in nature, many available and planned parallel systems for space-based Earth science and planetary exploration are characterized by their heterogeneity. Considering the fact that commercial off-the-shelf heterogeneous clusters can realize a very high level of aggregate performance, it is expected that these clusters will soon represent a tool of choice for the scientific community devoted to high-dimensional image analysis in remote sensing and other fields (e.g., magnetic resonance imaging for medical applications). Due to the recent incorporation of heterogeneous computing to remote sensing-based research, significant opportunities to exploit such techniques are available in the field of hyperspectral imaging.

The main objective in this paper is to describe a general framework for the understanding of parallel efficiency in heterogeneous systems which run very high-dimensional image processing applications. The remainder of the paper is structured as follows. Section 2 formulates the optimization problem. Section 3 describes a morphological algorithm that integrates the spatial and spectral information in hyperspectral image data, and further develops a heterogeneous parallel processing framework which is designed to maximize load balance in heterogeneous environments. Section 4 assesses the performance of the proposed heterogeneous algorithms by comparing their efficiency on a fully heterogeneous NOW with the efficiency of their equivalent homogeneous versions on a homogeneous NOW with the same aggregate performance as the heterogeneous one, following a recent study by Lastovetsky and Reddy [6]. For comparative purposes, results achieved by heterogeneous algorithms running on Thunderhead, a large-scale homogeneous cluster at NASA's Goddard Space Flight Center, are also provided. Finally, Sect. 5 presents the main conclusions of this paper.

2 Optimization problem

Our target architecture in this study is assumed to be a fully heterogeneous cluster composed of different-speed processors that communicate through links of different capacities. This computing platform can be modeled as a complete graph $G = (P, E)$, where each node in the graph models a computing resource P_i weighted by its relative cycle-time w_i. Each edge in the graph models a communication link weighted by its relative capacity, where $c_{i,j}$ denotes the maximum capacity of the slowest link in the path of physical communication links from P_i to P_j. We also assume that the system has symmetric costs: $c_{i,j} = c_{j,i}$ [14]. Given the above assumptions, an abstract view of our problem is stated as follows. A sequential algorithm repeatedly operates on a large volume data set, for example, a hyperspectral where each pixel is given by an N-dimensional (N-D) vector of values or "pixel vector" (see Fig. 1). Let W be the total amount of work to be performed at each step of the algorithm. This workload is split into slices that are allocated to the available number of processors, p. At each step of the algorithm, the slices are updated locally, and boundary information of size S is exchanged between slices, as it is common practice in most low-level parallel image processing operations [15]. A major question arising at this point is how to slice the available data into chunks so that the parallel execution time is minimized. For that purpose, there is a need to load-balance the workloads of the participating resources. Another important issue is whether resource selection is required or not. In other words, there is no reason *a priori* that all available processors need to be involved in the best possible configuration. However, our considered hyperspectral data analysis application involves very large computations. And when W is large enough, it is reasonable to assume that all processors available will be involved, because the impact of the communications becomes smaller in light of the cost of the computations, and these computations should be distributed to all available computing resources.

With the above assumptions in mind, a simple and successful solution for standard parallel image processing techniques in the literature has been arranging the processors in a virtual ring. In such an arrangement, each processor will only communicate twice, once with its predecessor in the ring and once with the successor. Using a standard notation, let us denote as $P_{\text{pred}(i)}$ and $P_{\text{succ}(i)}$ the predecessor and successor of P_i. Then, P_i requires $S \cdot c_{i,\text{succ}(i)}$ time-units to send a message of size S to its successor, plus $S \cdot c_{\text{pred}(i),i}$ time-units to receive a message of the same size from its predecessor. Processor P_i will accomplish a share of $\alpha_i \cdot W$ of the total workload, where $\alpha_i \geq 0$ for $1 \leq i \leq p$ and $\sum_{i=1}^{p} \alpha_i = 1$. The total cost of a single step in the image processing algorithm is the maximum, over all processors, of the time spent on computations and communications, given by:

$$T_{\text{STEP}} = \max_{1 \leq i \leq p} \left\{ \alpha_i \cdot W \cdot w_i + S \cdot \left(c_{\text{pred}(i),i} + c_{i,\text{succ}(i)} \right) \right\}. \tag{1}$$

In the following section, we shall investigate a task-replication-based alternative solution to the optimization problem above. Task replication is a scheduling method that replicates selected tasks to run on more than one processor in order to reduce the inter-processor communication [16]. Specifically, our goal is to minimize communication time by selectively introducing redundant computations so that a processor P_i

will now accomplish a share of $\alpha_i \cdot (V + R)$, where V is the original data volume and R is the total amount of redundant information introduced in the system. As will be shown in the following section, redundant information in our application corresponds to the boundary information that needs to be exchanged between processors during the course of morphological processing.

3 Heterogeneous parallel processing framework

This section presents a heterogeneous parallel processing framework for high-dimensional image processing applications. The framework described here was implemented as an effective alternative for heterogeneous environments since it automatically adjusts the workload to be processed on each processor depending on the speed at which the information is expected to be processed at the nodes. First, we briefly describe a morphological processing algorithm that will serve as our case study throughout the paper. Then, we develop two parallel versions specifically designed for heterogeneous platforms. Important issues in algorithm design, such as volume partitioning and task scheduling are also discussed. This section ends with several notes to facilitate implementation and reproduction of our results.

3.1 Morphological image processing algorithm

Mathematical morphology [17] is a standard image processing technique that provides an appropriate framework to integrate the spatial and spectral information in hyperspectral imaging applications [18]. Let us consider a multi-dimensional image $\boldsymbol{f}$ defined in an N-D space, where N is the number of components or bands. To define morphological operations in a multi-dimensional space, we first need to impose an ordering relation in terms of spectral purity in the set of pixel vectors lying within a search window (structuring element), designed by B. This is done by defining a cumulative distance between one particular pixel $\boldsymbol{f}(x, y)$, where $\boldsymbol{f}(x, y)$ denotes an N-D vector at discrete spatial coordinates $(x, y) \in Z^2$, and all the pixel vectors in the spatial neighborhood given by B (B-neighborhood) as follows:

$$D_B[\boldsymbol{f}(x, y)] = \sum_s \sum_t \text{dist}[\boldsymbol{f}(x, y), \boldsymbol{f}(x + s, y + t)] \quad \forall (s, t) \in Z^2(B), \tag{2}$$

where dist is a point-wise distance measure between two N-D vectors. As a result, $D_B[\boldsymbol{f}(x, y)]$ is given by the sum of dist scores between $\boldsymbol{f}(x, y)$ and every other pixel vector in the B-neighborhood. Based on the cumulative metric above, the erosion [18] of $\boldsymbol{f}$ by B is based on the selection of the B-neighborhood pixel vector that produces the minimum value for D_B:

$$\begin{aligned}(\boldsymbol{f} \ominus B)(x, y) &= \left\{ \boldsymbol{f}\left(x + s', y + t'\right), \left(s', t'\right) \right. \\ &\left. = \text{argmin}_{(s,t) \in Z^2(B)} \left\{ D_B[\boldsymbol{f}(x + s, y + t)] \right\} \right\}, \quad (x, y) \in Z^2. \end{aligned} \tag{3}$$

Table 1 Pseudo-code of the sequential morphological processing algorithm

Inputs: N-D image $\boldsymbol{f}$, Structuring element B

Output: 2-D image MPS containing a morphological purity score for each pixel.

Begin

1. For each pixel vector $\boldsymbol{f}(x, y)$ do
2. $D_B[\boldsymbol{f}(x, y)] = 0$
3. For each $\boldsymbol{f}(x', y')$ in the B-neighborhood of $\boldsymbol{f}(x, y)$ do:
4. $D_B[\boldsymbol{f}(x, y)] = D_B[\boldsymbol{f}(x, y)] + \text{SAD}(\boldsymbol{f}(x, y), \boldsymbol{f}(x', y'))$
5. Endfor
6. Endfor
7. For each pixel vector $\boldsymbol{f}(x, y)$ do
8. $(\boldsymbol{f} \ominus B)(x, y) = \{\boldsymbol{f}(x + s', y + t'), (s', t') = \text{argmin}_{(s,t)\in Z^2(B)}\{D_B[\boldsymbol{f}(x + s, y + t)]\}\}$
9. $(\boldsymbol{f} \oplus B)(x, y) = \{\boldsymbol{f}(x - s', y - t'), (s', t') = \text{argmax}_{(s,t)\in Z^2(B)}\{D_B[\boldsymbol{f}(x - s, y - t)]\}\}$
10. $\text{MPS}(x, y) = \text{SAD}[(\boldsymbol{f} \ominus B)(x, y), (\boldsymbol{f} \oplus B)(x, y)]$
11. Endfor

End

On other hand, the dilation [18] of $\boldsymbol{f}$ by B selects the B-neighborhood pixel vector that produces the maximum value for D_B as follows:

$$(\boldsymbol{f} \oplus B)(x, y) = \{\boldsymbol{f}(x - s', y - t'), (s', t') = \text{argmax}_{(s,t)\in Z^2(B)}\{D_B[\boldsymbol{f}(x - s, y - t)]\}\}, \quad (x, y) \in Z^2. \quad (4)$$

Our choice for dist in Eq. (2) is a widely used distance metric in applications involving spectral data analysis: the spectral angle distance (SAD), which can be defined for two N-D pixels $\boldsymbol{f}(x', y')$ and $\boldsymbol{f}(x'', y'')$ as follows [10]:

$$\text{SAD}[\boldsymbol{f}(x', y'), \boldsymbol{f}(x'', y'')] = \cos^{-1}\left(\frac{\boldsymbol{f}(x', y') \cdot \boldsymbol{f}(x'', y'')}{\|\boldsymbol{f}(x', y')\| \|\boldsymbol{f}(x'', y'')\|}\right), \quad (5)$$

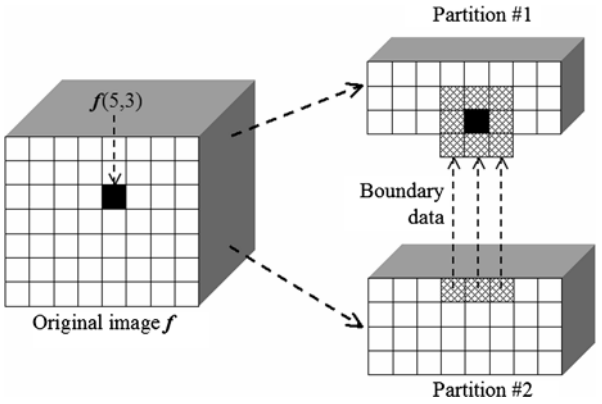
where $\boldsymbol{f}_i(x', y')$ and $\boldsymbol{f}_i(x'', y'')$ refer to the i-th component of pixel vectors $\boldsymbol{f}(x', y')$ and $\boldsymbol{f}(x'', y'')$, respectively. With the above definitions in mind, a sequential morphological processing algorithm is summarized in Table 1. This algorithm has been described before and we will not expand on its detailed implementation here; in particular, it has been demonstrated to outperform other available approaches for hyperspectral data analysis [18, 19]. The computational complexity is $O(\nu_{\boldsymbol{f}} \times \nu_B \times N)$, where $\nu_{\boldsymbol{f}}$ is the number of pixel vectors in the original image $\boldsymbol{f}$, ν_B is the number of pixels in the structuring element B, and N is the number of spectral bands. This results in very high computational processing cost in many applications [20]. However, an appropriate parallelization strategy can greatly enhance the computational performance of the algorithm. In the following sub-sections, we explore volume partitioning and scheduling strategies for implementing the above algorithm in heterogeneous computing environments.

3.2 Volume partitioning

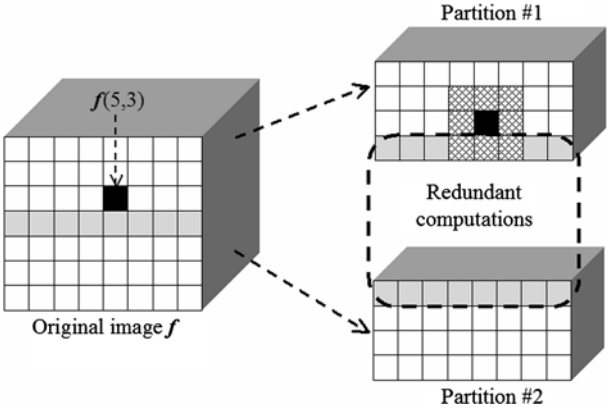
A major requirement for efficient parallel algorithms on distributed memory systems is finding a decomposition that minimizes the communication between the processors. Domain decomposition techniques [21] provide the greatest flexibility and scalability in parallel image processing [15]. Two types of partitioning can be exploited in multi-dimensional image analysis algorithms: spectral-domain partitioning and spatial-domain partitioning. Spectral-domain partitioning subdivides the volume into small cells or sub-volumes made up of contiguous spectral bands, and assigns one or more sub-volumes to each processor. With this model, each pixel vector is split amongst several processors and the communication cost for the proposed image processing algorithm is enormous, thus preventing efficient and scalable implementations [20]. In order to achieve load balance and to exploit parallelism as much as possible, a spatial-domain partitioning approach was adopted in our application. There are several reasons that justify our decision to use spatial-domain decomposition techniques. First, spatial-domain partitioning is a natural approach for low-level image processing, as many operations require the same function to be applied to a small set of elements around each entire pixel vector in the image volume [15]. A second reason has to do with the cost of inter-processor communication. In spectral-domain partitioning, the calculations made for each pixel vector need to originate from several processors, and thus require intensive inter-processor communication [22]. The overhead introduced by inter-processor communication will increase linearly with the increase in the number of processors. A final major reason is that spatial-domain partitioning greatly simplifies a processor arrangement in which every processor has a maximum of two neighbors, thus allowing a virtual ring implementation such as the one addressed in Sect. 2.

3.3 Task scheduling

Before describing our adopted approach for task scheduling, we should point out that an important issue in neighborhood-based image processing applications such as convolution or mathematical morphology is that additional inter-processor communications are required when the structuring element computation needs to be split amongst several different processing nodes due to boundary effects, as illustrated in Fig. 2a for a 3×3-pixel structuring element. In the example, the computations for a certain pixel need to originate from two heterogeneous processors, and a communication overhead involving three high-dimensional pixel vectors is introduced. Therefore, we need to revisit the general optimization problem stated in Sect. 2 in the context of our particular application. Each processor P_i needs to send some boundary data with size S to its neighbors $P_{\text{pred}(i)}$ and $P_{\text{succ}(i)}$. In a heterogeneous environment, it is clear that S does not represent a fixed volume of communication due to the fact that the size of the partitions depends on the cycle-time w_i of each processor. However, if an overlap border is added to one the adjacent partitions to avoid accesses outside image domain, as illustrated in Fig. 2b, then boundary data no longer need to be exchanged between neighboring processors. It is clear that such an overlap border would introduce redundant computations since the intersection between the two involved partitions would be non-empty. The amount of redundant information



(a)



(b)

Fig. 2 (a) 3 × 3-pixel structuring element computation split among two heterogeneous processing nodes. (b) Introduction of redundant computations to minimize inter-processor communication in a 3 × 3-pixel structuring element computation

would be given by S, the size of the boundary data. It should also be noted that the solution above may be prohibitive for large structuring element sizes. Subsequently, for a given platform, there is an application-dependent threshold to decide whether a redundant-information-based or data-exchange-based strategy should be adopted. Taking into account the particularities of our application, we can formally state our optimization problem as follows: given p processors of cycle-times w_i and $p(p-1)$ communication links of capacity $c_{i,j}$; given a total workload W and a communication overhead S at each step, the goal is to minimize:

$$T_{\text{STEP}} = \min\left\{ \max_{1\le i\le p}\left[\alpha_i \cdot (W+S)\cdot w_i\right], \max_{1\le i\le p}\left[\alpha_i \cdot W \cdot w_i + S\cdot\left(c_{\text{pred}(i),i} + c_{i,\text{succ}(i)}\right)\right]\right\}, \tag{6}$$

where the first term refers to the total cost of a single step of the algorithm when additional boundary data is *replicated*, and the second term refers to the cost of a single step when boundary data is *exchanged* between consecutive processors. Solving

the optimization problem in expression (6) is easy when all communication times are equal, i.e., when the target platform is given by a homogeneous communication network where each processor pair can communicate at the same speed: $c_{i,j} = c$ for all i, j. In that case, the load will be most balanced if the execution time is the same for all processors. This leads to $\alpha_i \cdot w_i \approx \text{const}$ for all processors, with $\sum_{i=1}^{p} \alpha_i = 1$. As a result, we can rewrite expression (6) for networks with homogeneous communication links as follows:

$$T_{\text{STEP}} = \min\{(W+S)\cdot w_{\text{cumul}}, W \cdot w_{\text{cumul}} + 2S \cdot c\}, \quad \text{where } w_{\text{cumul}} = \frac{1}{\sum_{i=1}^{p}(1/w_i)}, \tag{7}$$

with $w_i = w$ for all i. In the case of networks with fully heterogeneous communication links, the solution to the optimization problem in (6) would be similar to the one given in (7) for homogeneous communication networks if redundant computations are used, in which case we have $T_{\text{STEP}} = \min\{(W + S) \cdot w_{\text{cumul}}\}$. In this case, the workload assigned to each processor will only depend on the ratio between the total amount of information $W + R$ to be processed and the relative cycle-time, w_i. However, it has been demonstrated in the literature that the optimization problem in (6) is NP-complete if communication costs $c_{\text{pred}(i),i}$ and $c_{i,\text{succ}(i)}$ need to be taken into account [23]. In this case, all processors require the same amount of time to compute and communicate, with $T_{\text{STEP}} = \alpha_i \cdot W \cdot w_i + S \cdot (c_{\text{pred}(i),i} + c_{i,\text{succ}(i)})$. Since $\sum_{i=1}^{p} \alpha_i = 1$, we have $\sum_{i=1}^{p} \frac{T_{\text{STEP}} - S\cdot(c_{\text{pred}(i),i} + c_{i,\text{succ}(i)})}{W \cdot w_i} = 1$, which leads to:

$$T_{\text{STEP}} = W \cdot w_{\text{cumul}} \cdot \left[1 + \frac{S}{W} \sum_{i=1}^{p} (c_{\text{pred}(i),i} + c_{i,\text{succ}(i)})/w_i\right]. \tag{8}$$

Therefore, the goal in expression (8) is to minimize $\sum_{i=1}^{p}(c_{\text{pred}(i),i} + c_{i,\text{succ}(i)})/w_i$, which can be done by calculating the ring that corresponds to the shortest Hamiltonian cycle in the graph $G = (P, E)$, an NP-complete problem as pointed out above. In order to derive a polynomial-time heuristic for solving the optimization problem above, we resort to a greedy algorithm that is explained in the following subsection.

3.4 Implementation details

To provide an experimental validation of our approach and determine the best scheduling strategy in our particular case study, we have implemented two different heterogeneous algorithms using the MPI library. The first algorithm is called redundant-computation-based heterogeneous algorithm (RCHA), while the second is called data-exchange-based heterogeneous algorithm (DEHA). Both algorithms were programmed using MPICH for the intercommunication between processors. They use a server-client architecture [24, 25], in which the *server* node is responsible for the distribution of workload, while the *client* nodes carry out the morphological process and return their values to the server. The partitioner module has been implemented so that it scatters data structures without creating partial structures at the root. For that purpose, we made use of MPI *derived datatypes* to directly scatter hyperspectral data structures, which may be stored non-contiguously in memory, in a single communication step [26]. The parallel algorithms are divided into two units, the client program

Table 2 Pseudo-code of the server program for the RCHA algorithm

Inputs: N-D image $\boldsymbol{f}$, Structuring element B.

Output: 2-D image MPS containing a morphological purity score for each pixel.

Begin

1. Generate necessary system information, including the number of available processors p, each processor's $\{P_i\}_{i=1}^{p}$ identification number, and processor cycle-times $\{w_i\}_{i=1}^{p}$.
2. Using B and the information above, determine the total volume of information R that needs to be replicated from the original data volume V.
3. Set $\alpha_i = \left\lfloor \frac{(p/w_i)}{\sum_{i=1}^{p}(1/w_i)} \right\rfloor$ for all $i \in \{1, \ldots, p\}$.
4. For $m = \sum_{i=1}^{p} \alpha_i$ to $(V + R)$ do begin
5. Find $k \in \{1, \ldots, p\}$ such that $w_k \cdot (\alpha_k + 1) = \min\{w_i \cdot (\alpha_i + 1)\}_{i=1}^{p}$.
6. $\alpha_k = \alpha_k + 1$.
7. Use $\{\alpha_i\}_{i=1}^{p}$ to obtain a set of p spatial-domain heterogeneous partitions of $(V + R)$, and send its corresponding partition to each processor P_i along with B.
8. Collect all the individual results $\{\text{MPS}_i\}_{i=1}^{p}$ provided by each processor P_i, and merge them together to form a final image $\text{MPS} = \bigcup_{i=1}^{p}\{\text{MPS}_i\}$.

End

and the server program. In both cases, the client program is given by the sequential algorithm in Table 1. On the other hand, Table 2 shows the server program for RCHA algorithm. In step 1, the server program obtains the processor cycle-times $\{w_i\}_{i=1}^{p}$. In step 2, it determines the total volume of computation $(V + R)$ by using structuring element B, which is input as a parameter. Note that this calculation can be done *a priori* due to the regularity of morphological operations. Steps 3–6 calculate the load distribution, where load balance is achieved based on the assumption that the load of each processor P_i must be inversely proportional to its cycle-time w_i. Specifically, step 3 first approximates the $\{\alpha_i\}_{i=1}^{p}$ so that $\alpha_i \cdot w_i \approx \text{const}$ and $\sum_{i=1}^{p} \alpha_i \leq (V + R)$. Then, steps 4-6 iteratively increment some α_i until $\sum_{i=1}^{p} \alpha_i = (V + R)$. Step 7 uses the previously calculated distribution to partition the data set $\boldsymbol{f}$ using the approach described in Sect. 3.2, and provides each processor with its share of $(V + R)$ along with structuring element B so that the morphological processing algorithm in Table 1 can be applied locally at each partition. Finally, step 8 collects the individual outputs provided by the clients and merges them together to form the final result. The homogeneous version of RCHA algorithm is identical to the heterogeneous one described above, but replacing step 3 in Table 2 with $\alpha_i = p/w_i$ for all $i \in \{1, \ldots, p\}$, where w_i is a constant cycle-time for all processors in the network.

It should be noted that our implementation of the server program in DEHA algorithm simply replaces steps 2–7 in Table 2 by a greedy heuristic [27], which naturally arranges the processors in a virtual ring. The heuristic starts by selecting the fastest processor, i.e., $j \in \{1, \ldots, p\}$ is found such that $w_j = \max\{w_i\}_{i=1}^{p}$. Then, we iteratively include a new processor in the current solution ring. Let us assume that we have already selected a ring of r processors. For each remaining processor P_k, we need to search where to insert this processor in the current ring. Specifically, for each

pair $(P_i, P_{\text{succ}(i)})$, we compute

$$T_{\text{STEP}} = W \cdot w_{\text{cumul}} \cdot \left[1 + \frac{S}{W} \sum_{i=1}^{p} (c_{i,k} + c_{k,\text{succ}(i)})/w_i\right],$$

which gives the cost of inserting P_k between P_i and $P_{\text{succ}(i)}$ in the ring. Both the processor P_k and the pair that minimize the insertion cost are tallied, and the value of their associated T_{STEP} is stored. This step of the heuristic has a complexity proportional to $r(p-r)$. Finally, the ring is grown until all the processors are included, i.e., $r = p$. The total complexity of the heuristic above is $\sum_{r=1}^{p} r(p-r) = O(p^3)$. It should be noted that the homogeneous prototype of DEHA can be simply obtained by replacing the insertion cost used in the greedy algorithm above by $T_{\text{STEP}} = W \cdot w_{\text{cumul}} + 2S \cdot c$, where c is the constant speed of all the communication links in the homogeneous network. As a final comment, we note that the proposed heterogeneous techniques can be applied on the fly at runtime. In other words, the selection between a redundant-computation-based or a data-exchange-based scheduling strategy can be deferred to the actual moment of intended execution by comparing the values of T_{STEP} estimated by RCHA and DEHA in the server program, and adaptively selecting an optimal allocation on the fly before sending the partitions to the clients. In the following section, we provide a quantitative and comparative assessment of the parallel algorithms described above.

4 Performance evaluation

In this section, heterogeneous algorithms are compared with their homogeneous versions in the context of a realistic hyperspectral imaging application. The section is organized as follows. First, a framework introduced by Lastovetsky and Reddy for assessment of heterogeneous algorithms is presented and discussed in the context of our application. Next, we provide an overview of the parallel computing architectures used for evaluation purposes. Performance data are given in the last sub-section.

4.1 Framework for assessment of heterogeneous algorithms

We propose to assess heterogeneous algorithms using the principles and concepts introduced in a recent research study [6]. This work assumes that, typically, a heterogeneous algorithm is a modification of some homogeneous one. Therefore, a basic postulate is that the heterogeneous algorithm cannot be more efficient than its homogeneous prototype, which means that the heterogeneous algorithm cannot be executed on a heterogeneous NOW faster than its homogeneous prototype on the *equivalent* homogeneous NOW. Let us first explore the viability of using the evaluation framework above in our particular application. As shown in Sect. 3, RCHA algorithm solves the optimization problem in (6) by reducing inter-processor communications. On other hand, the communication cost of DEHA algorithm comes mainly from relatively rare point-to-point communications in a virtual ring distribution. Since we are dealing with high-dimensional pixel vectors, each communication consists of passing a relatively long message. The number of time-units to send a message of

size S from processor P_i to its successor $P_{\text{succ}(i)}$ in the virtual ring is $S \cdot c_{i,\text{succ}(i)}$, where $c_{i,\text{succ}(i)}$ is the constant speed of communications between P_i and $P_{\text{succ}(i)}$, and $c_{i,\text{succ}(i)} = c_{\text{succ}(i),i}$. Under the above assumptions, the only aggregate characteristic of the communication network that may have an impact on the execution time of the proposed heterogeneous algorithm is the average speed of point-to-point communications. Let us now assume that a heterogeneous NOW consists of $\{P_i\}_{i=1}^{p}$ workstations with different cycle-times, which span m communication segments $\{s_j\}_{j=1}^{m}$, and let $c^{(j)}$ be the communication speed of segment s_j. Similarly, let $p^{(j)}$ be the number of processors that belong to s_j, and let $w_t^{(j)}$ be the speed of the t-th processor connected to s_j, where $t = 1, \ldots, p^{(j)}$. Finally, let $c^{(j,k)}$ be the speed of the communication link between segments s_j and s_k, with $j, k = 1, \ldots, m$. According to [6], the heterogeneous NOW above can be considered equivalent to a homogeneous NOW with the same number of workstations, $\{Q_i\}_{i=1}^{p}$, and constant cycle-time w, interconnected through a homogeneous network with communication speed c, if the following expressions are satisfied:

$$c = \frac{\sum_{j=1}^{m} c^{(j)} \cdot [p^{(j)}(p^{(j)} - 1)/2] + \sum_{j=1}^{m} \sum_{k=j+1}^{m} p^{(j)} \cdot p^{(j)} \cdot c^{(j,k)}}{p(p-1)/2}, \tag{9}$$

$$w = \frac{\sum_{j=1}^{m} \sum_{t=1}^{p^{(j)}} w_t^{(j)}}{p}, \tag{10}$$

where Eq. (9) states that the average speed of point-to-point communications between processors $\{P_i\}_{i=1}^{p}$ in the heterogeneous cluster should be equal to the speed of point-to-point communications between processors $\{Q_i\}_{i=1}^{p}$ in the homogeneous cluster. On the other hand, Eq. (10) states that the aggregate performance of processors $\{P_i\}_{i=1}^{p}$ should be equal to the aggregate performance of processors $\{Q_i\}_{i=1}^{p}$. In the following subsection, we describe the specifications and configuration of the NOWs used for experiments in this work.

4.2 Platform description

We have experimented with three NOWs. The first one is a small-scale network of 16 different SGI, Solaris and Linux workstations, and four communication segments at University of Maryland. Table 3 shows the cycle-times of the heterogeneous processors in seconds per megaflop, where processors $\{P_i\}_{i=1}^{4}$ are attached to communication segment s_1, processors $\{P_i\}_{i=5}^{8}$ communicate through s_2, processors $\{P_i\}_{i=9}^{10}$ are interconnected via s_3, and processors $\{P_i\}_{i=11}^{16}$ share communication segment s_4. It should be noted that the values in Table 3 result from actual measurements and do not refer to theoretical peak values. The communication links between the different segments $\{s_j\}_{j=1}^{4}$ only support serial communication. For illustrative purposes, Table 4 shows the capacity of all point-to-point communications in milliseconds to transfer a one-megabit message between each processor pair (P_i, P_j) in the heterogeneous NOW. As it can be seen in Table 4, the communication network of the heterogeneous NOW consists of four relatively fast homogeneous communication segments interconnected by three slower communication links with capacities

Table 3 Processor cycle-times (in seconds per megaflop) for the heterogeneous cluster

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}
0.0058	0.0102	0.0206	0.0072	0.0102	0.0072	0.0072	0.0102	0.0072	0.0451	0.0131	0.0131	0.0131	0.0131	0.0131	0.0131

Table 4 Capacity of links (time in milliseconds to transfer a one-megabit message) for the heterogeneous cluster

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}
P_1	–	19.26	19.26	19.26	48.31	48.31	48.31	48.31	96.62	96.62	154.76	154.76	154.76	154.76	154.76	154.76
P_2	19.26	–	19.26	19.26	48.31	48.31	48.31	48.31	96.62	96.62	154.76	154.76	154.76	154.76	154.76	154.76
P_3	19.26	19.26	–	19.26	48.31	48.31	48.31	48.31	96.62	96.62	154.76	154.76	154.76	154.76	154.76	154.76
P_4	19.26	19.26	19.26	–	48.31	48.31	48.31	48.31	96.62	96.62	154.76	154.76	154.76	154.76	154.76	154.76
P_5	48.31	48.31	48.31	48.31	–	17.65	17.65	17.65	48.31	48.31	106.45	106.45	106.45	106.45	106.45	106.45
P_6	48.31	48.31	48.31	48.31	17.65	–	17.65	17.65	48.31	48.31	106.45	106.45	106.45	106.45	106.45	106.45
P_7	48.31	48.31	48.31	48.31	17.65	17.65	–	17.65	48.31	48.31	106.45	106.45	106.45	106.45	106.45	106.45
P_8	48.31	48.31	48.31	48.31	17.65	17.65	17.65	–	48.31	48.31	106.45	106.45	106.45	106.45	106.45	106.45
P_9	96.62	96.62	96.62	96.62	48.31	48.31	48.31	48.31	–	16.38	58.14	58.14	58.14	58.14	58.14	58.14
P_{10}	96.62	96.62	96.62	96.62	48.31	48.31	48.31	48.31	16.38	–	58.14	58.14	58.14	58.14	58.14	58.14
P_{11}	154.76	154.76	154.76	154.76	106.45	106.45	106.45	106.45	58.14	58.14	–	14.05	14.05	14.05	14.05	14.05
P_{12}	154.76	154.76	154.76	154.76	106.45	106.45	106.45	106.45	58.14	58.14	14.05	–	14.05	14.05	14.05	14.05
P_{13}	154.76	154.76	154.76	154.76	106.45	106.45	106.45	106.45	58.14	58.14	14.05	14.05	–	14.05	14.05	14.05
P_{14}	154.76	154.76	154.76	154.76	106.45	106.45	106.45	106.45	58.14	58.14	14.05	14.05	14.05	–	14.05	14.05
P_{15}	154.76	154.76	154.76	154.76	106.45	106.45	106.45	106.45	58.14	58.14	14.05	14.05	14.05	14.05	–	14.05
P_{16}	154.76	154.76	154.76	154.76	106.45	106.45	106.45	106.45	58.14	58.14	14.05	14.05	14.05	14.05	14.05	–

$c^{(1,2)} = 29.05$, $c^{(2,3)} = 48.31$, $c^{(3,4)} = 58.14$ milliseconds, respectively. Although the configuration described above is a quite typical and realistic one, we must point out that it represents a specific subtype of heterogeneous platforms, which generally disfavors communication-based algorithms (as compared to more generic heterogeneous setups) due to the fact that the segments communicate through serial channels.

The second parallel computing architecture used in experiments is a homogeneous NOW of 16 identical Linux workstations. The processor cycle-time of the $\{Q_i\}_{i=1}^{16}$ processors is $w = 0.0131$ seconds per megaflop, and they are interconnected via a homogeneous network with a capacity of $c = 26.64$ milliseconds. It should also be noted that the same processors $\{P_i\}_{i=11}^{16}$ used in the heterogeneous NOW were also used to construct the homogeneous one, which allowed us to better control the accuracy of experiments by ensuring that these processors had the same speed in the heterogeneous NOW running an heterogeneous algorithm, and in the equivalent homogeneous NOW running its corresponding homogeneous algorithm. It is also important to emphasize that the configuration of the two platforms above was custom-designed to satisfy equations (9) and (10).

Finally, in order to test the heterogeneous algorithms on a larger-scale parallel platform, we also experimented with Thunderhead, a Beowulf cluster located at NASA's Goddard Space Flight Center. It is composed of 512 Linux workstations at 2.4 GHz, interconnected via 1.2 Gbps Myrinet homogeneous communication network. Thunderhead has been widely used in the past for analyzing high-dimensional, remotely sensed images. Analyzing the performance of the proposed heterogeneous algorithms on this platform is of great interest in order to calibrate their performance, scalability and portability to existing massively parallel computers, and also to relate to previous studies in this area.

4.3 Experimental results

The parallel algorithms were applied to a hyperspectral scene collected by the AVIRIS instrument. The data set was gathered over the Indian Pines test site in Northwestern Indiana, a mixed agricultural/forested area, and is characterized by very high spectral resolution (224 narrow spectral bands in the range 0.4–2.5 μm) and moderate spatial resolution of 20-meter pixels. It represents a very challenging classification problem, as illustrated in Fig. 3a. Extensive ground-truth information is available for the area [see Fig. 3b], a fact that has made this scene a standard data set for evaluating hyperspectral analysis algorithms. The full scene consists of 2048×512 pixel vectors, where each vector value is codified using 2 bytes. As a result, the total size of the image exceeds 470 Mbytes. Part of these data, including ground-truth, are available online (from http://dynamo.ecn.purdue.edu/~biehl/MultiSpec). As a result, people interested in the proposed algorithms can reproduce our results and conduct their experiments to exploit various scenarios.

The two proposed heterogeneous algorithms, RCHA and DEHA, were applied to the AVIRIS image shown in Fig. 3a using structuring element sizes: $B_{3\times3}$, $B_{5\times5}$, $B_{7\times7}$, $B_{9\times9}$, $B_{11\times11}$, $B_{13\times13}$ and $B_{15\times15}$. Table 5 shows the percentages of correctly classified pixels obtained by a classification technique based on the proposed morphological processing algorithm [19]. For illustrative purposes, single-processor execution times of RCHA and DEHA algorithms on Thunderhead are also reported. In

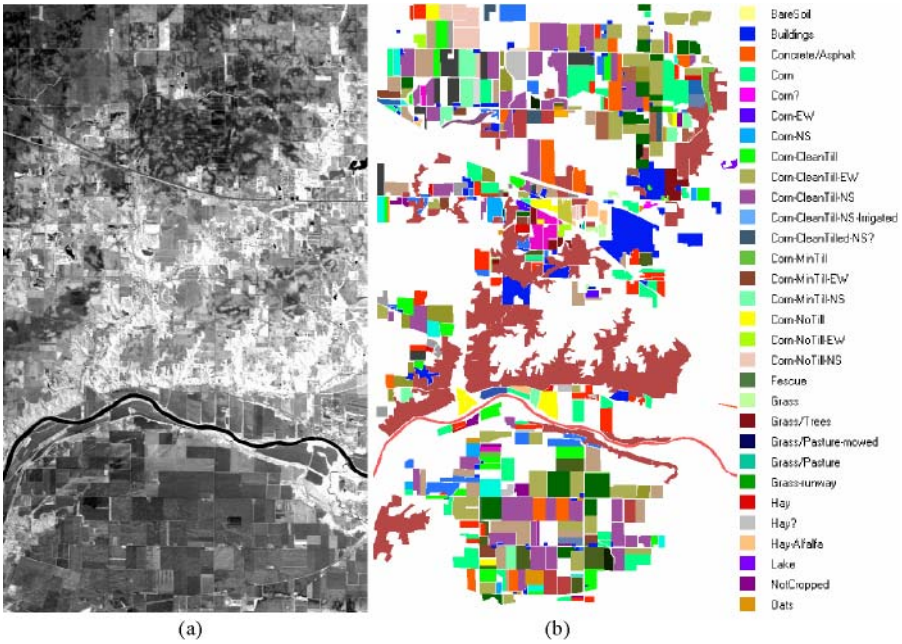


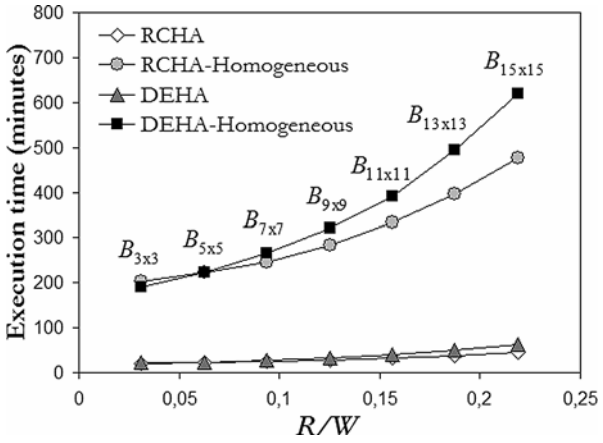
Fig. 3 (a) Spectral band at 587 nm wavelength of an AVIRIS scene comprising agricultural and forest features at Indian Pines test site, Indiana. (b) Ground-truth map with 30 mutually-exclusive land-cover classes

Table 5 Percentages of correctly classified pixels in the AVIRIS scene by a classification algorithm based on the proposed morphological processing, and single-processor running times (in minutes) on Thunderhead for RCHA and DEHA algorithms

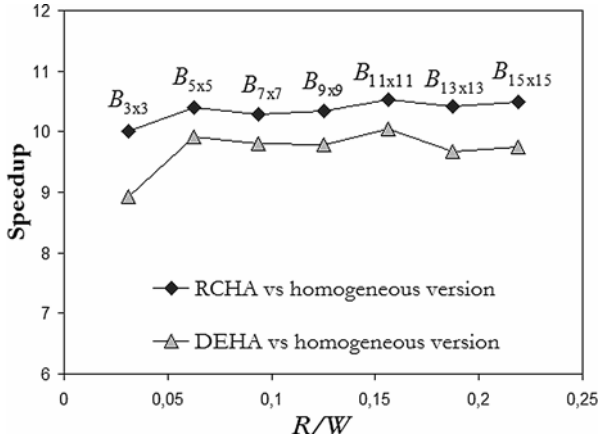
	$B_{3\times3}$	$B_{5\times5}$	$B_{7\times7}$	$B_{9\times9}$	$B_{11\times11}$	$B_{13\times13}$	$B_{15\times15}$
Classification accuracy	65.34	73.48	80.29	84.05	90.13	90.55	90.96
Single-processor time (RCHA)	200	239	284	321	368	426	515
Single-processor time (DEHA)	234	275	328	372	432	534	637

both cases, several hours of computation were required to produce the final classification results.

Figure 4a plots the execution times of RCHA, DEHA and their homogeneous prototypes (in seconds) on the heterogeneous NOW, as a function of the ratio R/W. It is worth noting that the execution times above include the data transfer time from the server, and that the R/W ratio is different for each considered structuring element size, where R denotes the amount of redundant information introduced in the system due to the spatial properties of the structuring element and W represents the constant volume of information contained in the original image. As expected, results in Fig. 4a show that heterogeneous algorithms were able to adapt much better to the heterogeneous NOW. It should be noted that DEHA slightly outperformed RCHA when $B_{3\times3}$ was used. For $B_{7\times7}$ and higher, RCHA outperformed DEHA due to a substantial increase in communication overhead. In other words, a very *simple* and *natural*



(a)



(b)

Fig. 4 (a) Execution times (in minutes) of the heterogeneous and homogeneous algorithms on the heterogeneous NOW. (b) Speedup of the heterogeneous algorithms over their corresponding homogeneous versions on the heterogeneous NOW

load balancing strategy based on: (1) replacing most communication costs by redundant computations, and (2) distributing the total workload $W + R$ proportionally to the speed of heterogeneous processors, proved to be more effective, computationally, than exchanging boundary data by taking into account the properties of the heterogeneous communication network in a virtual ring arrangement. This fact reveals that redundant computations can provide a cost-effective solution to increase parallel efficiency of heterogeneous algorithms dealing with very large data volumes.

For the sake of comparison, Fig. 4b plots the speedup of the heterogeneous algorithms over their corresponding homogeneous versions on the heterogeneous NOW, as a function of R/W. It can be seen that both RHCA and DEHA were approximately ten times faster than their homogeneous versions. The speedup was simply calculated

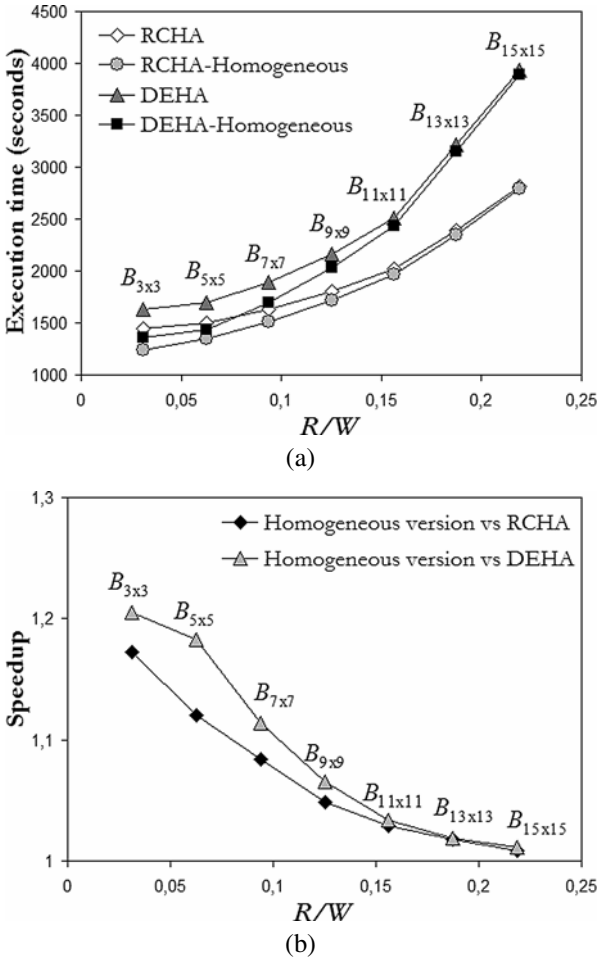


Fig. 5 (a) Execution times (in seconds) of the heterogeneous and homogeneous algorithms on the homogeneous NOW. (b) Speedup of the homogeneous algorithms over their corresponding heterogeneous algorithms on the homogeneous NOW

as the execution time of the homogeneous algorithm divided by the execution time of the heterogeneous algorithm for the same R/W ratio. As shown by Fig. 4b, the speedups achieved by RHCA were almost constant, while one can appreciate slight fluctuations in the speedup factors achieved by DEHA. This might be explained by the fact that the greedy heuristic used to solve the optimization problem may not be optimal for the considered problem. In particular, the optimization algorithm was developed from a general-purpose point of view, and hence it does not take into account the role of serial connectors between the segments when placing the processors in the virtual ring in the considered heterogeneous architecture.

Similarly, Fig. 5a shows a comparison of the execution times of RCHA and DEHA and their homogeneous versions on the homogeneous NOW, while Fig. 5b shows the speedup of the homogeneous algorithms over the heterogeneous ones on the homo-

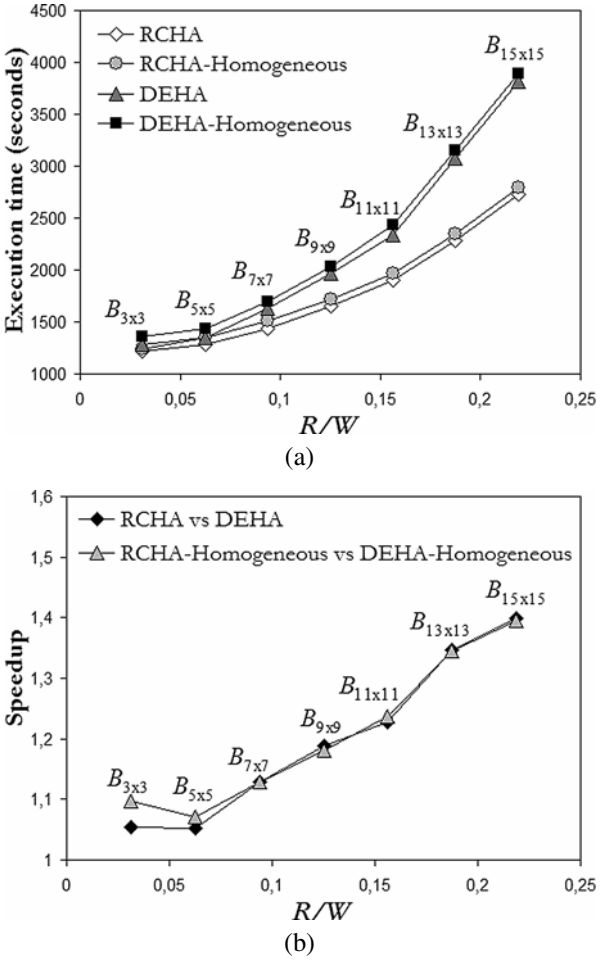


Fig. 6 (a) Execution times (in seconds) of the heterogeneous algorithms on the heterogeneous NOW, and execution times of the homogeneous algorithms on the equivalent homogeneous NOW. (b) Speedup achieved by redundant-computation-based algorithms over data-exchange-based algorithms on the heterogeneous and homogeneous NOWs

geneous NOW, as a function of R/W. Figure 5a reveals that the homogeneous algorithms slightly outperformed the heterogeneous ones for small structuring element sizes. However, as the volume of computation was increased, the heterogeneous algorithms achieved very similar performance to their respective homogeneous counterparts. One can see in Fig. 5b that the performance of heterogeneous algorithms is almost the same as that of homogeneous algorithms for large structuring element sizes. This demonstrates the flexibility of the proposed heterogeneous algorithms, which seemed to be able to adapt efficiently to homogeneous computing environments, in particular, when the volume of computations involved was extremely large.

Figure 6a shows a comparison of the execution times of RCHA and DEHA performed on the heterogeneous NOW, and of their homogeneous prototypes performed

on the homogeneous NOW. As Fig. 6a shows, both RCHA and its homogeneous version achieved almost the same speed, but each on its network. The same observation can be made for DEHA and its homogeneous version. This indicated that, in both cases, the proposed heterogeneous algorithm was very close to the optimal heterogeneous modification of the basic homogeneous algorithm [6]. It is also important to emphasize that the algorithms produced very close results for all considered structuring element sizes. In particular, the sub-optimal practical heuristic implemented by DEHA proved to be both fast and efficient. Other heuristics, however, may be more effective for small structuring elements in the context of our application. In this regard, further experimentation with additional heuristics is a topic deserving future research. Most importantly, results in Fig. 6a indicated that redundant-computation-based scheduling implemented by RCHA was more appropriate than data-exchange-based scheduling implemented by DEHA in our considered application, in particular, when relatively large structuring element sizes were considered. For illustrative purposes, Fig. 6b plots the speedup achieved by RCHA over DEHA in the same heterogeneous environment. It also plots the speedup achieved by the homogeneous version of RCHA over the homogeneous version of DEHA in the same homogeneous environment.

At this point, a more detailed discussion on the use of subtask duplication in the RCHA algorithm to enhance overall system performance is noteworthy. In particular, we focus on how overlap (redundant) computations are handled on the initial data scattering operation performed by the algorithm. To analyze this relevant issue in more detail, we have conducted an experimental comparison between the standard RCHA implementation (described in Table 2), and two alternative implementation strategies:

1. *Standard non-overlapping scatter*. In this alternative implementation of RCHA, the data is first divided into chunks without overlapping, and then an overlap communication follows for every structuring element-based computation (as shown in Fig. 2b), thus sending very small sets of pixels very often.
2. *Modified non-overlapping* scatter. In a second alternative version of RCHA, a standard non-overlapping scatter is followed by an overlap communication, to have all data available in the overlap border areas *before* the morphological computations (thus sending all redundant border data beforehand, but only once).

It should be noted that a major difference between the two alternative strategies discussed above and the standard RCHA implementation used throughout the paper is that, in the standard RCHA, a *combined* overlapping scatter operation is implemented, one that also sends out the overlap border data as part of the initial scatter operation itself. For illustrative purposes, Fig. 7a compares the execution times reported by the two alternative versions of RCHA on the heterogeneous NOW, and the execution times of their homogeneous versions on the homogeneous NOW, where the two alternative implementation strategies were respectively labeled as RCHA-1, i.e., overlap communication for every single pixel; and RCHA-2, i.e., overlap communication to have all data available before the morphological filtering. A comparison between Fig. 7a and Fig. 6a reveals that the execution times measured for RCHA-1 were much higher than those measured for RCHA, while the performance of RCHA-2 was much closer to that evidenced by the standard algorithm.

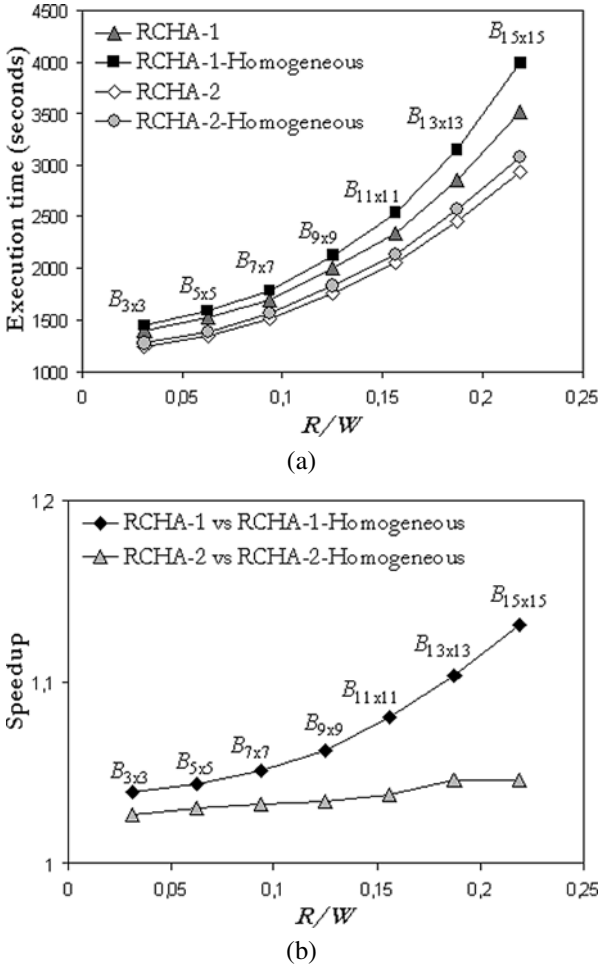
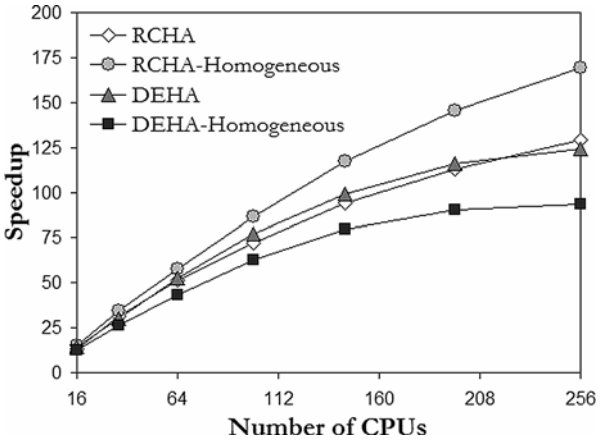
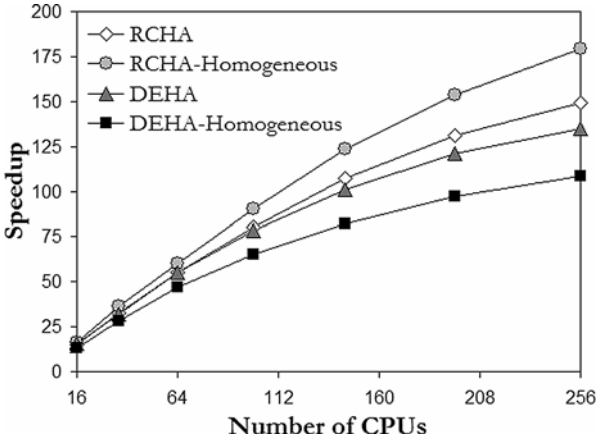


Fig. 7 (a) Execution times (in seconds) of two alternative versions of RCHA (called RCHA-1 and RCHA-2) on the heterogeneous NOW, and execution times of the corresponding homogeneous versions on the equivalent homogeneous NOW. (b) Speedup of the two alternative versions of RCHA over their corresponding homogeneous versions on the heterogeneous NOW

On the other hand, Fig. 7b plots the speedup of the heterogeneous versions of RCHA-1 and RCHA-2 over their corresponding homogeneous prototypes, as a function of R/W and executed on the heterogeneous platform. The main reason why RCHA-1 performed less effectively (in particular, as the amount of redundant computations increases) is due to its very expensive communication strategy. On the other hand, results in Fig. 7b show that RCHA-2 implemented a better overlap communication strategy, which produces better speedups as the ratio R/W is increased. Overall, experimental results in Fig. 7 reveal the importance of the initial data scattering on RCHA, and further demonstrate that the adopted implementation is significantly better than RCHA-1 and slightly better than RCHA-2. With the above results in mind, the rest of our argumentation will be based on the standard RCHA implementation.



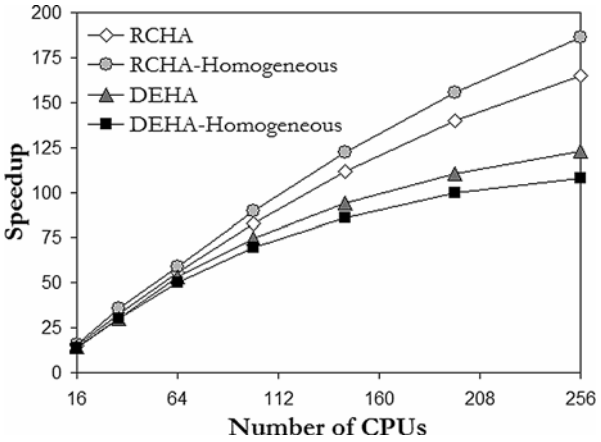
(a)



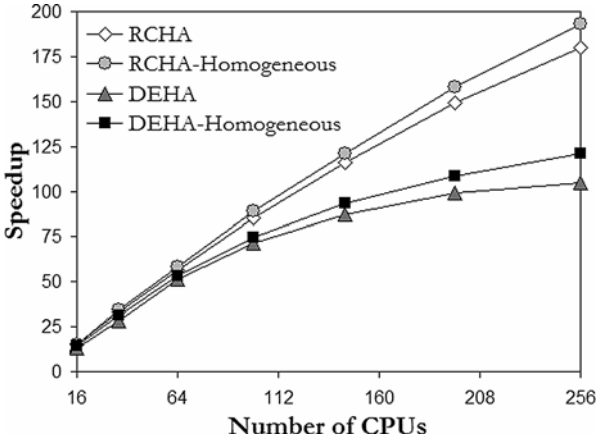
(b)

Fig. 8 Scalability of the heterogeneous and homogeneous algorithms on Thunderhead using structuring elements: (a) $B_{3\times3}$ with $R/W = 0.031$; (b) $B_{7\times7}$ with $R/W = 0.093$; (c) $B_{11\times11}$ with $R/W = 0.156$; (d) $B_{15\times15}$ with $R/W = 0.218$

In order to measure load balance [28] of RCHA and DEHA, Table 6 shows the imbalance scores achieved by the two algorithms on the heterogeneous NOW, defined as $D = T_{\text{MAX}}/T_{\text{MIN}}$. Therefore, perfect balance was 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}. It is clear from Table 6 that load balance was achieved in most cases, with both algorithms resulting in almost the same results for D_{ALL} and D_{MINUS}. Taking in mind the results presented above, and with the ultimate goal of exploring issues of scalability and portability of the proposed heterogeneous algorithms to existing massively parallel computing platforms, we have also compared the performance of RCHA, DEHA and their homogeneous versions on NASA's Thunderhead system. In particular, Fig. 8 shows the speedups achieved by multi-processor runs of RCHA, DEHA and their homogeneous versions over the



(c)



(d)

Fig. 8 (Continued)

single-processor execution times in Table 5. Four different structuring element sizes, i.e., $B_{3\times3}$ (see Fig. 8a), $B_{7\times7}$ (Fig. 8b), $B_{11\times11}$ (Fig. 8c), and $B_{15\times15}$ (Fig. 8d) were considered in experiments, with their corresponding R/W ratios given by 0.031, 0.093, 0.156 and 0.218, respectively. As Fig. 8 shows, the scalability of heterogeneous algorithms approached that of their homogeneous prototypes as the ratio R/W was increased, but one can see that the RCHA algorithm scaled slightly better than DEHA. On the other hand, Table 7 shows the execution times achieved by RCHA and DEHA algorithms on Thunderhead, using different numbers of processors. Results in Table 7 reveal that the tested algorithms were able to obtain highly accurate classification scores (see Table 5) but also quickly enough for practical use. For instance, using 256 processors the RCHA algorithm provided a 90%-accurate classification of the AVIRIS scene in about 2 minutes, while the DEHA algorithm was able to provide the same output in 2.5 minutes. The above results indicate significant improvements over the single-processor execution times reported on Table 5.

Table 6 Load balancing rates for RCHA and DEHA algorithms on the 16-processor heterogeneous network

Structuring element	RCHA		DEHA	
	D_{ALL}	D_{MINUS}	D_{ALL}	D_{MINUS}
$B_{3\times3}$	1.098	1.046	1.103	1.062
$B_{5\times5}$	1.076	1.027	1.095	1.055
$B_{7\times7}$	1.073	1.024	1.078	1.051
$B_{9\times9}$	1.070	1.023	1.075	1.049
$B_{11\times11}$	1.065	1.022	1.071	1.046
$B_{13\times13}$	1.063	1.021	1.062	1.040
$B_{15\times15}$	1.047	1.018	1.059	1.035

Table 7 Execution times (in seconds) by RCHA and DEHA algorithms in Fig. 7, obtained on Thunderhead using different structuring elements and numbers of processors

Structuring element	RCHA							DEHA						
	16	36	64	100	144	196	256	16	36	64	100	144	196	256
$B_{3\times3}$	896	382	234	166	127	106	93	989	464	268	181	141	121	113
$B_{7\times7}$	1121	527	309	212	159	130	114	1235	610	356	251	194	162	145
$B_{11\times11}$	1473	675	395	267	198	157	134	1779	856	487	347	275	235	210
$B_{15\times15}$	2087	933	545	362	266	207	171	2889	1348	746	536	437	386	364

Summarizing, experimental results in our study reveal that heterogeneous parallel algorithms offer a surprisingly simple, platform-independent, and scalable solution in the context of realistic image processing applications. We feel that the applicability of the proposed techniques extends beyond the domain of high-dimensional image 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

This paper provided a detailed discussion on the effects that platform heterogeneity has on degrading parallel performance in the context of applications dealing with large volumes of image data. Two representative parallel image processing algorithms were thoroughly analyzed. The first one minimizes inter-processor communication via task replication. The second one develops a polynomial-time heuristic for finding the best distribution of available processors along a fully heterogeneous ring. An interesting finding by experiments is that heterogeneous algorithms offer a surprisingly simple, yet effective and scalable solution in the context of very high-dimensional image processing applications. The use of redundant computations allowed us to inject knowledge in scheduling and mapping decisions about the computation and communication costs associated to each data chunk so that excellent performance can be

achieved via simple static load balancing strategies. Although the problem of distributing independent chunks of work to a fully heterogeneous one-dimensional array of processors can be addressed by the development of fast and effective practical heuristics, the best results in our case study of high-dimensional imaging on a realistic NOW were obtained by task replication strategies aimed at minimizing communication overhead. Our experimental results revealed important algorithmic aspects that may be of great importance for designing and adapting existing high-performance hyperspectral imaging applications (developed in the context of homogeneous computing platforms) to highly heterogeneous 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.

Acknowledgement This research was supported by the European Commission through the project "Performance analysis of endmember extraction and hyperspectral analysis algorithms," (contract no. HPRI–1999–00057). Additional funding from Junta de Extremadura (local government) through project 2PR03A026 is also gratefully acknowledged. The authors would like to thank Drs. John E. Dorband, James C. Tilton and J. Anthony Gualtieri for many helpful discussions, and also for their collaboration in the development of experiments on the Thunderhead cluster. The first author also acknowledges support received from the Spanish Ministry of Education and Science (Fellowship PR2003-0360) to conduct postdoctoral research at NASA's Goddard Space Flight Center and University of Maryland.

References

1. Lastovetsky A (2003) Parallel computing on heterogeneous networks. Wiley-Interscience, Hoboken
2. Casanova H, Thomason M, Dongarra J (1999) Stochastic performance prediction for iterative algorithms in distributed environments. J Parallel Distrib Comput 58:68–91
3. Braun TD, Siegel HJ, Beck N, Bölöni L, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B, Hensgen DA, Freund RF (2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distrib Comput 61:810–837
4. Beaumont O, Boudet V, Rastello F, Robert Y (2001) Matrix multiplication on heterogeneous platforms. IEEE Trans Parallel Distrib Syst 12:1033–1051
5. Lin C (2004) Heuristic contention-free broadcast in heterogeneous networks of workstations with multiple send and receive speeds. J Supercomput 30:37–64
6. Lastovetsky A, Reddy R (2004) On performance analysis of heterogeneous parallel algorithms. Parallel Comput 30:1195–1216
7. Hawick KA, Coddington PD, James HA (2003) Distributed frameworks and parallel algorithms for processing large-scale geographic data. Parallel Comput 29:1297–1333
8. Aloisio G, Cafaro M (2003) A dynamic earth observation system. Parallel Comput 29:1357–1362
9. Veeravalli B, Ranganath S (2003) Theoretical and experimental study on large size image processing applications using divisible load paradigm on distributed bus networks. Image Vis Comput 20:917–935
10. Chang C-I (2003) Hyperspectral imaging: techniques for spectral detection and classification. Kluwer Academic, New York
11. Green RO et al (1998) Imaging spectroscopy and the airborne visible/infrared imaging spectrometer AVIRIS. Remote Sens Environ 65:227–248
12. Dorband J, Palencia J, Ranawake U (2003) Commodity computing clusters at Goddard Space Flight Center. J Space Commun 1:1–12
13. Available online: http://satjournal.tcom.ohiou.edu/pdf/Dorband.pdf
14. Brightwell R, Fisk LA, Greenberg DS, Hudson T, Levenhagen M, Maccabe AB, Riesen R (2000) Massively parallel computing using commodity components. Parallel Comput 26:243–266

15. Culler DE, Singh JP (1999) Parallel computer architecture: a hardware/software approach. Morgan Kaufmann, San Francisco
16. Seinstra FJ, Koelma D, Geusebroek JM (2002) A software architecture for user transparent parallel image processing. Parallel Comput 28:967–993
17. Darbha S, Agrawal DP (1998) Optimal scheduling algorithm for distributed memory systems. IEEE Trans Parallel Distrib Syst 9:87–95
18. Soille P (2003) Morphological image analysis: principles and applications, 2nd edn. Springer, Berlin
19. Plaza A, Martínez P, Pérez RM, Plaza J (2002) Spatial/spectral endmember extraction by multidimensional morphological operations. IEEE Trans Geosci Remote Sens 40:2025–2041
20. Plaza A, Martínez P, Pérez RM, Plaza J (2004) A new approach to mixed pixel classification of hyperspectral imagery based on extended morphological profiles. Pattern Recognit 37:1097–1116
21. Plaza A, Valencia D, Plaza J, Chang C-I (2006) Parallel implementation of endmember extraction algorithms from hyperspectral data. IEEE Geosci Remote Sens Lett 3:334–338
22. Prieto M, Llorente IM, Tirado F (2000) Data locality exploitation in the decomposition of regular domain problems. IEEE Trans Parallel Distrib Syst 11:1141–1149
23. Plaza A, Valencia D, Plaza J, Martínez P (2006) Commodity cluster-based parallel processing of hyperspectral imagery. J Parallel Distrib Comput 66:345–358
24. Garey MR, Johnson DS (1991) Computers and intractability: a guide to the theory of NP-completedness. Freeman, New York
25. Banino C, Beaumont O, Carter L, Ferrante J, Legrand A, Robert Y (2004) Scheduling strategies for master-slave tasking on heterogeneous processor platforms. IEEE Trans Parallel Distrib Syst 15:319–330
26. Bazterra VE, Cuma M, Ferraro MB, Facelli JC (2005) A general framework to understand parallel performance in heterogeneous clusters: analysis of a new adaptive parallel genetic algorithm. J Parallel Distrib Comput 65:48–57
27. Seinstra FJ, Koelma D (2002) P-3PC: A point-to-point communication model for automatic and optimal decomposition of regular domain problems. IEEE Trans Parallel Distrib Syst 13:758–768
28. Renard H, Robert Y, Vivien F (2003) Static load-balancing techniques for iterative computations on heterogeneous clusters. Technical Report RR-2003-12. Laboratoire de l'Informatique du Parall(c)lisme (LIP), Ecole Normale Sup(c)rieure de Lyon, France. Available online: http://www.ens-lyon.fr/LIP/Pub/Rapports/RR/RR2003/RR2003-12.ps.gz

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 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 co-authored more than one hundred 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.

Javier Plaza received his Ph.D. degree in Computer Science from the University of Extremadura, Spain, in 2007, 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.

David Valencia is an Assistant Professor at the University of Extremadura, Spain, where he is currently pursuing the Ph.D. degree. His research interests are in the development of parallel implementations of algorithms for high-dimensional data analysis, with particular emphashis 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.