# Parallel Detection of Targets in Hyperspectral Images Using Heterogeneous Networks of Workstations

Antonio Plaza, David Valencia, Soraya Blazquez and Javier Plaza
Department of Computer Science, University of Extremadura
Avda. de la Universidad s/n, E-10071 Caceres, SPAIN
aplaza@unex.es

## Abstract

*Heterogeneous networks of workstations have rapidly become a cost-effective computing solution in many application areas. This paper develops several highly innovative parallel algorithms for target detection in hyperspectral imagery, considered to be a crucial goal in remote sensing-based homeland security and defense applications. In order to illustrate parallel performance, we consider four (partially and fully) heterogeneous networks of workstations distributed among different locations at University of Maryland, and also a massively parallel Beowulf cluster at NASA's Goddard Space Flight Center. Experimental results indicate that heterogeneous networks can be used as a viable low-cost alternative to homogeneous parallel systems in many on-going and planned remote sensing missions.*

## 1. Introduction

Hyperspectral imagers such as the NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) [6] are now able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of the reflected light of an area 2 to 12 kilometers wide and several kilometers long using 224 spectral bands [3]. The resulting 'image cube' is a stack of images in which each pixel (vector) has an associated spectral signature or 'fingerprint' that uniquely characterizes the underlying objects. The resulting data volume typically comprises several GBs per flight. Such wealth of spectral information provided by last-generation sensors has opened ground-breaking perspectives in many applications, including target detection for military and defense/security deployment. In particular, algorithms for detecting (moving or static) targets often require timely responses for swift decisions, which depend upon high computing performance of algorithm analysis [3]. Despite the growing interest in hyperspectral imaging research, only a few parallel hyperspectral algorithms exist in the open literature [7, 8, 5, 15]. However, with the recent explosion in the amount and dimensionality of hyperspectral imagery, parallel processing is expected to become a requirement in most remote sensing missions [19].

In this paper, we take a necessary first step towards the comparison of target detection algorithms implemented on parallel platforms. Although a few dedicated supercomputers have been employed by NASA and other institutions for this purpose, most of them are fully homogeneous in nature [4]. Quite opposite, a current trend in scientific and engineering applications is to utilize highly heterogeneous, distributed platforms which can benefit from local (user) computing resources. In particular, heterogeneous networks of workstations (NOWs) enable the use of existing resources [9]. Furthermore, such NOWs provide incremental scalability of hardware components. In other words, well-tuned parallel programs can be easily scaled to large configurations because additional workstations can always be added to a heterogeneous NOW. Also, these systems provide a high-degree of performance isolation, i.e., they allow analyzing parallel performance on a node-by-node basis. Finally, the technological evolution currently allows heterogeneous NOWs to support a variety of different workloads, including parallel, sequential and interactive jobs, as well as scalable computation-intensive applications. In particular, heterogeneous computing greatly benefits from previous work in dynamic, resource-aware task scheduling and load balancing in distributed platforms, including Grid systems [20, 2, 1].

The remainder of the paper is structured as follows. Section 2 outlines several considerations for the design of parallel target detection algorithms. Section 3 introduces new heterogeneous algorithms for this purpose. Section 4 assesses the performance of the parallel algorithms by comparing their accuracy and parallel properties using several heterogeneous and homogeneous platforms. Finally, section 5 concludes with some remarks.

15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP'07)
0-7695-2784-1/07 $20.00 © 2007 IEEE

## 2. Parallel algorithm design

### 2.1 Optimization problem

A fully heterogeneous NOWs can be modeled as a complete graph $G = (P, E)$, where each node 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 $c_{ij}$ denotes the maximum capacity of the slowest link in the path of physical communication links from $p_i$ to $p_j$ (we assume that the system has symmetric costs, i.e., $c_{ij} = c_{ji}$. With the above assumptions in mind [11], processor $p_i$ should accomplish a share of $\alpha_i \cdot W$ of the total workload, denoted by $W$, to be performed by a certain algorithm, with $\alpha_i \geq 0$ for $1 \leq i \leq P$ and $\sum_{i=1}^{P} \alpha_i = 1$. Taking into account the standard optimization problem above, an abstract view of our proposed hyperspectral image processing framework can be simply given in the form of a client-server architecture, in which a server processor is responsible for the efficient distribution of work among the $P$ nodes, the clients operate with the spectral signatures contained in a local partition, and some communications may also take place.

### 2.2 Data partitioning strategies

In a data-driven application environment such as the one described above, it is important to define efficient data partitioning strategies [18]. Two standard approaches have been traditionally considered for this purpose in remote sensing applications [15]:

- *Spectral-domain partitioning*. This approach subdivides the multi-channel remotely sensed image into small cells or sub-volumes made up of contiguous spectral wavelengths.

- *Spatial-domain partitioning*. This approach breaks the multi-channel image into slices made up of one or several contiguous spectral bands.

In this work, we adopt a hybrid strategy, in which the data is partitioned into blocks made up of spatially adjacent pixel vectors which retain the full spectral content associated to them. This approach has several advantages [13]. First and foremost, the application of a hybrid partitioning provides a *natural* approach for low-level image processing, as it generally involves a kernel which is repeatedly applied to small set of neighboring pixels within the image data structure [18]. A second major reason has to do with the cost of interprocessor communication.

In order to balance the load of the processors in the heterogeneous environment, each processor should execute an amount of work that is proportional to its speed. Therefore, two major goals of our partitioning algorithm are: i) to obtain an appropriate set of workload fractions $\{\alpha_i\}_{i=1}^{P}$ that best fit the heterogeneous environment, and ii) to translate the chosen set of values into a suitable decomposition of the input data, taking into account the properties of the heterogeneous system. To accomplish the above goals, we use a workload estimation algorithm (WEA) that assumes that the workload of each processor $p_i$ must be directly proportional to its local memory and inversely proportional to its cycle-time $w_i$. The algorithm performs the following operations:

1. Obtain necessary information about the heterogeneous system, including the number of available processors $P$, each processor's identification number $\{p_i\}_{i=1}^{P}$, and processor cycle-times $\{\alpha_i\}_{i=1}^{P}$.

2. Set $\alpha_i = \lfloor \frac{(P/w_i)}{\sum_{i=1}^{P}(1/w_i)} \rfloor$ for all $i \in \{1, \cdots, P\}$. In other words, this step first approximates the $\{\alpha_i\}_{i=1}^{P}$ so that the amount of work assigned to each processor is proportional to its speed and $\alpha_i \cdot w_i \approx const$ for all processors.

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

4. Once the set $\{\alpha_i\}_{i=1}^{P}$ has been obtained, a further objective is to produce $P$ partitions of the input hyperspectral data set. To do so, we proceed as follows:

   - Obtain a first partitioning of the hyperspectral data set so that the number of rows in each partition is proportional to the values of $\{\alpha_i\}_{i=1}^{P}$.
   - Refine the initial partitioning taking into account the local memory associated to each processor.

## 3. Parallel target detection algorithms

Four parallel target detection algorithms are described in this section: parallel automated generation process (P-ATGP), parallel unsupervised fully constrained least squares (P-UFCLS) algorithm, parallel iterative error analysis (P-IEA) algorithm, and a parallel anomaly detector (P-RXD). The inputs to all discussed algorithms are a hyperspectral image cube $\mathbf{F}$ with $N$ dimensions, where $\mathbf{F}(x, y)$ denotes the pixel vector at spatial coordinates $(x, y)$, and a maximum number of targets to be detected, $t$. The output in all cases is a set of target pixels $\{\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \cdots, \mathbf{t}^{(t)}\}$.

### 3.1 P-ATGP

The ATGP algorithm [17] finds a set of spectrally distinct target pixels vectors using orthogonal subspace projections

in the spectral domain. Below, we provide a step-by-step description of our parallel version of this algorithm, specifically adapted to heterogeneous environments:

1. Using the WEA algorithm, the master divides the original image cube $\mathbf{F}$ into $P$ heterogeneous partitions. Then, the master sends the partitions to the workers.

2. Each worker finds the brightest pixel in its local partition using $\mathbf{t}_i^{(1)} = arg\{max_{(x,y)}\mathbf{F}(x,y)^T \cdot \mathbf{F}(x,y)\}$, where the superscript $T$ denotes the vector transpose operation and $i = 1, 2, \cdots, P$. Each worker then sends the spatial locations of the pixel identified as the brightest ones in its local partition back to the master.

3. Once all the workers have completed their parts, the master finds the brightest pixel of the input scene, $\mathbf{t}^{(1)}$, by applying the $argmax$ operator in step 2 to all the pixels at the spatial locations provided by the workers, and selecting the one that results in the maximum score. Then, the master sets $\mathbf{U} = \mathbf{t}^{(1)}$ and broadcasts this matrix to all workers.

4. Each worker finds (in parallel) the pixel in its local partition with the maximum orthogonal projection relative to the pixel vectors in $\mathbf{U}$, using a projector given by $P_{\mathbf{U}}^{\perp} = \mathbf{I} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T$, where $\mathbf{U}$ is the identity matrix. The orthogonal space projector $P_{\mathbf{U}}^{\perp}$ is now applied to all pixel vectors in each local partition to obtain $t_i^{(2)} = argmax_{(x,y)}\{(P_{\mathbf{U}}^{\perp}\mathbf{F}(x,y))^T(P_{\mathbf{U}}^{\perp}\mathbf{F}(x,y))\}$. Each worker then sends the spatial location of the resulting local pixels to the master node.

5. The master now finds a second target pixel, $\mathbf{t}^{(2)}$, by applying $P_{\mathbf{U}}^{\perp}$to the pixel vectors at the spatial locations provided by the workers, and selecting the one which results in the maximum score. The master now sets $\mathbf{U} = \{\mathbf{t}^{(1)}, \mathbf{t}^{(2)}\}$ and broadcasts this matrix to all workers.

6. Repeat from step 4 until a set of $t$ target pixels, $\{\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \cdots, \mathbf{t}^{(t)}\}$, are extracted from the input data.

### 3.2 P-UFCLS

The UFCLS algorithm [3] generates a set of $t$ targets using the concept of least square-based error minimization. Our parallel version of UFCLS (called P-UFCLS) assumes that $P$ processors are available and uses a master-slave implementation, in which a master processor coordinates the activities carried out by $P$ worker processors. The algorithm uses the following steps:

1. Execute steps 1-3 of the P-ATGP algorithm to obtain $\mathbf{t}^{(1)}$, the brightest pixel of the input scene, and broadcast $\mathbf{U} = \mathbf{t}^{(1)}$ to all the workers.

2. Each worker forms a local error image $E^{(i)}$ by calculating the least squares-based error for each pixel vector in the input data represented in terms of a fully constrained linear mixture of all the spectra in $\mathbf{U}$.

3. Each worker then finds the pixel $\mathbf{F}(x,y)$ in the local partition with the largest associated score in the error image $E^{(i)}$. The spatial coordinates of this pixel (and its associated error score) are sent back to the master.

4. The master obtains a second target $\mathbf{t}^{(2)}$ by selecting the pixel vector with largest associated error score from the pixel vectors at the spatial locations provided by the workers and broadcasts $\mathbf{U} = \{\mathbf{t}^{(1)}, \mathbf{t}^{(2)}\}$ to the workers.

5. Repeat from step 4 to incorporate a new target pixel $\mathbf{t}^{(3)}, \mathbf{t}^{(4)}, \cdots, \mathbf{t}^{(t)}$ to $\mathbf{U}$ until a set of $t$ target pixels have been extracted.

### 3.3 P-IEA

The IEA algorithm [12] is similar to the UFCLS in the sense that both of them make use of least square-based error minimization to search for possible targets. While the P-UFCLS algorithm finds a pixel with the largest vector length to be used as its initial pixel to start the algorithm, the P-IEA calculates (in parallel) the sample mean vector for initialization. It should be noted that this choice does not necessarily have to be better than the brightest pixel used by P-UFCLS; it is simply a different initialization parameter. As a result, the only input parameter is the number of targets to be extracted, and the spectral signatures returned by the algorithm correspond to real pixel vectors in the data cube.

### 3.4 P-RXD

Finally, we provide a parallel version of an anomaly detection algorithm originally proposed by Reed and Xiaoli [16] and referred to as RXD, which has also been widely used for target detection purposes. It finds target pixels which are spectrally distinct from their neighboring pixels. Our parallel algorithm is given by the following steps:

1. Divide the original image cube $\mathbf{F}$ into $P$ partitions using the WEA algorithm.

2. The master calculates the N-dimensional mean vector $\mathbf{m}$ concurrently, where each component is the average of the pixel values of each spectral band of the unique set. This vector is formed once all the processors finish their parts. At the same time, the master also calculates the sample spectral covariance matrix $\mathbf{K}$ concurrently as the average of all the individual matrices produced by the workers using their respective portions.

3. Using the above information, each worker applies (locally) a so-called RXD filter given by the well-known Mahalanobis distance [3] to all the pixel vectors in the local partition as follows: $\delta^{(RXD)}(\mathbf{F}(x,y)) = (\mathbf{F}(x,y)-\mathbf{m})^T\mathbf{K}^{-1}(\mathbf{F}(x,y)-\mathbf{m})$.

4. The master now selects the $t$ pixel vectors with higher associated value of $\delta^{(RXD)}$, and uses them to form a final set of targets $\{\mathbf{t}^{(1)},\mathbf{t}^{(2)},\cdots,\mathbf{t}^{(t)}\}$.

# 4. Experimental results

## 4.1 Parallel computing architectures

The parallel computing architectures used in this study comprise four NOWs distributed among different locations at University of Maryland and the Thunderhead Beowulf cluster at NASA's Goddard Space Flight Center (see http://thunderhead.gsfc.nasa.gov). The NOWs were custom-designed in order to approximate a recently proposed framework for evaluation of heterogeneous parallel algorithms [10], which relies on the assumption that a heterogeneous algorithm cannot be executed on a heterogeneous network faster than its homogeneous version on the equivalent homogeneous network. In [10], a homogeneous computing environment is considered equivalent to the heterogeneous one in light of the three following principles:

1. Both environments should have exactly the same number of processors.

2. The speed of each processor in the homogeneous environment should be equal to the average speed of processors in the heterogeneous environment.

3. The aggregate communication characteristics of the homogeneous environment should be the same as those of the heterogeneous environment.

With the above three principles in mind, a heterogeneous algorithm may be considered optimal if its efficiency on a heterogeneous network is the same as that evidenced by its homogeneous version on the equivalent homogeneous network. This allows using the parallel performance achieved by the homogeneous version as a benchmark for assessing the parallel efficiency of the heterogeneous algorithm. The four considered networks are considered approximately equivalent under the above framework. Their description follows:

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

**Table 2. Capacity of communication links (in time in milliseconds to transfer a one-megabit message).**

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

- *Fully homogeneous network*. Consists of 16 identical Linux workstations with processor cycle-time of $w=0.0131$ seconds per megaflop, interconnected via a homogeneous communication network where the capacity of links is $c=26.64$ milliseconds.

- *Partially heterogeneous network*. Formed by the set of 16 heterogeneous workstations in Table 1 but interconnected using the same homogeneous communication network with capacity $c=26.64$ milliseconds.

- *Partially homogeneous network*. Formed by 16 identical Linux workstations with cycle-time of $w=0.0131$ seconds per megaflop, but interconnected using the heterogeneous network shown in Table 2.

In order to test the proposed algorithms on a larger-scale parallel platform, we have also experimented with Thunderhead, a 256-node Beowulf cluster at NASA's Goddard Space Flight Center. This system is composed of 256 dual 2.4 GHz Intel Xeon nodes, each with 1 GB of main memory, 80 GB of disk space and 512 KB of cache, interconnected via 2 GHz optical fibre Myrinet. The total peak performance of the system is 2457.6 Gflops. The operating

**Table 1. Specifications of heterogeneous processors.**

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

system used at the time of measurements was Linux RedHat 8.0, and MPICH was the message-passing library used.

### 4.2 Hyperspectral data

The image scene used for experiments in this work was collected by the AVIRIS instrument, which was flown by NASA's Jet Propulsion Laboratory over the World Trade Center (WTC) area in New York City on September 16, 2001, just five days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex. The full data set selected for experiments consists of 2133x512 pixels, 224 spectral bands and a total size of (approximately) 1 GB. The spatial resolution is 1.7 meters per pixel. Fig. 1 shows a false color composite of the data set selected for experiments using the 1682, 1107 and 655 nm channels, displayed as red, green and blue, respectively. Vegetated areas appear green in Fig. 1, while burned areas appear dark gray. Smoke coming from the WTC area (in the red rectangle) and going down to south Manhattan appears bright blue due to high spectral reflectance in the 655 nm channel.

In this work, we use a U.S. Geological Survey thermal map (available online: http://pubs.usgs.gov/of/2001/ofr-01-0429/hotspot.key.tgif.gif) which shows the target locations of the thermal hot spots at the WTC area, displayed as bright red, orange and yellow spots in Fig. 1. The map is centered at the region where the towers collapsed, and the temperatures range from 700F to 1300F. This thermal map will be used in this work as ground-truth to validate the target detection accuracy of the proposed parallel algorithms.

### 4.3 Performance evaluation

Before empirically investigating the parallel performance of the proposed heterogeneous parallel algorithms, we first evaluate their target detection accuracy in the context of the considered application. Table 3 shows the spectral angle distance (SAD) [3] between the most similar target pixels detected by P-ATGP, P-UFCLS, P-IEA and P-RXD and the pixel vectors at the known target positions (labeled from 'A' to 'H' in Fig. 1). In all cases, the number of target pixels to be detected, $t$, was set to 18 after calculating the intrinsic dimensionality of the data. It is worth noting that the SAD between two pixel vectors at different

**Table 3. Spectral similarity between target pixels and known ground targets.**

| Hot spot | P-ATGP (1263) | P-UFCLS (916) | P-IEA (948) | P-RXD (1392) |
|---|---|---|---|---|
| 'A' | 0.002 | 0.123 | 0.126 | 0.004 |
| 'B' | 0.001 | 0.005 | 0.005 | 0.008 |
| 'C' | 0.005 | 0.012 | 0.021 | 0.003 |
| 'D' | 0.003 | 0.002 | 0.005 | 0.012 |
| 'E' | 0.008 | 0.026 | 0.031 | 0.021 |
| 'F' | 0.001 | 0.169 | 0.169 | 0.008 |
| 'G' | 0.001 | 0.001 | 0.009 | 0.011 |

spatial locations, say $\mathbf{F}(x, y)$ and $\mathbf{F}(x', y')$, was calculated using the following expression: $\text{SAD}(\mathbf{F}(x, y), \mathbf{F}(x', y')) = cos^{-1}[(\mathbf{F}(x, y) \cdot \mathbf{F}(x', y'))/(\|\mathbf{F}(x, y)\| \cdot \|\mathbf{F}(x', y')\|)$. The lower the SAD score, the more similar the two pixel vectors are. As shown by Table 3, the P-ATGP extracted targets were almost identical, spectrally, to the known ground-truth targets. A similar comment applies to P-RXD, although some of the targets extracted by this algorithm showed less spectral similarity with regards to the ground-truth in Fig. 1. Finally, both the P-UFCLS and P-IEA could not accurately detect several target pixels, including the one labeled as 'F' which corresponds to a thermal hot spot with high (700F) temperature. For illustrative purposes, Table 3 also gives processing times in seconds for the sequential versions (implemented using the GNU-C/C++ compiler in its 4.0 version) of the algorithms above on a single processor of the Thunderhead Beowulf cluster. The table reveals that P-ATGP was slightly faster than P-UFCLS and P-IEA on the considered mono-processor environment, while the P-RXD was the most computationally expensive algorithm, mainly due to covariance matrix calculations. In all cases, processing times were quite significant, with more than fifteen minutes of computation for the considered problem size.

In order to improve computational performance, we tested the parallel heterogeneous versions above on the five considered parallel platforms. The algorithms were implemented using C++ with calls to message passing interface (MPI). 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. To investigate their parallel properties, the algorithms were first tested by timing the parallel versions on the four considered NOWs. Table 4 shows the measured execution


**Figure 1. AVIRIS hyperspectral (left) and location of fires in the World Trade Center area (right).**

times for the proposed parallel heterogeneous algorithms and their respective homogeneous versions. It is important to emphasize that the homogeneous versions were obtained by simply replacing step 3 of the WEA algorithm with $\alpha_i = P/W$ for all $i \in \{1, 2, \cdots, P\}$. In the table, we denote the homogeneous versions by using 'H-' instead of 'P-' to distinguish them from the standard, parallel heterogeneous versions.

As expected, the execution times reported on Table 4 show that the heterogeneous algorithms were able to adapt much better to fully (or partially) heterogeneous environments than the homogeneous versions, which only performed satisfactorily on the fully homogeneous network. One can see that the heterogeneous algorithms were always several times faster than their homogeneous counterparts in the fully heterogeneous NOW, and also in both the partially homogeneous and the partially heterogeneous networks. On the other hand, the homogeneous algorithms only slightly outperformed their heterogeneous counterparts in the fully homogeneous NOW.

Table 4 also indicates that the performance of the heterogeneous algorithms on the fully heterogeneous platform was almost the same as that evidenced by the equivalent homogeneous algorithms on the fully homogeneous NOW. This indicated that the proposed heterogeneous algorithms were always close to the optimal heterogeneous modification of the basic homogeneous ones. On the other hand, the homogeneous algorithms performed much better on the partially homogeneous network (made up of processors with the same cycle-times) than on the partially heterogeneous network. This fact reveals that processor heterogeneity has a more significant impact on algorithm performance than network heterogeneity, a fact that is not surprising given our adopted strategy for data partitioning in the design of parallel heterogeneous algorithms. Finally, Table 4 shows that the homogeneous versions only slightly outperformed the heterogeneous algorithms in the fully homogeneous NOW. This clearly demonstrates the flexibility of the proposed heterogeneous algorithms, which were able to adapt efficiently to the four considered environments.

To further explore the parallel properties of the considered algorithms in more detail, an in-depth analysis of computation and communication times achieved by the different methods is also highly desirable. For that purpose, Table 5 shows the total time spent by the tested algorithms in communications and computations in the four considered networks, where two types of computation times were analyzed, namely, sequential (those performed by the root node with no other parallel tasks active in the system, labeled as SEQ in the table) and parallel (the rest of computations, i.e., those performed by the root node and/or the workers in parallel, labeled as PAR in the table). The latter includes the times in which the workers remain idle.

It can be seen from Table 5 that, among all considered heterogeneous parallel algorithms, SEQ scores were particularly significant for the P-UFCLS and P-IEA algorithms. This is mainly due to the fact that these algorithms involve several steps based on sequential computations. SEQ scores were also relevant for the P-ATGP and P-RXD, which involve several gather/scatter operations followed by compute-intensive orthogonal space projections and covariance computations at the master, which need to be completed in sequential fashion before a new parallel operation can be accomplished by the workers. Finally, it can also be seen from Table 5 that the cost of parallel (PAR) computations dominated that of communications (COM) in all the considered parallel algorithms. In particular, the ratio of PAR to COM scores achieved by the homogeneous algorithms executed on the (fully or partially) heterogeneous network was very high, but this is mainly due to a less efficient workload distribution among the heterogeneous workers. Therefore, a study of load balance is highly required to

**Table 4. Execution times (seconds) of heterogeneous algorithms and their homogeneous versions.**

| Algorithm | Fully heterogeneous | Fully homogeneous | Partially heterogeneous | Partially homogeneous |
|---|---|---|---|---|
| P-ATGP | 84 | 89 | 87 | 88 |
| H-ATGP | 667 | 81 | 638 | 374 |
| P-UFCLS | 51 | 56 | 55 | 56 |
| H-UFCLS | 506 | 50 | 497 | 253 |
| P-IEA | 54 | 59 | 60 | 63 |
| H-IEA | 524 | 51 | 538 | 263 |
| P-RXD | 104 | 112 | 107 | 109 |
| H-RXD | 724 | 101 | 702 | 421 |

fully substantiate the parallel properties of the considered algorithms.

To analyze the issue of load balance in more detail, Table 6 shows the *imbalance* scores achieved by the parallel algorithms on the four considered networks. 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. Therefore, perfect balance is achieved when $D = 1$. In the table, we display the imbalance considering all processors, $D_{all}$, and also considering all processors but the root, $D_{minus}$. As we can see from Table 6, only the P-ATGP was able to provide values of $D_{all}$ close to 1 in all considered networks, with P-RXD being able to produce results which are also relatively close to 1. Further, the P-ATGP provided almost the same results for both $D_{all}$ and $D_{minus}$ while, for the other tested methods, load balance was generally better when the root processor was not included. While the homogeneous algorithms executed on the (fully or partially) heterogeneous networks provided the highest values of $D_{all}$ and $D_{minus}$ (and hence the highest imbalance), the heterogeneous algorithms executed on the homogeneous network resulted in values of $D_{minus}$ which were close to 1. It is our belief that the (relatively high) unbalance scores measured for the P-UFCLS and P-IEA algorithms, in particular, when these are executed on the fully heterogeneous network, are not due to memory considerations or to an inefficient allocation of data chunks to heterogeneous resources, but to the impact of communications. Our future research will include considerations about the heterogeneous communication network in the data partitioning algorithm [14].

Taking into account the results presented above, and with the ultimate goal of exploring issues of scalability, we have also compared the performance of the parallel target detection algorithms on NASA's Thunderhead Beowulf cluster. For that purpose, Fig. 2 plots the speedups achieved by multi-processor runs of the heterogeneous parallel algorithms over their corresponding single-processor runs on Thunderhead. It can be seen from Fig. 2 that P-ATGP scaled significantly better than both P-UFCLS and P-IEA, and only slightly better than P-RXD. This has to do with the high number of sequential computations involved in both P-UFCLS and P-IEA. For instance, using 256 processors, all algorithms were able to complete their calculations in only 6 or 7 seconds. The above results represent significant improvements over the single-processor versions of the same algorithms, which can take up to several minutes of computation for the considered problem.

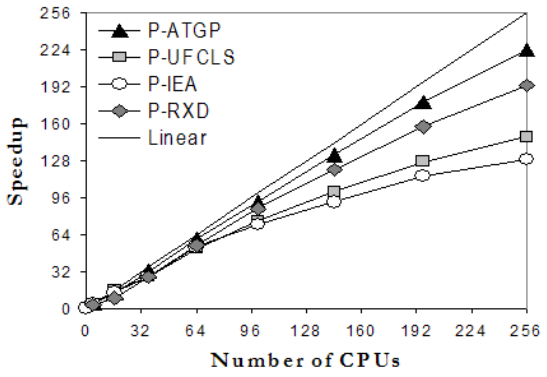


**Figure 2. Scalability of heterogeneous parallel algorithms on Thunderhead.**

## 5. Conclusions

This paper described several innovative parallel algorithms for target detection in hyperspectral data sets. As a case study of specific issues involved in the exploitation of heterogeneous computing systems for remote sensing applications, we provided a detailed discussion on the effects that platform heterogeneity has on degrading parallel performance of target detection algorithms. The evaluation strategy conducted in this work was based on experimentally assessing heterogeneous algorithms by comparing their efficiency on (fully or partially) heterogeneous networks of workstations with the efficiency achieved by their homogeneous versions on equally powerful homogeneous networks. Experimental results indicate that heterogeneous networks represent a source of computational power that is both accessible and applicable to obtaining results quickly enough and with high reliability in many on-going and planned Earth observing and planetary exploration missions.

**Table 5. Communication (COM), sequential computation (SEQ) and parallel computation (PAR) times.**

| | Fully heterogeneous | | | Fully homogeneous | | | Partially heterogeneous | | | Partially homogeneous | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | COM | SEQ | PAR | COM | SEQ | PAR | COM | SEQ | PAR | COM | SEQ | PAR |
| P-ATGP | 7 | 19 | 58 | 11 | 16 | 62 | 8 | 18 | 61 | 8 | 20 | 60 |
| H-ATGP | 14 | 19 | 634 | 6 | 16 | 59 | 9 | 18 | 611 | 12 | 20 | 342 |
| P-UFCLS | 4 | 27 | 20 | 7 | 24 | 25 | 6 | 27 | 22 | 8 | 26 | 22 |
| H-UFCLS | 9 | 27 | 470 | 3 | 24 | 23 | 5 | 27 | 465 | 13 | 26 | 214 |
| P-IEA | 9 | 33 | 12 | 13 | 28 | 18 | 10 | 29 | 21 | 10 | 28 | 25 |
| H-IEA | 17 | 33 | 474 | 7 | 28 | 16 | 9 | 29 | 500 | 11 | 28 | 226 |
| P-RXD | 9 | 18 | 77 | 13 | 19 | 90 | 10 | 21 | 76 | 10 | 22 | 77 |
| H-RXD | 17 | 22 | 685 | 7 | 18 | 76 | 9 | 19 | 674 | 13 | 23 | 485 |

**Table 6. Load balancing rates for the heterogeneous algorithms and their homogeneous versions.**

| | Fully heterogeneous | | Fully homogeneous | | Partially heterogeneous | | Partially homogeneous | |
|---|---|---|---|---|---|---|---|---|
| | $D_{all}$ | $D_{minus}$ | $D_{all}$ | $D_{minus}$ | $D_{all}$ | $D_{minus}$ | $D_{all}$ | $D_{minus}$ |
| P-ATGP | 1.19 | 1.05 | 1.16 | 1.03 | 1.24 | 1.06 | 1.22 | 1.03 |
| H-ATGP | 1.62 | 1.23 | 1.20 | 1.06 | 1.67 | 1.26 | 1.41 | 1.05 |
| P-UFCLS | 1.49 | 1.06 | 1.51 | 1.05 | 1.69 | 1.06 | 1.54 | 1.08 |
| H-UFCLS | 1.68 | 1.25 | 1.54 | 1.11 | 1.75 | 1.34 | 1.77 | 1.09 |
| P-IEA | 1.69 | 1.08 | 1.54 | 1.07 | 1.77 | 1.08 | 1.59 | 1.10 |
| H-IEA | 1.82 | 1.29 | 1.58 | 1.15 | 1.86 | 1.41 | 1.82 | 1.11 |
| P-RXD | 1.20 | 1.05 | 1.17 | 1.04 | 1.25 | 1.06 | 1.23 | 1.05 |
| H-RXD | 1.64 | 1.25 | 1.19 | 1.07 | 1.65 | 1.29 | 1.44 | 1.06 |

## References


[1] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix multiplication on heterogeneous platforms. *IEEE Trans. Parallel and Distributed Systems*, 12:1033–1055, 2001.

[2] H. Casanova, M. Thomason, and J. Dongarra. Stochastic performance prediction for iterative algorithms in distributed environments. *Journal of Parallel and Distributed Computing*, 58:68–91, 1999.

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

[4] J. Dorband, J. Palencia, and U. Ranawake. Commodity clusters at Goddard Space Flight Center. *Journal of Space Communication*, 3:227–248, 2003.

[5] T. El-Ghazawi, S. Kaewpijit, and J. L. Moigne. Parallel and adaptive reduction of hyperspectral data to intrinsic dimensionality. *Cluster Computing*, pages 102–110, 2001.

[6] R. O. Green. Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS). *Remote Sensing of Environment*, 65:227–248, 1998.

[7] K. Itoh. Massively parallel fourier-transform spectral imaging and hyperspectral image processing. *Optics and Laser Technology*, 25:202, 1993.

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

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

[10] A. Lastovetsky and R. Reddy. On performance analysis of heterogeneous parallel algorithms. *Parallel Computing*, 30:1195–1216, 2004.

[11] A. Legrand, H. Renard, Y. Robert, and F. Vivien. Mapping and load-balancing iterative computations on heterogeneous clusters with shared links. *IEEE Trans. Parallel and Distributed Systems*, 15:549–558, 2004.

[12] R. A. Neville, K. Staenz, T. Szeredi, J. Lefebvre, and P. Hauff. Automatic endmember extraction from hyperspectral data for mineral exploration. In *21st Canadian Symposium on Remote Sensing*, pages 401–415, 1999.

[13] A. Plaza, J. Plaza, and D. Valencia. AMEEPAR: Parallel morphological algorithm for hyperspectral image classification in heterogeneous networks of workstations. *Lecture Notes in Computer Science*, 3993(3):24–31, 2006.

[14] A. Plaza, J. Plaza, and D. Valencia. Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data. *Journal of Supercomputing*, Accepted for publication, in press, 2007.

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

[16] I. Reed and X. Yu. Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution. *IEEE Trans. Acoustics*, 10:1760–1770, 1990.

[17] H. Ren and C.-I. Chang. Automatic spectral target recognition in hyperspectral imagery. *IEEE Trans. Aerospace and Electronic Systems*, 39:1232–1249, 2003.

[18] F. J. Seinstra, D. Koelma, and J. M. Geusebroek. A software architecture for user transparent parallel image processing. *Parallel Computing*, 28:967–993, 2002.

[19] S. Tehranian, Y. Zhao, T. Harvey, A. Swaroop, and K. Mckenzie. A robust framework for real-time distributed processing of satellite data. *Journal of Parallel and Distributed Computing*, 66:403–418, 2006.

[20] T. Yang and C. Fu. Heuristic algorithms for scheduling iterative task computations on distributed memory machines. *IEEE Trans. Parallel and Distributed Systems*, 8:608–622, 1997.