# Design and Implementation of a Parallel Heterogeneous Algorithm for Hyperspectral Image Analysis Using HeteroMPI

David Valencia[1], Alexey Lastovetsky[2], Antonio Plaza[1]

[1]Department of Computer Science
University of Extremadura
E-10071 Caceres, Spain
{davaleco, aplaza}@unex.es

[2]School of Computer Science & Informatics
University College Dublin,
Belfield, Dublin 4, Ireland
alexey.lastovetsky@ucd.ie

## Abstract

*The development of efficient techniques for transforming the massive volume of remotely sensed hyperspectral data collected on a daily basis into scientific understanding is critical for space-based Earth science and planetary exploration. Although most available parallel processing strategies for hyperspectral image analysis assume homogeneity in the computing platform, heterogeneous networks of computers represent a promising cost-effective solution expected to play a major role in many on-going and planned remote sensing missions. To address the need for cost-effective parallel hyperspectral imaging algorithms, this paper develops an innovative heterogeneous parallel algorithm for spatial/spectral morphological analysis of hyperspectral image data. The algorithm has been developed using Heterogeneous MPI (HeteroMPI), an extension of MPI for programming high-performance computations on heterogeneous networks of computers. Experimental results are presented and discussed in the context of a realistic application, based on hyperspectral data collected by NASA's Jet Propulsion Laboratory.*

## 1. Introduction

Hyperspectral imaging identifies materials and objects in the air, land and water on the basis of the unique reflectance patterns that result from the interaction of solar energy with the molecular structure of the material [1]. Most applications of this technology require timely responses for swift decisions which depend upon high computing performance of algorithm analysis. Examples include target detection for military and defense/security deployment, urban planning and management, risk/hazard prevention and response including wild-land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination. The concept of hyperspectral imaging was introduced when NASA's Jet Propulsion Laboratory developed the Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) system, which covers the wavelength region from 0.4 to 2.5 μm using 224 spectral channels (see Fig. 1). This imager is able to continuously produce snapshot image cubes of tens or even hundreds of kilometers long, each of them with hundreds of MB in size, and this explosion in the amount of collected information has rapidly introduced new processing challenges [2].

Although most dedicated parallel machines for remote sensing data analysis employed by NASA and other institutions during the last decade have been chiefly homogeneous in nature [3], computing on heterogeneous networks of computers (HNOCs) has soon become a viable alternative to expensive parallel computing systems [4]. These networks enable the use of existing resources and provide incremental scalability of hardware components with performance isolation. At the same time, HNOCs can achieve high communication speed at low cost, using switch-based networks such as ATMs, as well as distributed service and support, especially for large file systems.

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

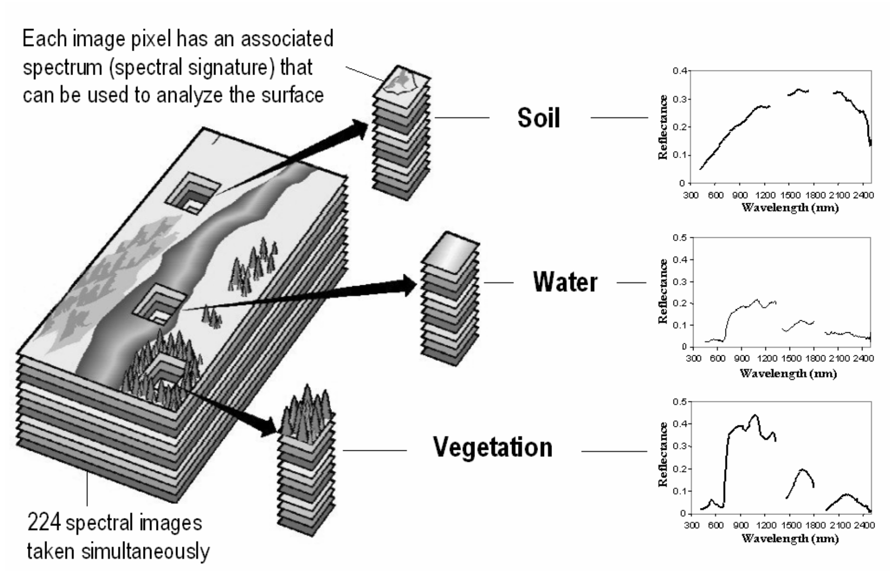
Each image pixel has an associated spectrum (spectral signature) that can be used to analyze the surface
Soil
Water
Vegetation
224 spectral images taken simultaneously
Reflectance
Wavelength (nm)

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

Although the standard MPI [5] has been widely used to implement parallel algorithms for HNOCs in the past, it does not provide specific means to address some additional challenges posed by these networks, including the distribution of computations and communications unevenly, taking into account the computing power of the heterogeneous processors and the bandwidth of the communications links. To achieve the above goals, HeteroMPI was developed as an extension of MPI which allows the programmer to describe the performance model of a parallel algorithm in generic fashion [6]. This is a highly desirable feature in hyperspectral imaging applications, in which the main features of the underlying parallel algorithm have an essential impact on execution performance.

The paper is structured as follows. Section 2 outlines the main features of HeteroMPI. Section 3 develops a HeteroMPI-based parallel algorithm for joint spatial/spectral analysis of hyperspectral imagery. Section 4 assesses the performance of the algorithm by analyzing its accuracy and parallel properties on a heterogeneous cluster made up of 15 processors. Finally, section 5 concludes with some remarks and hints at plausible future research.

## 2. Outline of HeteroMPI

The standard MPI specification provides communication and group constructors which allow the application programmer to create a group of processes explicitly chosen from an ordered set [5]. This approach is feasible when the application is run on a homogeneous distributed-memory computer system. However, selection of a group for execution on HNOCs must take into account the computing power of the heterogeneous processors and the speed/bandwidth of communication links between each processor pair [6]. This feature is of particular importance in applications dominated by large data volumes such as hyperspectral image analysis, but is also quite difficult to accomplish from the viewpoint of the programmer.

The main idea of HeteroMPI is to automate and optimize the selection of a group of processes that executes a heterogeneous algorithm faster than any other possible group. For this purpose, HeteroMPI provides a small and dedicated definition language for the specification of such performance model. This language is a subset of mpC, defined in [7], and allows the programmer to explicitly define an *abstract* network and distribute data, computations and communications over the network. Then, HeteroMPI automatically maps (at run time) the abstract network to a *real* execution network by dynamically adapting the performance model to specific network parameters such as the computing power of processors or the capacities of communication links in the real environment. By means of a compiler, the description of a performance model is translated into a set of functions that make up an algorithm-specific part of HeteroMPI runtime system. Below, we provide a brief outline of the most important HeteroMPI functions which have been used to implement the proposed parallel algorithm. Detailed information about these and other HeteroMPI functions is available in [6].

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

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

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

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

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

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

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

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

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

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

where **gid** is the HeteroMPI handle to the group of MPI processes. Again, this is a collective operation that must be called by all members of this group. In order to finalize the runtime system, the following operation is used:

```
HeteroMPI_Finalize(int exitcode)
```

## 3. Parallel hyperspectral algorithm

This section describes a parallel heterogeneous algorithm for automated morphological analysis of hyperspectral image data. Mathematical morphology is a standard image processing technique that provides a remarkable framework to achieve the desired integration of spatial and spectral responses [8]. First, we describe the standard morphological algorithm. Then, we outline important aspects about its parallel implementation such as data partitioning and communication issues. Finally, we provide a HeteroMPI-based implementation for HNOCs. Performance data are given in the following section.

### 3.1. Morphological algorithm

Morphological analysis has been successfully used in previous research to analyze hyperspectral data sets [8]. The morphological algorithm selected in this work as a representative case study takes into account both the spatial and spectral information of the data in simultaneous fashion. Such spatial/spectral, hybrid techniques represent the most advanced generation of hyperspectral imaging algorithms currently available.

Before describing our proposed approach, let us denote by $\boldsymbol{f}$ a hyperspectral data set defined on an N-dimensional (N-D) space, where N is the number of channels or spectral bands. The main idea of the algorithm is to impose an ordering relation in terms of spectral purity in the set of pixel vectors lying within a spatial search window or *structuring element* (SE) around each image pixel vector [8]. To do so, we first define 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 a SE denoted by $B$ ($B$-neighborhood) as follows:

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

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

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

Based on the distance above, we calculate the extended morphological erosion of $\boldsymbol{f}$ by $B$ [8] for each pixel in the input data scene as follows:

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

where the argmin operator selects the pixel vector is most highly similar, spectrally, to all the other pixels in the $B$-neighborhood.

On the other hand, the extended morphological dilation of $\boldsymbol{f}$ by $B$ [8] is calculated as follows:

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

With the above definitions in mind, we provide below an unsupervised classification algorithm for hyperspectral imagery based on extended morphological operations:

*Automated Morphological Classification (AMC)*

*Inputs*: Data cube: $\boldsymbol{f}$; morphological SE: $B$; Number of classes: $c$; Number of iterations: $I_{max}$.

*Output*: 2-D matrix which contains a classification label for each pixel vector $\boldsymbol{f}(x,y)$ in the input image.

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

As shown in previous work [2], computational complexity is $O(p_{\boldsymbol{f}}\times p_B\times I_{\max}\times N)$, where $p_{\boldsymbol{f}}$ is the number of pixels in $\boldsymbol{f}$ and $p_B$ is the number of pixels in $B$. However, an adequate parallelization strategy can greatly enhance the computational performance of the proposed algorithm, as will be outlined in the following subsection.

## 3.2. Data partitioning

Two types of parallelism can be exploited in hyperspectral image analysis algorithms: spatial-domain parallelism and spectral-domain parallelism. Spatial-domain parallelism subdivides the image into multiple blocks made up of entire pixel vectors, and assigns one or more blocks to each processor. Spectral-domain parallelism subdivides the hyperspectral data into blocks made up of contiguous spectral bands (sub-volumes), and assigns one or more sub-volumes to each processor. The latter approach breaks the spectral identity of the data because each pixel vector is split amongst several processing units, and operations such as morphological erosion and dilation would need to originate from several processors, thus requiring intensive inter-processor communication. In this work, we use spatial-domain parallelism in order to preserve the entire spectral information of each image pixel. This is a natural approach for low-level image processing, as many operations require the same function to be applied to a small set of elements around each data element present in the image data structure.

With the above ideas in mind, the main goal of our parallelization framework for the AMC algorithm is to use a low-level image processing-oriented approach, in which each heterogeneous processor will be able to process a spatial/spectral data partition *locally*. In previous work, we have defined the concept of parallelizable spatial/spectral partition (PSSP) as a hyperspectral data partition that can be processed independently without communication [2]. Here, we use the concept of PSSP above to define a virtual processor grid organization in which processors apply the AMC algorithm locally to each partition, thus producing a set of *local* classification outputs which are then combined to form a *global* classification output.

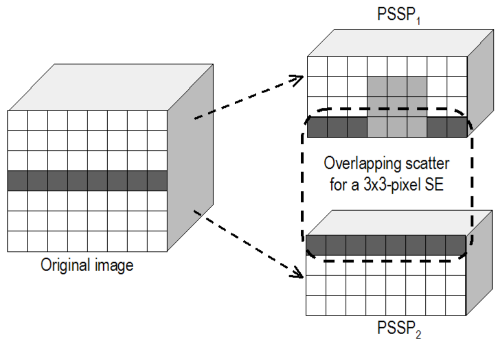

**Fig. 2. Overlapping scatter to avoid inter-processor communication in the processing of two data partitions.**

In order to adequately exploit the concept of PSSP, a function to update overlapping parts of partial data structures has been implemented in order to alleviate inter-processor communication when the SE computation is split amongst several different processors. In order to eliminate such overhead, a *scratch border* is placed around each PSSP in the virtual grid to reduce inter-processor communication (see Fig. 2, which gives a simplified view using two adjacent, homogeneous partitions). To avoid introducing a significant amount of redundant information resulting from scratch borders, we limit the $B$-neighborhood function to a 3x3-pixel structuring element, and increase the number of algorithm iterations ($I_{max}$) to obtain a better spatial/spectral description of features in the hyperspectral data [8]. The main challenge of this approach is to find an optimal mapping of PSSPs on the virtual grid of processors, i.e., the size of the resulting partitions (including scratch borders) must be in accordance with the computing power of heterogeneous processors. This is accomplished through the definition of a performance model, as explained below.

### 3.3. HeteroMPI-based parallel implementation

In order to implement the parallel morphological algorithm outlined above using HeteroMPI, the first step is to define a performance model able to capture the data partitioning and communication framework described in the previous subsection. Fig. 3 shows the most important fragments of the mpC-based code that describes the adopted performance model, which has 6 input parameters. Parameter **m** specifies the number samples of the data cube, while parameter **n** specifies the number of lines. Parameters **se_size** and **iter** respectively denote the size of the SE and the number of iterations executed by the algorithm. Finally, parameters **p** and **q** indicate the dimensions of the computational grid (in columns and rows, respectively), which are used to map the spatial coordinates of the individual processors within the processor grid layout. Finally, **partition_size** is an array that indicates the size of the local PSSPs (calculated automatically using the computing power of the heterogeneous processors). Fig. 3 shows different communication links, defined based on the spatial localization of each processor within the grid. It should be noted that some of the definitions have been removed from Fig. 3 for simplicity. However, some of the most representative sections are included. Keyword **algorithm** begins the specification of the performance model of an algorithm followed by its name and a list of formal parameters. The **coord** section defines the mapping of individual abstract processors performing the algorithm onto the grid layout using variables **I** and **J**. The **node** primitive defines the amount of computations that will be made by every processor, which depends on its spatial coordinates in the grid as indicated by **I** and **J** and the computing power of the individual processors as indicated by **partition_size**. Finally, the **parent** directive indicates the spatial localization of the master processor in the grid. An additional **link** section is used to define the individual communications that every processor carries out based on its position in the grid. Further information on performance model definition is available in [7].

Once a performance model for the parallel algorithm has been defined, implementation using the standard HeteroMPI in Section 2 is quite straightforward [6]. Fig. 4 shows the most interesting fragments of the HeteroMPI-based code of our parallel implementation. The HeteroMPI runtime system is initialized using operation **HeteroMPI_Init**. Then, operation **HeteroMPI_Recon** updates the estimation of performances of processors. This is followed by the creation of a group of processes using operation **HeteroMPI_Group_create**. The members of this group then perform the computations of the heterogeneous parallel algorithm using standard MPI mechanisms. This is followed by freeing the group using operation **HeteroMPI_Group_free**, and the finalization of the HeteroMPI runtime system using operation **HeteroMPI_Finalize**. In this code, the benchmark function used to measure the processing power of the processors in **HeteroMPI_Recon** is essential, mainly because a poor estimation of the power and memory capacity of processors may result in load balancing problems. This issue will be addressed via experiments in the following section.

## 4. Experiments

This section evaluates the proposed parallel algorithm. First, a parallel heterogeneous cluster is described. Then, we briefly describe a real hyperspectral data set collected by the AVIRIS system that will be used in experiments. The section ends with a detailed evaluation of the accuracy and parallel performance of the proposed parallel algorithm.

### 4.1. Parallel heterogeneous cluster

A heterogeneous network of 11 Linux/SunOS workstations and a total of 15 processors at University College Dublin (UCD) was used in experiments. Table 1 shows the specifications of the heterogeneous processors, including their relative speeds. The processors in Table 1 are interconnected via 100 Mbit Ethernet communication network with a switch enabling parallel communications among the processors. Although this is a simple configuration, it is also a quite typical and realistic one as well. We measure the relative speeds in Table 1 with the core computation of the algorithm (processing a 3x3-pixel neighborhood using morphological operations).

### 4.2. Hyperspectral image data

Fig. 5 (left) shows the Indian Pines AVIRIS hyperspectral data set considered in experiments. It consists of 614 samples, 512 lines and 224 spectral bands (more than 140 MB).

```
algorithm hpamc_rend(int m, int n, int se_size, int iter, int p, int q, int partition_size[p*q]) {
coord I = p, J = q;
node { I>=0 && J>=0: benchmark*((partition_size[I*q+J]*iter); };
parent[0,0];
}
```

**Fig. 3. The core of the performance model for the parallel hyperspectral imaging algorithm defined in mpC.**

```
main(int argc, char *argv[]){
  HeteroMPI_Init(&argc,&argv);
  if (HeteroMPI_Is_member(HMPI_COMM_WORLD_GROUP)){
       HeteroMPI_Recon(benchmark_function, dims, 15, &output_p);
  }
  HeteroMPI_Group_create(&gid, &MPC_NetType_hpamc_rend, modelp, num_param);
  if (HeteroMPI_Is_free()){
       HeteroMPI_Group_create(&gid, &MPC_NetType_hpamc_rend, NULL, 0);
  }
  if (HeteroMPI_Is_free()){
       HeteroMPI_Finalize(0);
  }
  if (HeteroMPI_Is_member(&gid)){
     HeteroMPI_Group_performances(&gid, speeds);
     Read_image(name,image,lin,col,bands,data_type,init);
     for (i=imax; i>1; i=i--){
                AMC_algorithm(image,lin,col,bands,sizeofB,res);
     }
     if (HeteroMPI_Is_member(&gid)){
        free(image);
     }
     HeteroMPI_Group_free(&gid);
     HeteroMPI_Finalize(0);
   }
}
```

**Fig. 4. The core of the HeteroMPI program implementing the parallel hyperspectral analysis algorithm.**

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

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

As shown by Fig. 5 (left), the scene represents a very challenging classification problem. Extensive ground-truth information is available for the scene, as shown by Fig. 5 (right). This map is composed of 30 land cover classes that will be used to validate the classification accuracy of our parallel morphological classification algorithm. This scene is regarded as a universal benchmark to validate hyperspectral image analysis algorithms.

## 4.3. Assessment of the parallel algorithm

The parallel algorithm was applied to the AVIRIS Indian Pines scene using a fixed, 3x3-pixel SE and seven different values for parameter $I_{max}$, which defines the number of iterations executed by the algorithm (ranging from 1 to 7 in experiments). Table 2 shows the classification accuracies (percentage of correctly classified pixels) obtained using the seven considered numbers of iterations, along with the single-processor execution times (in minutes) measured in a Linux workstation with Intel XEON processor at 2 GHz, 1 GB of RAM and 512 KB of cache.

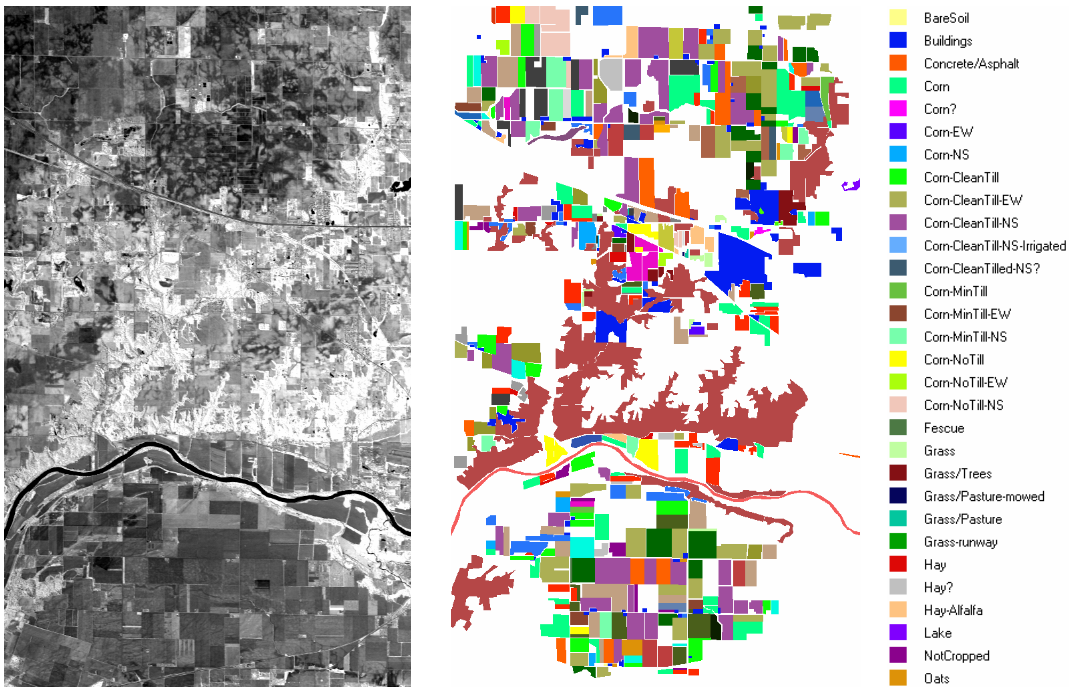

**Fig. 5. (Left) Spectral band at 587 nm wavelength of an AVIRIS scene comprising agricultural and forest features at Indian Pines, Indiana. (Right) Ground-truth map with 30 mutually exclusive land-cover classes.**

**Table 2. Classification accuracies and single-processor times for the morphological algorithm.**

| $I_{max}$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Accuracy (%) | 75.23 | 78.43 | 81.94 | 83.99 | 87.95 | 88.79 | 90.02 |
| Time (mins) | 9.54 | 19.56 | 27.82 | 37.06 | 46.91 | 54.68 | 64.79 |

As shown by Table 2, the morphological algorithm was able to achieve very high classification accuracies, especially for $I_{max} = 7$ (above 90%), but the measured processing times were extremely high and generally unacceptable in remote sensing applications. To investigate the parallel properties of the considered HeteroMPI-based algorithm, it was implemented on the heterogeneous cluster at UCD (see Table 1). Before reporting the timing results, we emphasize that the relative speeds of the heterogeneous processors were first estimated for different problem sizes (i.e., number of iterations ranging from $I_{max} = 1$ to $I_{max} = 7$) by incorporating the core computations of the morphological algorithm (erosion, dilation and MEI calculations) to a HeteroMPI-defined performance model. In order for such estimation to be accurate, it was necessary to include memory management considerations in the benchmark function to avoid disregarding important aspects such as virtual memory paging and cache considerations. In our particular implementation, we used a rather conservative approach which assumes that each heterogeneous processor has memory capacity sufficient to work with the entire hyperspectral data set locally. Based on previous work [2], this is a reasonable assumption in most hyperspectral imaging scenarios. Further, this provides us with a means to effectively model memory hierarchy-related parameters by simulating a largely *unfavorable* scenario in which each processor is forced to make use of reallocation/paging mechanisms due to cache misses. With the above assumptions in mind, Table 3 shows the execution times (in seconds) of the HeteroMPI-based parallel morphological algorithm in each of the processors of the heterogeneous cluster. As shown by Table 3, the heterogeneous algorithm was able to adapt efficiently to the heterogeneous computing environment where it was run. In particular, one can see that the heterogeneous algorithm executed on the HNOC was always about eleven times faster than the equivalent sequential algorithm executed on a Linux workstation which is almost identical to the **csultra** nodes in the considered HNOC (see Table 2). Most importantly, we experimentally tested that the mean processing times in the eight **Pg1cluster** processors were almost identical to the mean processing times in the seven **csultra** nodes (for all considered problem sizes). This fact reveals that the slight differences in the execution times reported on Table 3 are due to the intrinsic characteristics of the parallel problem, and not to platform heterogeneity which is accurately modeled by HeteroMPI.

**Table 3. Execution times (in seconds) of the HeteroMPI-based algorithm in each of the heterogeneous processors for different numbers of iterations.**

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

**Table 4. Load balancing rates for the HeteroMPI-based algorithm with different numbers of iterations.**

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

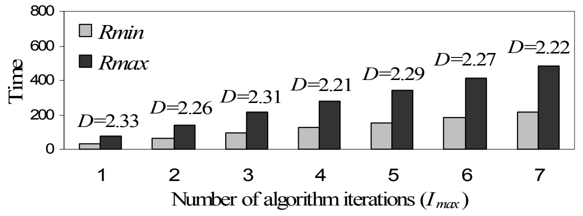

**Fig. 6. Load-balancing without memory considerations.**

In order to measure load balance, Table 4 shows the imbalance scores achieved by the parallel heterogeneous algorithm on the considered HNOC. The imbalance is defined as $D = R_{max} / R_{min}$, where $R_{max}$ and $R_{min}$ are the maxima and minima processor run times, respectively. Therefore, perfect balance is achieved when $D = 1$. The load balancing rates on Table 4 are superior to those reported in [2] for standard, spectral-based hyperspectral analysis algorithms executed in homogeneous computing platforms. Before concluding this section, we would like to emphasize the importance of incorporating considerations about memory capacity of the different nodes in the benchmark function used in the performance model. For illustrative purposes, Fig. 6 shows the values of $R_{max}$, $R_{min}$ and $D$ obtained on the considered HNOC for a parallel version of the proposed algorithm in which the benchmark function only modeled the processing power of heterogeneous processors and did not take into account memory-related parameters. The imbalance scores are also reported for completeness. Overall, Fig. 6 shows that disregarding memory considerations in the HeteroMPI performance model results in higher imbalance scores.

## 5. Conclusions and future lines

This paper describes our first experiences towards the utilization of HeteroMPI to implement efficient hyperspectral analysis algorithms on HNOCs. A spatial/spectral, morphological analysis algorithm is selected as a case study. Despite the fact that conventional hyperspectral imaging algorithms do not take into account the spatial information explicitly into the computations (which has traditionally been perceived as an advantage for the development of parallel implementations), experimental results suggest that the proposed HeteroMPI-based parallel algorithm is effective in terms of workload distribution. Although further work is required to improve load balance, the reported accuracies, load-balancing rates and execution times are superior to those reported in previous studies.

## References

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

[2] A. Plaza, D. Valencia, J. Plaza, P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing*, vol. 66, pp. 345-358, 2006.

[3] J. Dorband, J. Palencia, U. Ranawake, "Commodity computing clusters at Goddard Space Flight Center," *Journal of Space Communication*, vol. 1, no. 3, 2003.

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

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

[6] A. Lastovetsky and R. Reddy, "HeteroMPI: Towards a message-passing library for heterogeneous networks of computers," *Journal of Parallel and Distributed Computing*, vol. 66, pp. 197-220, 2006.

[7] A. Lastovetsky, "Adaptive parallel computing on heterogeneous networks with mpC," *Parallel Computing*, vol. 28, pp. 1369-1407, 2002.

[8] A. Plaza, P. Martinez, J. Plaza, R. Perez, "Dimensionality reduction and classification of hyperspectral image data using sequences of extended morphological transformations," *IEEE Trans. Geoscience and Remote Sensing*, vol. 43, no. 3, pp. 466-479, March 2005.