AMEEPAR: Parallel Morphological Algorithm for Hyperspectral Image Classification on Heterogeneous Networks of Workstations

Antonio Plaza, Javier Plaza, and David Valencia

Department of Computer Science, University of Extremadura
Avda. de la Universidad s/n, E-10071 Caceres, Spain
{aplaza, jplaza, davaleco}@unex.es

Abstract. Hyperspectral imaging is a new technique in remote sensing that generates hundreds of images corresponding to different wavelength channels for the same area on the surface of the Earth. Most available techniques for hyperspectral image classification focus on analyzing the data without incorporating the spatial information; i.e. the data is treated not as an image but as an unordered listing of spectral measurements where the spatial coordinates can be shuffled arbitrarily without affecting the final analysis. Despite the growing interest in the development of techniques for interpretation and classification of such high-dimensional imagery, only a few efforts devoted to the design of parallel implementations exist in the open literature. In this paper, we describe AMEEPAR, a parallel morphological algorithm that integrates the spatial and spectral information. The algorithm has been specifically optimized in this work for execution on heterogeneous networks of workstations. The parallel properties and classification accuracy of the proposed approach are evaluated using four networks of workstations distributed among different locations, and a massively parallel Beowulf cluster at NASA's Goddard Space Flight Center.

1 Introduction

The rapid development of space and computer technologies has made possible to store a sheer volume of remotely sensed image data, collected from heterogeneous sources. In particular, NASA is continuously gathering imagery data with hyperspectral Earth observing sensors such as Jet Propulsion Laboratory's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)[1], which covers the wavelength region from 0.4 to 2.5 μm using 224 spectral channels at nominal spectral resolution of 10 nm (see Fig. 1). The incorporation of hyperspectral sensors on airborne/satellite platforms is currently producing a nearly continual stream of high spatial and spectral resolution data, and this high data volume demands efficient and robust data analysis techniques.

The underlying assumption governing most available techniques for hyperspectral analysis is that each pixel vector measures the response of multiple underlying materials at each site. A hyperspectral image (sometimes referred to as "image cube") is often a combination of two situations: a few sites in a scene are pure macroscopic materials, e.g., soil or water, but many other are mixtures of materials. For instance, the vegetation pixel in Fig. 1 may comprise a mixture of different types of vegetation, soil, atmospheric interferers, etc. Further, most available techniques only use the spectral information available in the image data. Therefore, such techniques would yield the same result for a data cube, and for the same data cube where the spatial positions have been randomly permuted. By taking into account the complementary nature of spatial and spectral information in simultaneous fashion, it is possible to alleviate the problems related to each of them taken separately.

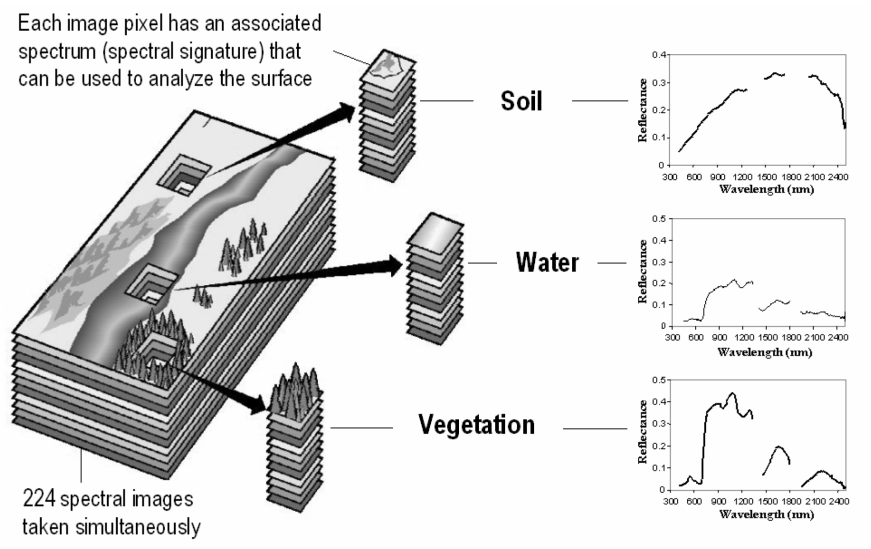
Each image pixel has an associated spectrum (spectral signature) that can be used to analyze the surface
Soil
Water
Vegetation
224 spectral images taken simultaneously
Reflectance
Wavelength (nm)
300 600 900 1200 1500 1800 2100 2400

Fig. 1. Hyperspectral imaging. Each pixel is given by a vector of values or "spectral signature".

While integrated spatial/spectral developments hold great promise, they also introduce new processing challenges. In turn, many applications require a quick response (e.g., target detection for homeland defense/security purposes, risk/hazard prevention/response including wild land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination). In recent years, several efforts have been focused on the incorporation of high-performance computing (HPC) models in remote sensing applications. Unfortunately, only a few research efforts devoted to HPC-based hyperspectral imaging exist in the open literature (which is partly due to non-disclosure restrictions in some cases). This paper develops an efficient parallel algorithm for spatial/spectral analysis in heterogeneous computing environments[2], which are expected to become a tool of choice in many on-going and planned remote sensing missions. The method is a parallel version of the Automated Morphological Endmember Extraction (AMEE) algorithm[3], which integrates the spatial and spectral information in the data. The paper is organized as follows. In the following section, we introduce the AMEE algorithm and its parallel implementation. The parallel algorithm is then evaluated using four networks of workstations and a Beowulf cluster at NASA's Goddard Space Flight Center. The paper concludes with some remarks.

2 Parallel Algorithm for Hyperspectral Image Classification

The AMEEPAR algorithm is based on mathematical morphology[3], a classic nonlinear spatial processing technique that provides a remarkable framework to achieve the desired integration of spatial and spectral responses. First, we provide an overview of the AMEE algorithm. Then, we provide a description of its parallel implementation.

2.1 AMEE Algorithm

The AMEE algorithm relies on two basic morphological operations: erosion and dilation[3]. Let us denote by $f(x,y)$ the pixel vector at spatial coordinates (x,y) of a hyperspectral scene., The erosion of f by B (a so-called "structuring element") consists of selecting the "minimum" pixel vector in the spatial neighborhood of $f(x,y)$ defined by B. Similarly, the dilation of f by B consists of selecting the "maximum" pixel vector (called endmember in hyperspectral analysis terminology). We provide below a version of AMEE which is tuned for unsupervised classification of hyperspectral data.

AMEE algorithm
Inputs: Image cube f , B , Number of iterations I_{max} , Number of endmembers p.
Output: MEI image.

1. Set $i=1$ and initialize $\mathrm{MEI}(x,y)=0$ for each pixel.
2. Move B through all the pixels of f , defining a local spatial search area around each $f(x,y)$, and calculate the maximum and the minimum pixel at each B-neighborhood using morphological dilation and erosion, respectively, as follows:
$$(f \oplus B)(x,y) = \{f(\mathrm{x}-\mathrm{s}',\mathrm{y}-\mathrm{t}'),(s',t') = \arg\max_{(s,t)\in Z^2(B)}\{D_B[f(\mathrm{x}-\mathrm{s},\mathrm{y}-\mathrm{t})]\}\}$$
$$(f \Theta B)(x,y) = \{f(x+s',y+t'),(s',t') = \arg\min_{(s,t)\in Z^2(B)}\{D_B[f(x+s,y+t)]\}\}$$
3. Update $\mathrm{MEI}(x,y)$ at each pixel using $\mathrm{MEI}(x,y)=\mathrm{SAD}[(f \oplus B)(x,y),(f \oplus B)(x,y)]$, where SAD is the spectral angle distance[3], i.e., the arc cosine of the vector dot product divided by the product of the norms.
4. Set $i=i+1$. If $i=I_{\max}$ then go to step 4. Otherwise, set $f=f \oplus B$ and go to 2.
5. Select the set of p pixel vectors in f with higher associated score in the resulting MEI image (called endmember pixels) and form a unique spectral set of $q \le p$ pixels by calculating the SAD for all vector pairs. Estimate the fractional abundance, $\alpha_i(x,y)$, of those q signatures at $f(x,y)$ using a linear mixture model.
6. Obtain a classification label for each pixel $f(x,y)$ by assigning it to a class given by the endmember with the highest fractional abundance score in that pixel. All estimated abundance fractions $\{\alpha_1(x,y),\alpha_2(x,y),...,\alpha_q(x,y)\}$ are compared, and the one with the maximum value is found, say:
$$\alpha_{i*}(x,y)\text{, with } i* = \arg\left\{\max_{1\le i\le q}\{\alpha_i(x,y)\}\right\}.$$

2.2 AMEEPAR Algorithm

A major requirement for efficient parallel algorithms on distributed memory systems is finding a data decomposition that minimizes the communication between the processors[4]. For that purpose, AMEEPAR adopts a spatial-domain partitioning approach, in which the same pixel vector is never split among several processors (in spectral-domain parallel, the structuring element-based calculations made for each hyperspectral pixel need to originate from several processing elements, and thus require intensive inter-processor communication). A second important issue in the design of the AMEEPAR is that redundant information (*overlap* borders) are added to local partitions to avoid accesses outside the partition domain when the structuring element computation requires pixel vectors from other partitions[5].

The algorithm has been implemented in the C++ programming language using calls to message passing interface (MPI). It uses a master-slave paradigm in which the master scatters hyperspectral data without creating partial data structures at the root. For that purpose, we make use of MPI *derived datatypes* to directly scatter data structures, which may be stored non-contiguously in memory, in a single communication step. In order to slice the available data into chunks, there is a need to balance the workloads of k heterogeneous resources so that each processor P_i will accomplish a share α_i of the total workload, with $\alpha_i \geq 0$ for $1 \leq i \leq k$ and $\sum_{i=1}^{k} \alpha_i = 1$.

AMEEPAR algorithm

Inputs: Image cube f, B, Number of iterations I_{max}, Number of endmembers p.

Output: MEI image.

1. Obtain necessary information about the heterogeneous system, including the number of available processors, k, each processor's identification number, $\{P_i\}_{i=1}^{k}$, and processor cycle-times, $\{w_i\}_{i=1}^{k}$.
2. Determine the total volume of information, R, that needs to be replicated from the original data volume, V, in accordance with the adopted border overlap strategy.
3. Let the total workload to be handled by the algorithm be given by $W = V + R$.
4. Set $\alpha_i = \left\lfloor \frac{(p/w_i)}{\sum_{i=1}^{k} (1/w_i)} \right\rfloor$ for all $i \in \{1,\ldots,k\}$.
5. For $m = \sum_{i=1}^{k} \alpha_i$ to $(V + R)$ find $j \in \{1,\ldots,k\}$, so that $w_j \cdot (\alpha_j + 1) = \min\{w_i \cdot (\alpha_i + 1)\}_{i=1}^{k}$ and set $\alpha_k = \alpha_k + 1$.
6. Use the resulting $\{\alpha_i\}_{i=1}^{k}$ to obtain a set of k spatial-domain heterogeneous partitions of f, and send its corresponding partition to each processor along with B.
7. Broadcast B, I_{max}, p to heterogeneous processors, and execute AMEE in parallel.
8. Collect all the individual classification results provided by each processor P_i, and merge them together to form a final classification image.

It should be noted that a homogeneous version of the AMEEPAR algorithm above (called HomoAMEEPAR)[5] can be obtained by rewriting step 4 as $\alpha_i = p/w_i$ for all $i \in \{1,\ldots,k\}$, where w_i is a constant communication speed between each processor pair.

3 Experimental Results

Before describing our results, we introduce the parallel computing architectures used in experiments, which include four networks of workstations and a Beowulf cluster:

1. *Fully heterogeneous network.* Consists of 16 different SGI, Solaris and Linux workstations, and four communication segments. Table 1 shows the cycle-times of the processors. The communication network consists of four 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. Table 2 shows the capacity of all point-to-point communications, expressed as the time in milliseconds to transfer a one-megabit message between each pair (P_i, P_j) in the network.
2. *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 with capacity $c = 26.64$ milliseconds.
3. *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.
4. *Partially homogeneous network.* Formed by 16 identical Linux workstations with processor cycle-time of $w = 0.0131$ seconds per megaflop but interconnected using the heterogeneous communication network shown in Table 2.
5. *Thunderhead Beowulf cluster.* Formed by 256 identical 2.4 GHz Intel Xeon nodes, each with 1 GB of memory and 80 GB of main memory (see http://newton.gsf.nasa.gov/thunderhead).

Table 1. Processor cycle-times (in seconds per megaflop) for the heterogeneous processors

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}
.005	.010	.020	.007	.010	.007	.007	.010	.007	.045	.013	.013	.013	.013	.013	.013

According to a recent study[2], the four networks of workstations above can be considered *equivalent* since they satisfy the following three principles: 1) they have the same number of processors; 2) the average processor speed is the same in all cases; and 3) the aggregate characteristics of the communication network are all the same. At this point, we reiterate that the configuration of the four networks above was carefully designed to make sure that the three principles above were satisfied, and the aggregate performance was the same.

The AMEEPAR algorithm (and its homogeneous version) were applied to a hyperspectral scene collected by the AVIRIS sensor, which consists of 2048x614 pixels, 224 spectral bands, and moderate spatial resolution (20-meter pixels). It was gathered over the Indian Pines region in Indiana, and represents a challenging classification problem. Part of these 275 MB data set are available online, along with ground-truth, from http://dynamo.ecn.purdue.edu/~biehl/MultiSpec. In experiments, the structuring element size was fixed to B_{3x3} to reduce the amount of redundant computations[3]. The number of iterations I_{max} was increased from 1 to 5 (according to our implementation, increasing the value of I_{max} is equivalent to considering a larger spatial context). Finally, we set the number of endmembers to be extracted, p, to 16 after estimating the intrinsic dimensionality of the data[1].

Table 2. Capacity of links (measured by the time in milliseconds to transfer a one-megabit message) for the heterogeneous communication network

	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	-	19.2	19.2	19.2	48.3	48.3	48.3	48.3	96.6	96.6	154	154	154	154	154	154
P_2	19.2	-	19.2	19.2	48.3	48.3	48.3	48.3	96.6	96.6	154	154	154	154	154	154
P_3	19.2	19.2	-	19.2	48.3	48.3	48.3	48.3	96.6	96.6	154	154	154	154	154	154
P_4	19.2	19.2	19.2	-	48.3	48.3	48.3	48.3	96.6	96.6	154	154	154	154	154	154
P_5	48.3	48.3	48.3	48.3	-	17.6	17.6	17.6	48.3	48.3	106	106	106	106	106	106
P_6	48.3	48.3	48.3	48.3	17.6	-	17.6	17.6	48.3	48.3	106	106	106	106	106	106
P_7	48.3	48.3	48.3	48.3	17.6	17.6	-	17.6	48.3	48.3	106	106	106	106	106	106
P_8	48.3	48.3	48.3	48.3	17.6	17.6	17.6	-	48.3	48.3	106	106	106	106	106	106
P_9	96.6	96.6	96.6	96.6	48.3	48.3	48.3	48.3	-	16.3	58.1	58.1	58.1	58.1	58.1	58.1
P_{10}	96.6	96.6	96.6	96.6	48.3	48.3	48.3	48.3	16.3	-	58.1	58.1	58.1	58.1	58.1	58.1
P_{11}	154	154	154	154	106	106	106	106	58.1	58.1	-	14.2	14.2	14.2	14.2	14.2
P_{12}	154	154	154	154	106	106	106	106	58.1	58.1	14.2	-	14.2	14.2	14.2	14.2
P_{13}	154	154	154	154	106	106	106	106	58.1	58.1	14.2	14.2	-	14.2	14.2	14.2
P_{14}	154	154	154	154	106	106	106	106	58.1	58.1	14.2	14.2	14.2	-	14.2	14.2
P_{15}	154	154	154	154	106	106	106	106	58.1	58.1	14.2	14.2	14.2	14.2	-	14.2
P_{16}	154	154	154	154	106	106	106	106	58.1	58.1	14.2	14.2	14.2	14.2	14.2	-

Table 3. Execution times and classification accuracies on the four networks of workstations

Algorithm	**AMEEPAR**			**HomoAMEEPAR**		
I_{max}	1	3	5	1	3	5
Fully heterogeneous network	284	321	379	1456	1528	1589
Fully homogeneous network	298	339	372	281	317	355
Partially heterogeneous network	288	325	365	1223	1267	1301
Partially homogeneous network	294	336	371	437	472	512
Classification accuracy (%)	69.23	74.48	91.05	69.23	74.48	91.05

Table 3 shows the classification accuracies and execution times obtained by both algorithms in the four considered networks of workstations. As expected, the execution times reported on Table 3 show that the heterogeneous algorithm was able to adapt much better to fully (or partially) heterogeneous environments than the homogeneous version, which only performed satisfactorily on the fully homogeneous network. One can see that AMEEPAR was several times faster than its homogeneous counterpart in the fully heterogeneous network, and also in both the partially homogeneous and the partially heterogeneous networks. On the other hand, the HomoAMEEPAR algorithm only slightly outperformed its heterogeneous counterpart in the fully homogeneous network. Table 3 also reveals that the performance of the heterogeneous algorithm on the fully heterogeneous network was almost the same as that evidenced by the homogeneous algorithm on the fully homogeneous network. This reveals that the heterogeneous algorithm was very close to the optimal heterogeneous modification of the basic homogeneous one[2].

In order to explore load balance of the two algorithms above on the four networks of workstations, Table 4 shows the imbalance scores achieved by the different algorithms (implemented with I_{max} set to 5 iterations). 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 report the imbalance considering all processors, D_{All}, and also considering all processors but the root, D_{Minus}. In all cases, load balance was similar when the root processor was not included, which means that the master node does not have high computation load. It is also clear from Table 4 that the homogeneous algorithm executed on the heterogeneous network provided the highest values of D_{All} and D_{Minus} (and hence the highest imbalance), while the heterogeneous algorithm always resulted in values of D_{All} and D_{Minus} which were closer to 1, regardless of the platform where it was run.

Table 4. Load balancing rates on the four networks of workstations

Algorithm	**AMEEPAR**		**HomoAMEEPAR**	
Imbalance	D_{All}	D_{Minus}	D_{All}	D_{Minus}
Fully heterogeneous network	1.09	1.02	1.51	1.47
Fully homogeneous network	1.16	1.07	1.08	1.02
Partially heterogeneous network	1.11	1.03	1.44	1.41
Partially homogeneous network	1.13	1.05	1.33	1.26

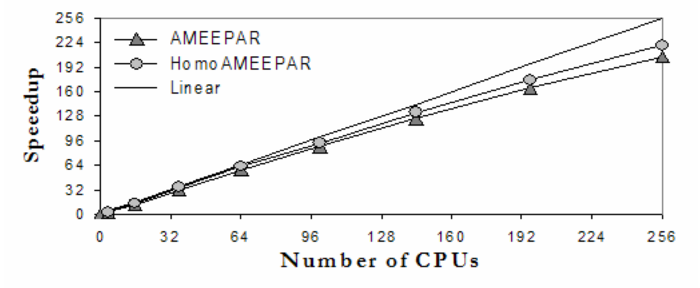

Fig. 2. Scalability of AMEEPAR and HomoAMEEPAR on the Thunderhead Beowulf cluster

Finally, and with the ultimate goal of exploring issues of scalability and portability of heterogeneous algorithms to existing massively parallel computing platforms (which are mainly homogeneous in nature), we have also compared the performance of the two algorithms on the Thunderhead Beowulf cluster. Fig. 2 shows the speedups achieved by AMEEPAR (and its homogeneous version) over a single-processor run of the sequential AMEE algorithm on Thunderhead, all of them implemented with I_{max} set to 5. As Fig. 2 shows, the scalability of AMEEPAR was similar to that achieved by its homogeneous prototype on the Beowulf cluster. Although AMEEPAR introduces redundant calculations, which are expected to slow down the computation *a priori*, the measured speedups tend to be higher for large structuring element sizes, a fact that reveals that the proposed scheme scales better as the size of the problem increases. As a result, the dominant issue in AMEEPAR is problem size, which makes the algorithm particularly appealing for high-dimensional imaging applications. To conclude this section, we must also note that AMEEPAR was able to provide near real-time classification performance in the Thunderhead cluster. The algorithm only required 8 seconds to produce a result using 256 processors, and 30 seconds with 64 processors. This is a relevant achievement, in particular, if we take into account that 1874 seconds were required to process the entire data set using a single Thunderhead processor.

4 Conclusions

This paper provided an investigation of a parallel morphological technique to extract relevant information from hyperspectral image data sets in heterogeneous computing environments. Experimental results reveal that the proposed algorithm offers a simple, yet highly scalable and relatively platform-independent solution in the context of hyperspectral image classification applications. Although many available approaches do not take into account the spatial information explicitly (a fact that has been perceived as an advantage for the development of parallel implementations), experimental results in this paper suggest that spatial/spectral classification approaches may indeed be tuned for "pleasingly parallel execution" due to the windowing nature of such algorithms, and also because they can effectively balance the load in heterogeneous systems. The proposed method seems ideally suitable for data mining applications, which previously looked too computationally intensive due to the immense data archives common to remote sensing problems. Combining the readily available computational power offered by heterogeneous platforms with the new sensor instruments may introduce major changes in the systems used for exploiting Earth and planetary data.

References

1. Green, R.O., et al.: Imaging Spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS). Remote Sensing of Environment, 65 (1998) 227–248
2. Lastovetsky, A., Reddy, R.: On Performance Analysis of Heterogeneous Parallel Algorithms. Parallel Computing, 30 (2004) 1195–1216
3. Plaza, A., Martinez, P., Perez, R., Plaza, J.: Spatial/Spectral Endmember Extraction by Multidimensional Morphological Operations. IEEE Transactions on Geoscience and Remote Sensing, 9 (2002) 2025–2041
4. Seinstra, F.J., Koelma, D., Geusebroek, J.M.: A Software Architecture for User Transparent Parallel Image Processing. Parallel Computing 28 (2002) 967–993
5. Plaza, A., Valencia, D., Plaza, J.: Commodity Cluster-Based Parallel Processing of Hyperspectral Imagery. Journal of Parallel and Distributed Computing 66 (2006) 345–358