DOI 10.12694/scpe.v17i1.1146
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 1, pp. 1–11. http://www.scpe.org © 2016 SCPE

PERFORMANCE OPTIMIZATIONS FOR AN AUTOMATIC TARGET GENERATION PROCESS IN HYPERSPECTRAL ANALYSIS

FERNANDO SIERRA-PAJUELO, ABEL PAZ-GALLARDO* AND ANTONIO PLAZA†

Abstract. Hyperspectral sensors acquire images with hundreds of spectral channels. These images have a lot of information in both spectral and spatial domain, and with this kind of information different research studies can be accomplished. In this work, we present several optimizations for hyperspectral image processing algorithms intended to detect targets in hyperspectral images. The hyperspectral image selected for our study was collected by the NASAs Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) over the World Trade Center (WTC) in New York, five days after September 11th attack. The algorithm used in our experiments is the automated target generation process (ATGP) and our optimizations comprise parallel versions of the algorithm developed using open multi-processing (OpenMP) and message passing interface (MPI). Our experiments indicate that the ATGP can be successfully implemented in parallel in multicore and cluster computing architectures, including Intel Xeon Phi.

Key words: Hyperspectral imaging, automatic target generation process (ATGP), open multi-processing (OpenMP), message passing interface (MPI), Xeon Phi

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction.

Hyperspectral imaging [1] is concerned with the analysis and interpretation of spectra acquired form a given scene (or specific object) by an airborne or satellite sensor [2]. Instruments such as the Airborne Visible Infrared Imaging Spectrometer (AVIRIS) [3] are able to record the visible and near-infrared spectrum of the reflected light using 224 spectral bands. As shown in Fig. 1.1, the resulting "image cube" is a stack of images in which each pixel has an associated spectral signature or fingerprint that uniquely characterizes the underlying objects [4]. The resulting data volume typically comprises several GBs per flight [5].

The special properties of hyperspectral data have significantly expanded the domain of many analysis techniques, including (supervised and unsupervised) classification, spectral unmixing, compression, target, and anomaly detection [6, 7, 8, 9, 10]. Specifically, the automatic detection of targets and anomalies is highly relevant in many application domains, like fire control in forests or detect deposit of minerals, including those addressed in Fig. 1.2 [11, 12, 13].

The automatic detection of targets and anomalies in hyperspectral images is highly relevant in many applications and it is particularly important for defense and security applications [14, 15], as well as for rare mineral detection in geology [16] or location of infected trees in forestry. In this paper, we developed and compared several efficient parallel versions of the automatic target generation process (ATGP) algorithm [4]. This algorithm was designed to find spectral signatures with orthogonal projections. The considered method includes the spectral angle distance (SAD) and the parallel versions are developed with open multi-processing (OpenMP) and message passing interface (MPI). They are focused on identifying thermal hot spots in a complex urban background, using AVIRIS hyperspectral data collected over the World Trade Center in New York just five days after the terrorist attack of September 11th, 2001.

2. Methods.

In this section, we will describe the target detection algorithm that will be efficiently implemented in parallel: the ATGP algorithm [4], it was created to find spectral signatures using orthogonal projections. The starting point of the algorithm is the brightest pixel in the image, similar to other existing measures, it is possible to use different starting points instead of the brightest pixel. But, in these cases, it has been experimentally verified that the pixel is always detected in a small number of iterations if not chosen as a point starting [17]. Therefore, it seems reasonable to use as a starting condition. Next, we show a detailed algorithmic description of the classical version of this algorithm. It begins by an orthogonal subspace projector specified by the following expression:

$$P_{\mathbf{U}}^{\perp} = \mathbf{I} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T \tag{2.1}$$

*Centro Extremeño de Tecnologías Avanzadas, Trujillo, Cáceres, Spain (fernando.sierra@externos.ciemat.es, abelfrancisco.paz@ ciemat.es).

†Hyperspectral Computing Laboratory, University of Extremadura, Politécnica de Cáceres, Cáceres, Spain (aplaza@unex.es)

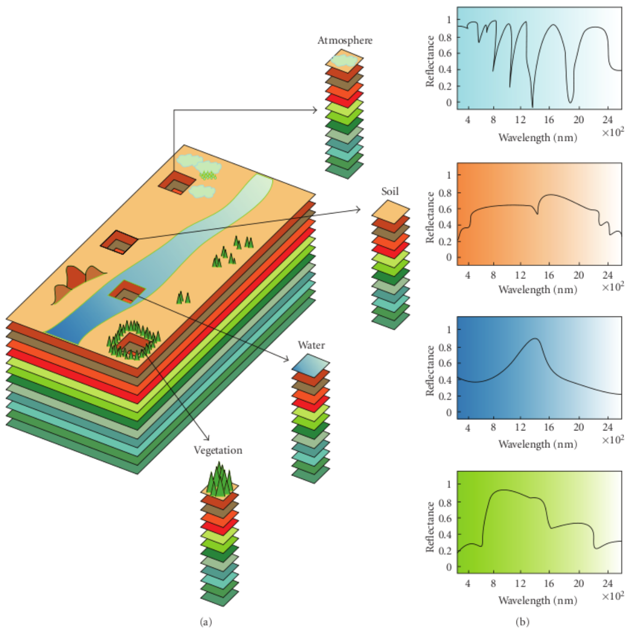

Fig. 1.1: Hyperspectral imaging concept. The reflectance and wavelength of pure pixels and mixed pixels are described in the figure. They are different and unique for each kind of pixel.

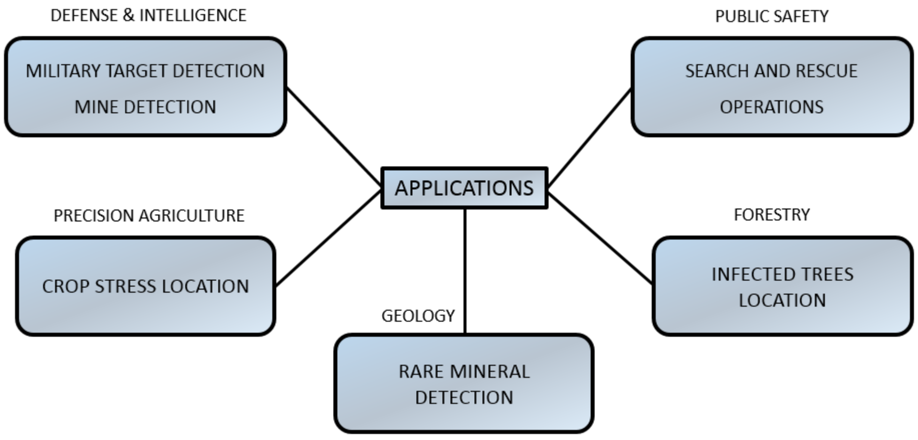

Fig. 1.2: Applications of target and anomaly detection. Detect targets in a war, humanity missions or you could use this to find mines and deactivate it.

where $\mathbf{U}$ is a matrix of spectral signatures, $\mathbf{U}^T$ is the transpose of the matrix and $\mathbf{I}$ is the identity matrix. ATGP algorithm uses the orthogonal projection of the equation 2.1 in each iteration to find a number of pixels or bands vectors from an initial pixel that is passed to the algorithm as ATGP value and which is usually the brightest pixel. This algorithm performs the following steps:

1. Calculate t_0, the brightest pixel of the hyperspectral image, using equation 2.2, where $\mathbf{F}_{(x,y)}$ is the pixel (vector) at coordinates (x, y) in the image. The brightest pixel is that with greater value performing

the vector product between the associated vector with that pixel and its transposed $\mathbf{F}(x,y)^T$.

$$\mathbf{t}_0 = arg\{max_{(x,y)}[\mathbf{F}(x,y)^T \cdot \mathbf{F}(x,y)]\} \tag{2.2}$$

2. Apply an orthogonal projection operator tagged as $P_{\mathbf{U}}^{\perp}$, using the expression 2.1, with $\mathbf{U} = \mathbf{t}_0$. This operator is applied to all pixels of the hyperspectral image.
3. Then, the algorithm finds a new target named as $\mathbf{t}_1$ with the greater value in the complementary space $< \mathbf{t}_0 >^{\perp}$, orthogonal to $\mathbf{t}_0$, using equation 2.3. In other words, the algorithm finds the pixel with higher orthogonality respect to $\mathbf{t}_0$.

$$\mathbf{t}_1 = arg\{max_{(x,y)}[P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]^T \cdot [P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]\} \tag{2.3}$$

4. The next step is to modify the $\mathbf{U}$ matrix and adding the new target found, that is $\mathbf{U} = [t_0 t_1]$.
5. The algorithm finds a new target named $\mathbf{t}_2$ with the highest complementary space $< \mathbf{t}_0, \mathbf{t}_1 >^{\perp}$, orthogonal to $\mathbf{t}_0$ and $\mathbf{t}_1$, using the expression 2.4. At this point, the orthogonal projector is based on a matrix $\mathbf{U} = [\mathbf{t}_0 \mathbf{t}_1]$ and the orthogonally concept is different.

$$\mathbf{t}_2 = arg\{max_{(x,y)}[P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]^T \cdot [P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]\} \tag{2.4}$$

6. The process is repeated iteratively, to find a third target, $\mathbf{t}_3$, a fourth target $\mathbf{t}_4$, until a certain condition satisfies the termination for the algorithm. The termination condition considered in this paper is to achieve a number of targets p that is determined as an input parameter to the algorithm.

3. Parallel Implementations. Partitioning or data division prior to processing of the hyperspectral image can be done essentially by using two different strategies [18]:

- Spectral partitioning considers that different parallel architecture processors may contain non-overlapping parts of the same spectral signature (pixel). This schema has the disadvantage that, considering the spectral signature (vector) as a minimum unit for processing algorithms, it would be necessary to include more communication operations for each calculation of the metric that is used. From the viewpoint of the parallelization of the algorithm, which is based on applying repetitive computations, this type of partitioning means a huge cost in terms of communication operations. Clusters of computers are made up of different processing units interconnected via a communication network [19]. In previous works, it has been reported that data-parallel approaches, in which the hyperspectral data is partitioned among different processing units, are particularly effective for parallel processing in this type of high-performance computing systems [5, 20].
- Spatial partitioning considers that the same spectral signature or pixel cannot be partitioned in different units of the parallel processing architecture. We can work locally with the image on each processor, eliminating much of the communication load of the algorithm. In this way, we just need to make global communications to synchronize processes or get results in each iteration of the algorithm [25].

Our parallel implementation uses spatial partitioning so that each node carries a certain portion of the image, which can be managed easily indicating each participant node from where to start reading and the number of lines associated with the node. Besides, another reason for selecting spatial versus spectral partitioning is that, with spatial partitioning, each pixel vector (spectral signature) remains in the same processing unit. As the spectral signature is the minimum unit of computation in most hyperspectral imaging algorithms, keeping the spectral signatures in the same processor reduces drastically the amount of inter-processor communications. A comparison between spatial versus spectral partitioning for parallel hyperspectral algorithms has been reported in [21]. The parallelization by spatial decomposition adopted by our implementation is described graphically in Fig. 3.1. We opted for a spatial partitioning for that reason and other that we describe below:

- The spatial partitioning is a natural alternative to parallelize algorithms based on a processing by window, such as algorithms for detect anomalies and target detection.
- Another reason to select the spatial partitioning versus spectral is that the spatial partitioning allows to reduce communications between processors of the parallel architecture significantly. The most part of hyperspectral analysis algorithms consider the spectral signature as the minimum processing unit and, therefore, spectral partitioning involve inter-processor communications on pixel level increasing

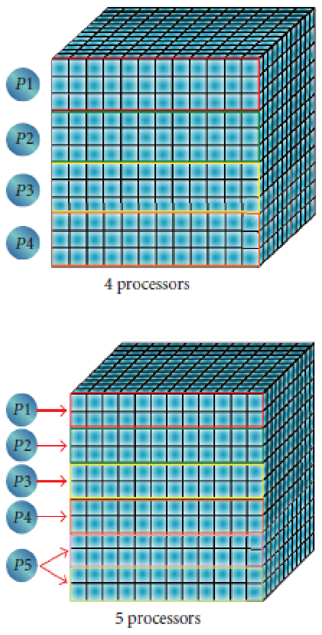

Fig. 3.1: Spatial-domain decomposition of a hyperspectral data set.

Table 4.1: Hyperspectral Image Features that we use.

Lines	614
Samples	512
Spectral Bands	224
Spectral Range	0,4 - 2,5 μm
Spatial Resolution	1.7 metres/pixel

the associated communications costs. The result of this is a lower parallel performance and scalability issues when the processor number increases.

- The last reason to select spatial partitioning is the ability to reuse code and improve the portability of parallel algorithms developed to different architectures. Is highly desirable to reuse serial code when we are developing parallel version due to the complexity of some analysis techniques. The spatial partitioning allows parallelism of fine-grained to make easier to use a parallel algorithm to different portions of data in which all spectral information is saved, allowing the transformation of the serial code to parallel code easier than applying a spectral partitioning.

This parallel scheme preserves in any case the sequential algorithm functionality of ATGP, except that in this case a matrix of intermediate values is calculated in each of the nodes and then an update is performed globally to share which node has the maximum value.

4. Experimental Results. In this section, we evaluate the parallel performance of the implementation introduced in the previous section.

Hyperspectral image considered. The image to be processed (AVIRIS World Trade Center) was taken five days after September 11th attacks. The main features of the image are in the Table 4.1. Note that the spatial resolution of the image is very high for what is usually in AVIRIS. This is because the image corresponds to a low altitude flight in which this flight pretended to obtain the highest possible spatial resolution. Therefore,the impact of mixed pixels is much smaller than we would expect and it is possible to do an study more focused on

Table 4.2: RGB Spectral Bands used for Fig. 4.1.

Color	Band
Red	1682
Green	1107
Blue	655

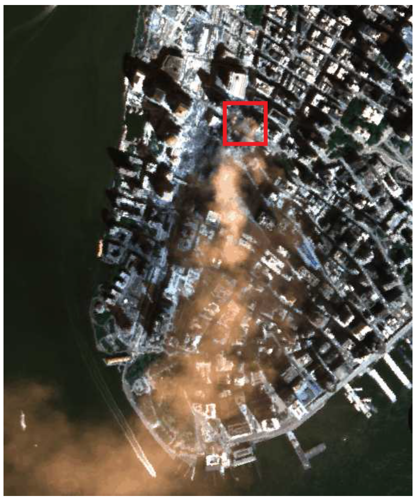
Fig. 4.1: WTC Hyperspectral image with RGB Colors.

detecting anomalies (fires).

We make a fake color composition in RGB with three bands (see Table 4.2) and we could see the vegetation in green color and the fires in gray hue. The smoke has the source in the red square (WTC Area) and its going to the south of the island, with a blue hue because the smoke has a high reflectance in the $655\mu m$ wavelength.

As we could see in the Fig. 4.1, it should be pointed out that the automatic detection of fires in the WTC is a very complex problem, due to the diversity of the urban environment in which fires are located. This complicates the discrimination between points of interest (fires) and background due to the complexity of the background, that has many different spectral substances as expected in a urban landscape.

4.1. Sequential. For comparison we use the ATGP sequential version and run this code on a computer with an 2x Intel Xeon processors model E5649 at 2.53 GHz with 6 cores and 24 GB of DDR3 memory. The experimental results show the time to load and process the image completely. We calculated the average of five executions for each result. The time spent by the sequential version of the algorithm in the considered platform was 18.42 seconds with a standard deviation of 0.21 seconds.

4.2. Performance optimizations with OpenMP. In this section, we evaluate the performance optimizations with OpenMP. The test-bed was performed in two different platforms:

- s6030: NUMA (Non-Uniform Memory Access) shared memory platform with 64 cores, 8x Intel Xeon X7550 at 2.00 GHz and 1 Terabyte of memory.
- Computational node: 2x Intel Xeon processors model E5649 at 2.53 GHz with 6 cores and 24 GB of DDR3 memory.

The most important part of the algorithm is the ATGP method. As we have seen in Section 2, it's a highly iterative algorithm with three *for* loops to scroll the image and perform operations. In these loops we will do the study using OpenMP. Before we start to get the optimal results, we have performed various tests to find the optimal state of OpenMP code.To make easier the understanding, we include 2 different pseudo-codes

Algorithm 1 ATGP algorithm pseudo-code

```
Input: ImageFile ImageHeader Targets
1: Read header information.
2: Read_Header(ImageHeader) →bands,lines,samples
3:
4: Inicialize Matrix and Vectors
5: iVector[targets][2] ←0
6:
7: Load image into an array
8: Load_Image(ImageFile, iVector)
9:
10: Pixel more brilliant to start.
11: Get_Max_Bright() →iVector
12:
13: Calculate targets
14: for i =0,i <targets do
15:    max_atgp =ATGP(iVector,lines,samples)
16:    Save the coordinates P[i][0] and P[i][1]
17: end for
18: Return the results P[targets][2]
```

Algorithm 2 ATGP method pseudo-code

```
Input: iVector imageLines imageSamples
Output: maxValue
1: Compare the image pixels
2: Where n and n2 are values for lines and samples, respectively.
3: pragma omp parallel for num_threads(n) private(i)
4: for i =0,i <lines do
5:    pragma omp parallel for num_threads(n2) private(Values,k)
6:    for k =0,k <samples do
7:       Calculate de maximum distance
8:       for j =0,j <bands do
9:          Get vector for compare
10:         vector[j] ←iVector[j*lines]
11:      end for
12:      Calulate de Spectral Angle Distance
13:      maxValue ←Distance(vector[j],bands)
14:   end for
15: end for
16: Return maxValue
```

(Algorithm 1 and 2). It can help to reproduce the code. In Algorithm 2 we use two openmp *pragma* to distribute block of data on cores and nodes. While **n** is used to select the lines, **n2** is used for the samples. According to our experiments, the optimal **n** and **n2** are *16* and *32*, respectively.

After performing various executions we reached the optimal solution for each machine, s6030 and computational node. We run our optimized OpenMP code in the computational node with 2, 4 and 8 cores. We always try to use the thread-to-core binding with 1-1 ratio to be more efficient. If we evaluate the results obtained (see Table 4.3), we could see that the s6030 environment is faster than the computational environment, as Fig. 4.2 shows.

Table 4.3: Multicore scalability study with OpenMP code optimizations (Time in seconds) with five executions.In s6030 with same cores is faster because is a shared memory machine and OpenMP code take advantage of this.

	Computational Node	Speed-up	s6030	Speed-up
2 Cores	2,4868 s	7,4071	1,5246 s	12,0818
4 Cores	2,3790 s	7,7427	1,4953 s	12,3185
8 Cores	2,1709 s	8,4849	1,4926 s	12.3408

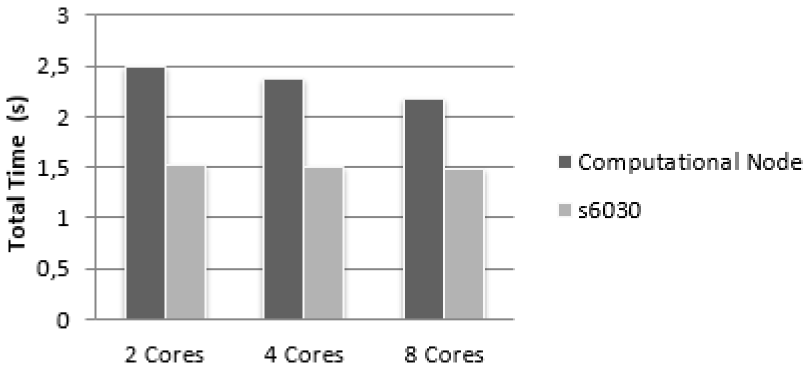

Fig. 4.2: Multicore scalability study in the two environments (Time in seconds) with five executions.It's clearly how the multicore scalability is faster in a shared memory machine (s6030).

Besides the ATGP method, we parallelized using OpenMP other loops into the code and we always make a study of the optimal number of threads for each loop. We are compiling it with using the icc compiler (version 14.0.2) with -openmp and -O3 flags.

The obtained results are expected, because the s6030 environment is a system designed for computing codes like this one. The next performance results were obtained using 16, 32 and 64 cores on s6030 environment and the best result were obtained using 64 cores (see Table 4.4), as we can see in Fig. 4.3.

4.3. Performance optimizations with MPI. We performed several optimizations in our sequential code using MPI. In this subsection, we considered up to 16 computational nodes composed by the resources previously described each, where we do many test until find the best option for this algorithm. For all executions we used the maximum time in each node and then calculate the average. We will select the worst time from each test to calculate the average time.

All our tests were conducted using MPI with 2, 4, 8 and 10 nodes and 1 thread in each node. Besides, we have tested using MPI with 12, 14 and 16 nodes, the maximum number of nodes that we could use at cluster, but the results that we obtained are worse and we see no need to include these in this results.

So we tried to find which are the best results by combining any number of nodes with any number of threads. Specifically, we calculated the results, and after different tests we could guarantee that the best result is achieved using one node and ten threads (see Table 4.5). In that case, we have tried various combinations with nodes and threads and we dont have used these results because the values obtained are worse than others.

We considered important to get two new times for the MPI implementation, in order to obtain a more reliable comparative evaluating the time spent in communication. We calculate the average send time (Broadcast) with the worst time that we get in total send communications. Moreover, we estimate the time of receive (Gather) and calculate the average using the worst time that we get in total receive communications.

We believe that the best result in that case is using one node because when running the code within the same node does not suffer delay by MPIBroadcast or MPIGather calls. We use MPI variables within the code

Table 4.4: Multicore scalability on a shared memory machine (Time in seconds) with five executions. The total time in a shared memory machine is improvement when we use more cores,the scalability is normal and the time improvement is small because the algorithm is highly iterative with data dependencies.

OpenMP	Total time(s)	Speed-up
16 Cores	1,4834	12,4174
32 Cores	1,4502	12,7016
64 Cores	1,3858	13,2919

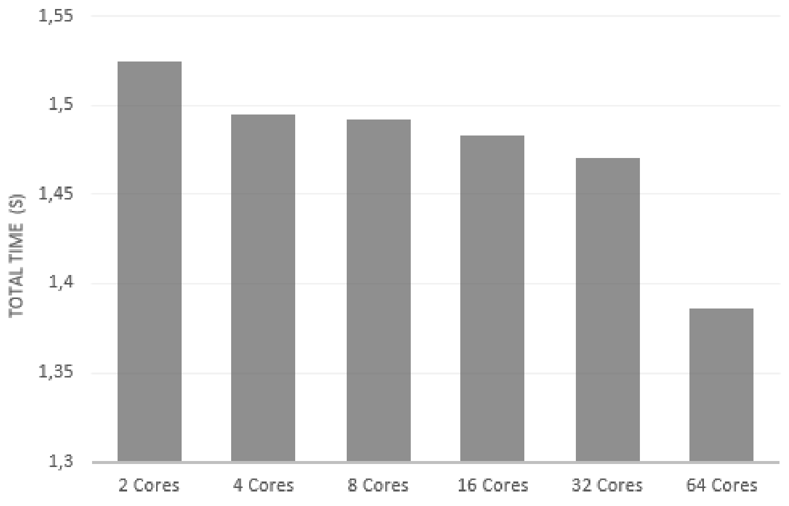

Fig. 4.3: Multicore scalability on a shared memory machine with five executions. The progress is bigger with 64cores because OpenMp use the strength of the shared memory machine.

and we are compiling it using the mpicc compiler(version 14.0.2) and -lpmi and -O3 compilation flags. It can be seen that the scalability study with the best results in Fig. 4.4. Also, we obtained send and receive times from all tests and compared them in Fig. 4.5.

As we could see, with 4, 8 and 10 nodes the results that we obtain are highly similar and this is because the ATGP algorithm is iterative and we think that we need to work with a larger volume of data. Maybe, the size of our problem (this particular hyperspectral image) is small for a such distributed memory platform.

4.4. Performance optimizations over Intel Xeon Phi^{TM}. We performed several optimizations in our OpenMP code for Xeon Phi [22, 23]. In this subsection, we considered an Intel Xeon Phi node:

- 2x Intel Xeon CPU E5-2620 v2 @ 2.10 GHz with 6 cores and 32 GB of DDR3 memory.
- Xeon Phi Card 5110P with 8 GB of DDR3 memory and 60 cores.

The performance of the OpenMP runtime can be essential for the overall scalability of OpenMP codes.

In the experiments using Intel Xeon Phi we could distinguish two modes [24]:

- *Offload Mode*: The application starts the execution on the host. As the computation proceeds it can decide to send data to the coprocessor and let that work on it and the host and the coprocessor may or may not work in parallel.
- *Native Mode*: This execution environment allows the users to view the coprocessor as another compute node. In order to run the code natively, an application has to be cross compiled for Phi operating environment.

Many experiments were done using OpenMP with Xeon Phi directives until finding the best option for this algorithm. All experiments were executed five times and the results are the average of all of them. The

Table 4.5: Total time using MPI in a cluster environment (Time in seconds) with five executions. Try to find the harmony between send time and receive time.We suspect that the best time is with 1Node and 10Threads because the receive time is minor.

MPI	Total Send (s)	Total Receive (s)	Total Time (s)	Speed-up
2 Nodes	0,0531	0,0404	4,0052	4,5990
4 Nodes	0,1050	0,0977	2,6626	6,9180
8 Nodes	0,1902	0,1645	2,5050	7,3532
10 nodes	0,1400	0,1161	2,3372	7,8812
1 Node 10 Threads	0,1558	0,0343	1,1002	16,7424

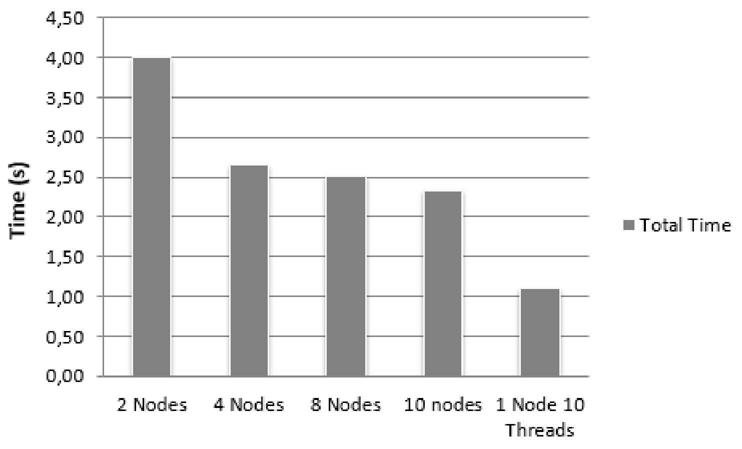

Fig. 4.4: Scalability study in a cluster environment using MPI with five executions. The total time is better in a single node because the communications spend a lot of time.

executions were done using Xeon Phi offload and native mode and results are shown in Table 4.6.

As we can see, using offload the results that we obtain are faster than using native mode. This is obvious because in native mode you need to communicate the results with the other coprocessor and work in parallel with the host, and for nested parallel regions, the overhead is much larger on the Xeon Phi system. We use OpenMP variables with Xeon Phi support within the code and we are compiling it using the mpicc compiler (version 14.0.2) and -mmic, -openmp and -O3 compilation flags.

5. Conclusions. In this paper we did several performance optimizations for the automatic target detection process (ATGP) algorithm for hyperspectral imaging. The results and parallel performance of the proposed parallel implementations, conducted using OpenMP and MPI [26], have been presented and thoroughly discussed in the context of a real defense and security application: the analysis of hyperspectral data collected by NASAs AVIRIS instrument over the World Trade Center (WTC) in New York, five days after the terrorist attacks that collapsed the two main towers in the WTC complex. From the results obtained we can conclude that the best performance is obtained with 1 node and 10 threads, using MPI. This is also the result that we expected because the ATGP algorithm is highly iterative and with high data dependency between iterations.

6. Future Research. Although the results reported in this work are very encouraging, further experiments should be conducted in order to increase the parallel performance of other versions of the proposed parallel algorithms and also optimizing the parallel design of the algorithms. We will do some research comparing this results with accelerators like GPUs (using CUDA).

In addition, we could use a heavier image and compare if the results over MPI are better than with the

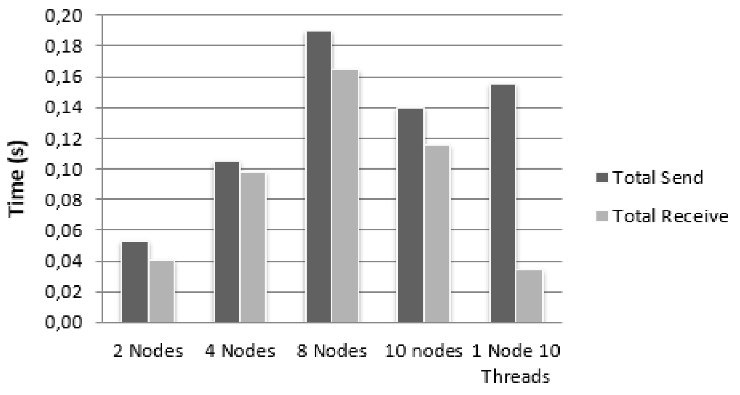

Fig. 4.5: Comparative between send and receive times in each test using MPI with five executions.In this figure we could see how the receive time is better in the same node and this is too relevant for the final total time.

Table 4.6: Total time using OpenMP over Intel Xeon Phi (Time in seconds) with five executions.

MODE	Total Time (s)	Speed-up
OFFLOAD	4,1322	4,4576
NATIVE	7,3886	2,4930

actual image.

Finally, new target detection and endmember extraction algorithms could be optimized and evaluated using different multi-core and many-core architectures.

Acknowledgment. This work was partially supported by the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

REFERENCES

[1] A. F. H. Goetz, G. Vane, J.E. Solomon and B.N. Rock,*Imaging spectrometry for Earth remote sensing*, Science 228, pp. 1147-1153, 1985.
[2] A. Plaza, J.A. Benediktsson, J.Boardman, J.Brazile, L.Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, J.Gualtieri, M. Marconcini, J.C Tilton and G. Trianni, *Recent advances in techniques for hyperspectral image processing*, Remote Sensing of Environment 113, pp. 110-122, 2009.
[3] R. O. Green, M.L Eastwwod, C.M. Sarture, T.G. Chrien, M. Aronsson, J.Chippendale, J.A. Faust, B.E. Pavri, C.J. Chovit, M. Solis, M.R. Olah and O. Williams, *Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)*, Remote Sensing of Environment 65, pp. 227-248, 1998.
[4] C.-I. Chang, *Hyperspectral imaging: Techniques for spectral detection and classification*, Kluwer Academic/Plenum Publishers: New York, 2003.
[5] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*, CRC Press: Boca Raton FL. Chapman & Hall, 2007.
[6] R. A. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing*, Academic Press: London, 1997.
[7] D. A. Landgrebe, *Signal Theory Methods in Multispectral Remote Sensing*, John Wiley & Sons, New York, NY, USA, 2003.
[8] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis: An Introduction*, Springer, 2006.
[9] C.-I.Chang, *Recent Advances in Hyperspectral Signal and Image Processing*, John Wiley & Sons, 2007.
[10] C.-I.Chang, *Hyperspectral Data Exploitation: Theory and Applications*, John Wiley & Sons, 2007.
[11] C.-I. Chang and H. Ren, *An experiment-based quantitative and comparative analysis of target detection and image classification algorithms for hyperspectral imagery*, IEEE Transactions on Geoscience and Remote Sensing 38.2, pp. 1044-1063, 2000.

[12] H. Ren and C-I.Chang, *Automatic Spectral Target Recognition in Hyperespectral Imagery*, IEEE Transactions on Aerospace and Electronic Systems 39.4, pp. 1232-1249, 2003.
[13] D. Manolakis, D. Marden and G. A. Shaw, *Hyperspectral image processing for automatic target detection applications*, MIT Lincoln Laboratory 14, pp. 79-116, 2003.
[14] A. Paz, A. Plaza and S. Blazquez, *Parallel Implementation of Target Detection Algorithms for Hyperspectral Imagery*, International Geoscience and Remote Sensing Symposium 2, pp. 589-592, 2008.
[15] Y. Tarabalka, T.V. Haavardsholm, I. Kasen and T.Skauli, *Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and gpu processing*, Journal of Real-Time Image Processing 4, pp. 1-14, 2009.
[16] S. Sanchez, A.Plaza, G.Martin and A.Plaza, *Parallel Unmixing of Remotely Sensed Hyperspectral Images on Commodity Graphics Processing Units*, Concurrency and Computation: Practice & Experience 23.13, pp. 1538-1557, 2011.
[17] A. Plaza and C.-I. Chang, *Impact of initialization on design of endmember extraction algorithms*, IEEE Transactions on Geoscience and Remote Sensing 43.11, pp. 3397-3407, 2006.
[18] A. Plaza, J. Plaza, A. Paz and S. Sanchez, *Parallel Hyperspectral Image and Signal Processing*, IEEE Signal Processing Magazine 28, pp. 119-126, 2011.
[19] R. Brightwell, L.Fisk, D. Greenberg, T. Hudson, M.Levenhagen, A. Maccabe and R. Riesen, *Massively parallel computing using commodity components*, Parallel Computing 26, pp. 243-266. 2000.
[20] 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*, The Journal of Supercomputing 40.1, pp. 81-107, 2007.
[21] 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, pp. 345-358, 2006.
[22] Schmidl et al. ,*Assessing the Performance of OpenMP Programs on the Intel Xeon Phi*, Euro-Par 2013 Parallel Processing, p.12, 2013.
[23] C. Heirman et al., *Automatic SMT threading for OpenMP applications on the Intel Xeon Phi co-processor*, 4th International Workshop on Runtime and Operating Systems for Supercomputers, Article No.7, ACM New York, NY, USA, 2014.
[24] M. Koesterke et al., *Tutorial: Programming for the Intel Xeon Phi (MIC)*, IEEE Cluster 2013.
[25] A. Paz and A. Plaza, *Clusters vs. GPUs for Parallel Automatic Target Detection in Hyperspectral Images*, EURASIP Journal of Advances in Signal Processing, 2010, Article ID 915639, 18 pages doi:10.1155/2010/915639.
[26] Y. He and Q.Ding, *MPI and OpenMP Paradigms on Cluster of SMP Architectures: The Vacancy Tracking Algorithm for Multi-Dimensional Array Transposition*, Scalable Computing: Practice and Experience, Volume 5, No 2, 2002.

Edited by: Pedro Valero Lara
Received: Sept 28, 2015
Accepted: Febr 25, 2016