GPU Implementation of the Pixel Purity Index Algorithm for Hyperspectral Image Analysis

Sergio Sánchez and Antonio Plaza
Hyperspectral Computing Laboratory
Department of Technology of Computers and Communications
University of Extremadura, Avda. de la Universidad s/n
E-10071 Caceres, Spain
Email: {sersanmar, aplaza}@unex.es

***Abstract*—Hyperspectral imaging is a new technique in remote sensing that generates images with hundreds of spectral bands, at different wavelength channels, for the same area on the surface of the Earth. The price paid for such a wealth of spectral information is the enormous amounts of data to be processed. In recent years, several efforts have been directed towards the incorporation of high-performance computing models in remote sensing missions. For this purpose, graphics processing units (GPUs) have emerged as a very interesting type of hardware architecture in hyperspectral image processing due to its low weight and compact size, which allows for on-board data processing. In this paper, we develop an innovative GPU implementation of a standard hyperspectral image processing algorithm called pixel purity index (PPI) and utilized, among others, in commercial software tools such as ITTVIS Environment for Visualizing Images (ENVI) software originally developed by Analytical Imaging and Geophysics (AIG), one of the most popular tools currently available for processing remotely sensed data. The algorithm has been implemented using the compute device unified architecture (CUDA), and tested on the NVidia Tesla C1060 architecture, achieving a significant performance increase in the analysis of both synthetic and real hyperspectral data.**

***Index Terms*—Hyperspectral data, pixel purity index, graphics processing units (GPUs).**

I. Introduction

Hyperspectral imaging instruments are capable of collecting hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth [1]. For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS), 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 [2]. The resulting multidimensional data cube typically comprises several Gigabytes per flight. Figure 1 illustrates the concept of hyperspectral imaging. The wealth of spectral information provided by latest-generation hyperspectral sensors has opened ground-breaking perspectives in many applications [3]. One of the main problems in the analysis of hyperspectral data cubes is the presence of mixed pixels [4], [5], which arise when the spatial resolution of the sensor is not enough to separate spectrally distinct materials [6]. For instance, the pixel vector labeled as "vegetation" in Figure 1 may actually be a mixed pixel comprising a mixture of vegetation and soil, or different types of soil and vegetation canopies. In this case, several spectrally pure signatures (*endmembers*) are combined into the same (mixed) pixel. Spectral unmixing [7] is one of the most popular techniques to analyze hyperspectral data. It comprises two stages: 1) identification of endmembers; and 2) estimation of the abundance of each endmember in each pixel. The unmixing process is computationally expensive due to the high dimensionality of hyperspectral data cubes [8], [9].

Spectral unmixing involves the separation of a pixel spectrum into its pure component endmember spectra, and the estimation of the abundance value for each endmember [10]. Let us assume that a remotely sensed hyperspectral scene with N bands is denoted by $\mathbf{F}$, in which a pixel at discrete spatial coordinates is represented by a vector $\mathbf{f} = [f_1, f_2, \cdots, f_N]$, where f_k denotes the spectral response at the k-th wavelength, with $k = 1, \ldots, N$. Under the linear mixture model assumption, each pixel vector in the original scene can be modeled using the following expression:

$$\mathbf{f} = \sum_{i=1}^{p} a_i \cdot \mathbf{e}_i + \mathbf{n}, \tag{1}$$

where $\mathbf{e}_i$ designates the i-th pure spectral component (endmember) residing in the pixel, a_i is a scalar value designating the fractional abundance of the endmember $\mathbf{e}_i$ at the pixel $\mathbf{f}$, p is the total number of endmembers, and $\mathbf{n}$ is a noise vector. The solution of the linear spectral mixture problem described in Eq. (1) involves: 1) identifying a collection of $\{\mathbf{e}_i\}_{i=1}^{p}$ endmembers in the image, and 2) estimating their abundance in each pixel. Several techniques have been proposed for such purposes, but all of them are very expensive in computational terms. Although these techniques map nicely to high performance computing systems such as commodity clusters [11], [12], [13], [14], [8], these systems are difficult to adapt to on-board processing requirements introduced by applications such as wild land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contami-

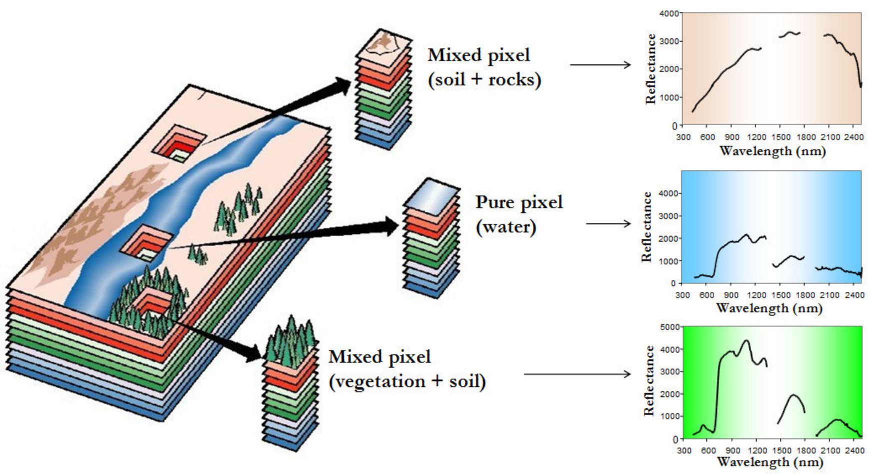


Figure 1. Presence of mixed pixels in remotely sensed hyperspectral images.

nation. In those cases, low-weight integrated components such as graphics processing units (GPUs) offer tremendous potential to bridge the gap towards real-time analysis of remotely sensed hyperspectral data, i.e. the data can be processed at the same time as it is collected using a compact hardware device [15], [16], [17].

In this paper, we develop a new GPU implementation of the well-known pixel purity index (PPI) algorithm [18], [19] for endmember extraction in hyperspectral images. Despite the fact that this algorithm is available in ITTVIS[1] Environment for Visualizing Images (ENVI) software originally developed by Analytical Imaging and Geophysics (AIG), which is one of the most popular tools currently available for processing remotely sensed data, this algorithm has never been implemented in GPUs in the past. The proposed PPI implementation has been tested using NVidia's compute device unified architecture (`CUDA`), and implemented on a NVidia Tesla C1060, achieving very significant speedups when compared to an optimized implementation of the same code in a standard multiple-core CPU. The remainder of the paper is organized as follows. Section II describes the PPI algorithm. Section III describes its parallel implementation on the GPU. Section IV conducts an experimental validation of the proposed GPU implementation (in terms of analysis accuracy and computational performance) using both synthetic and real hyperspectral data. Finally, section V concludes with some remarks and hints at plausible future research.

II. Pixel Purity Index Algorithm

The pixel purity index (PPI) algorithm [18] has been widely used in hyperspectral image analysis for endmember extraction due to its publicity and availability in ENVI software. The algorithm searches for a set of vertices of a convex hull in a given dataset, which are supposed to be pure signatures present in the data. Due to its propriety and limited published results, its detailed implementation has never been made publicly available. Since the details of the specific steps to implement ENVI's PPI are not available in the literature, the PPI algorithm described below is only based on the limited published results and our own interpretation [19]. Nevertheless, except a step, step 4), both our approximation and the PPI in ENVI 4.0 produce exactly the same results. The inputs to the algorithm are a hyperspectral data cube $\mathbf{F}$ with N dimensions; the number of random skewers to be generated during the process, K; and a cut-off threshold value, t_v, used to select as final endmembers only those pixels that have been selected as extreme pixels at least t_v times throughout the PPI process.

The algorithm is given by the following steps:

1) *Skewer generation*. Produce a set of K randomly generated unit vectors $\{\mathbf{skewer}_j\}_{j=1}^{K}$.

2) *Extreme projections*. For each $\mathbf{skewer}_j$, $j = \{1, \cdots, K\}$, all pixel vectors $\mathbf{f}_i$ in the original data set $\mathbf{F}$ are projected onto $\mathbf{skewer}_j$ via dot products of $\mathbf{f}_i \cdot \mathbf{skewer}_j$ to find sample vectors at its extreme (maximum and minimum) projections, thus forming an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}(\mathbf{skewer}_j)$ (see Fig. 2). Despite the fact that different skewers generate different extrema sets, it is very likely that some sample vectors may appear in more than one extrema set. To account for this, we define an indicator function of a set $\mathbf{F}$, denoted by $I_S(\mathbf{x})$, to denote membership of an element $\mathbf{x}$ to that particular set as follows:

$$I_S(\mathbf{x}) = \begin{Bmatrix} 1 \text{ if } \mathbf{x} \in S \\ 0 \text{ if } \mathbf{x} \notin S \end{Bmatrix} \quad (2)$$

3) *Calculation of PPI scores*. Using the indicator function above, we calculate the PPI score associated to each pixel vector $\mathbf{f}_i$ (i.e., the number of times that given pixel has been selected as extreme in step 2) using the following equation:

[1]http://www.ittvis.com

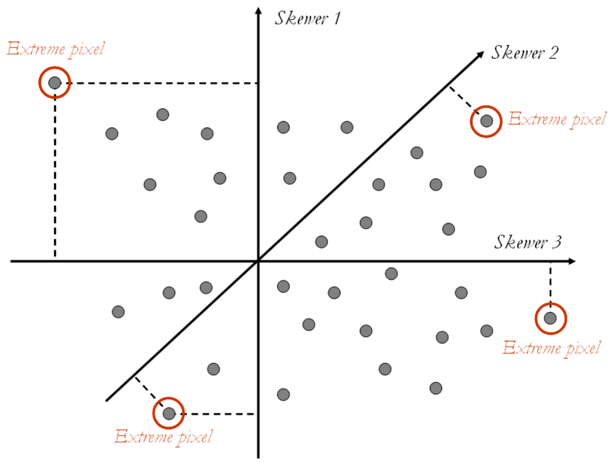


Figure 2. Toy example illustrating the performance of the PPI endmember extraction algorithm in a 2-dimensional space.

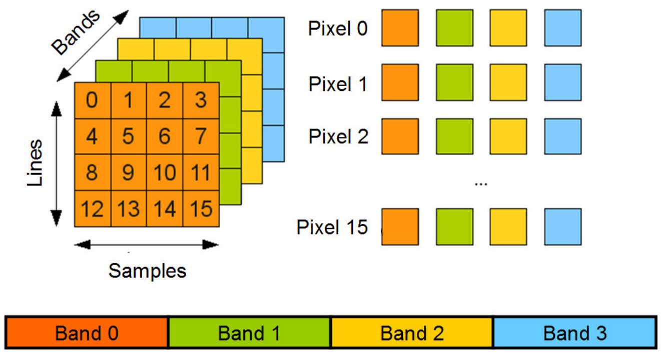


Figure 3. Storage of a hyperspectral image in the GPU memory.

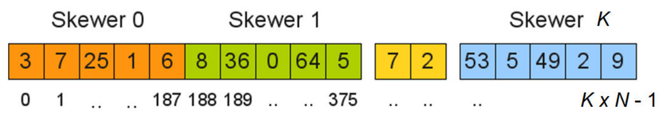


Figure 4. Skewer generation and storage in the GPU.

$$N_{PPI}(\mathbf{f}_i) = \sum_{j=1}^{k} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i) \tag{3}$$

4) *Endmember selection*. Find the pixel vectors with scores of $N_{PPI}(\mathbf{f}_i)$ which are above t_v and label them as spectral endmembers. An optional post-processing (not implemented in this work) based on removing potentially redundant endmembers may be also applied.

The most time consuming stage of the PPI algorithm is stage 2 (extreme projections). For example, the PPI algorithm available in ENVI 4.0 version took more than 50 minutes of computation to project every data sample vector of a hyperspectral image with 614×512 pixels (the standard number of pixels produced by NASA's AVIRIS instrument in a single frame, each with 224 spectral bands) using 10^4 skewers in a PC with AMD Athlon 2.6 GHz processor and 512 MB of RAM. Fortunately, the PPI algorithm is well suited for parallel implementation. The computation of skewer projections are independent and can be performed simultaneously, leading to many ways of parallelization.

III. GPU Implementation

In order to implement the PPI algorithm in a GPU, the first issue that needs to be addressed is how to map a hyperspectral image onto the memory of the GPU. The image is stored in band-sequential format, i.e. the hyperspectral image file first storages the first band, then the second, and so on (see Fig. 3). Since the size of hyperspectral images may exceed the capacity of the GPU memory, in that case we split the image into multiple spatial-domain partitions [8] made up of entire pixel vectors. In the following, we assume that a hyperspectral pixel vector is made up of $N = 188$ components, which is the number of values that we have retained after eliminating the noisy and water absorption bands in our test hyperspectral images. With this partitioning strategy in mind, our GPU implementation of the PPI algorithm (called GPU-PPI hereinafter) can be summarized by the following steps:

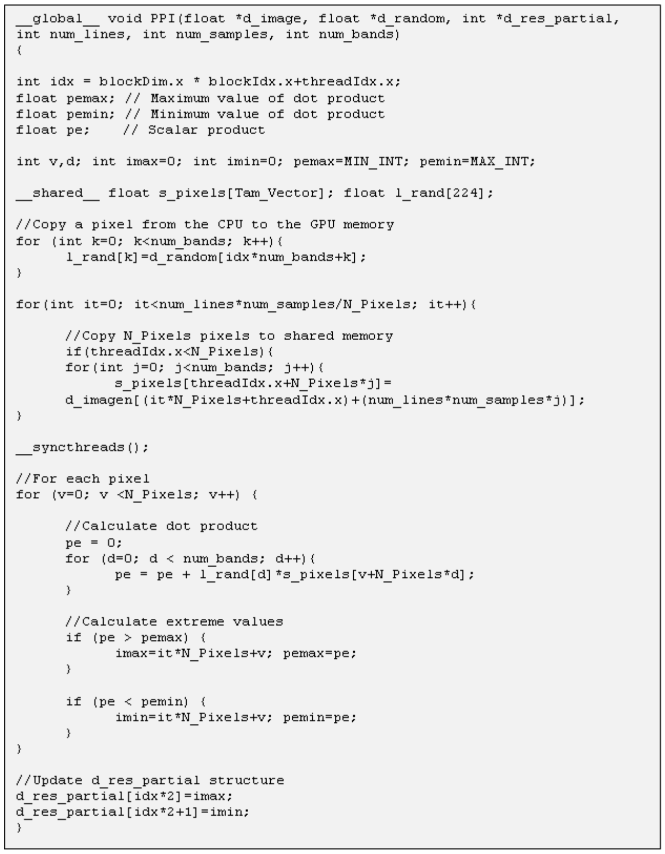
A. Skewer generation

The skewers are N-dimensional vectors in which the components are generated randomly. Since the number of skewers is generally in the order of $K = 10^4$ or more, the total number of random numbers needed $K \times N$ is generally very high and an efficient strategy for random number generation in the GPU is needed. One of the most widely respected methods for random number generation in software is the Mersenne twister, which has extremely good statistical quality. In this work, we have adopted this method and used the `RandomGPU` function provided by CUDA to generate the random values needed to simulate the skewers. For instance, if we assume that $N = 188$, this function allowed us to generate more than $K = 10^4$ skewers in less than 10 milliseconds, which is widely acceptable in our application context. For illustrative purposes, Fig. 4 illustrates the procedure for generating a set of skewers $\{\mathbf{skewer}_j\}_{j=1}^{K}$, each with N components, and how these skewers are stored in the GPU for subsequent processing.

B. Extreme projections and calculation of PPI scores

Once the skewer generation process is completed, the next step is the projection of all pixel vectors $\mathbf{f}_i$ in the original data set $\mathbf{F}$ onto each $\mathbf{skewer}_j$ via dot products of $|\mathbf{f}_i \cdot \mathbf{skewer}_j|$ to find sample vectors at its extreme (maximum and minimum) projections. For this purpose, we have developed a CUDA kernel (see Fig. 5) in which each thread performs the projection of all pixel vectors onto a skewer. The parameters of this kernel can be summarized as follows:

- The first parameter of the kernel (`d_image`) is the hyperspectral image in the storage format described in Fig. 3.
- The second parameter is the structure that contains the generated skewers, in the storage format described in Fig. 4.



```
__global__ void PPI(float *d_image, float *d_random, int *d_res_partial,
int num_lines, int num_samples, int num_bands)
{

int idx = blockDim.x * blockIdx.x+threadIdx.x;
float pemax; // Maximum value of dot product
float pemin; // Minimum value of dot product
float pe;    // Scalar product

int v,d; int imax=0; int imin=0; pemax=MIN_INT; pemin=MAX_INT;

__shared__ float s_pixels[Tam_Vector]; float l_rand[224];

//Copy a pixel from the CPU to the GPU memory
for (int k=0; k<num_bands; k++){
      l_rand[k]=d_random[idx*num_bands+k];
}

for(int it=0; it<num_lines*num_samples/N_Pixels; it++){

      //Copy N_Pixels pixels to shared memory
      if(threadIdx.x<N_Pixels){
      for(int j=0; j<num_bands; j++){
            s_pixels[threadIdx.x+N_Pixels*j]=
      d_imagen[(it*N_Pixels+threadIdx.x)+(num_lines*num_samples*j)];
}

__syncthreads();

//For each pixel
for (v=0; v <N_Pixels; v++) {

      //Calculate dot product
      pe = 0;
      for (d=0; d < num_bands; d++){
            pe = pe + l_rand[d]*s_pixels[v+N_Pixels*d];
      }

      //Calculate extreme values
      if (pe > pemax) {
            imax=it*N_Pixels+v; pemax=pe;
      }

      if (pe < pemin) {
            imin=it*N_Pixels+v; pemin=pe;
      }
}

//Update d_res_partial structure
d_res_partial[idx*2]=imax;
d_res_partial[idx*2+1]=imin;
}
```

Figure 5. CUDA Kernel developed to implement the extreme projections step of the PPI algorithm on the GPU.

- The third parameter (`d_res_partial`) is the strucuture in which the output of the CUDA kernel will be stored.
- The kernel also receives as input parameters the dimensions of the hyperspectral image, i.e. `num_lines`, `num_samples` and `num_bands`.

The kernel uses a structure `l_rand` (local to each thread) in which a skewer is stored. It should be noted that each thread works with a different skewer and the values of the skewer are continuously used by the thread in each iteration, hence it is reasonable to store each skewer in the local memory associated to each thread since this local memory is hundreds of times faster than the global GPU memory.

The second structure used by the kernel is `s_pixels`, which is shared by all threads. Since each thread needs to perform the dot product using the same image pixels, it is reasonable to make use of a shared structure which can be accessed by all threads, thus avoiding that such threads access the global GPU memory separately. Since the `s_pixels` structure is also stored in local memory, the accesses are also much faster. It should be noted that, in our implementation, the `s_pixels` structure can only allocate 10 pixels due to the reduced size of the local memory, hence all hyperspectral image pixels are processed in blocks of 10. Once the skewer and a group of 10 pixels are stored in the local memory associated to a thread, the next step is to perform the dot product between each pixel and the skewer.

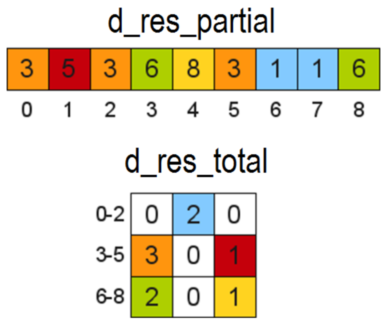


Figure 6. Generation of a final PPI image from the structures used in the CUDA kernel described in Fig. 5.

Each thread uses two variables `pemin` and `pemax` which store the minima and maxima projection values, respectively, and other two variables `imin` and `imax` which store the relative index of the pixels resulting in the maxima and minima projection values. Once the projection process is finalized for a group of 10 pixels, another group is loaded and the process finalizes when all hyperspectral image pixels have been processed. Finally, the `d_res_partial` structure is updated with the minima and maxima projection values. This structure is reshaped into a PPI score image by simply counting the number of times that each pixel was selected as extreme during the process and updating its associated PPI score with such number, this producing a final structure `d_res_total` that stores the final PPI image $N_{PPI}(\mathbf{f}_i)$. This situation is graphically illustrated by an example in Fig. 6.

To conclude this section, it should be noted that the *end-member selection* stage of the PPI can be performed in the form of a simple thresholding of the PPI image obtained in the previous stage, although in some cases a more refined procedure is needed to avoid the selection of duplicate endmembers, which can be achieved using the orthogonal subspace projection algorithm [20]. However, in this work we have mainly focused on optimizing the original PPI algorithm as described in the ITTVIS Environment for Visualizing Images (ENVI) and, hence, we leave the development of parallel implementations for subsequent optimizations of the algorithm, such as those described in [19], for further developments.

IV. Experimental Results

A. The NVidia Tesla C1060 Architecture

The proposed GPU implementation has been tested on a Nvidia Tesla C1060 GPU, which features 240 processor cores operating at 1.296 Ghz, with single precision floating point performance of 933 Gflops, double precision floating point performance of 78 Gflops, total dedicated memory of 4 GB, 800 MHz memory (with 512-bit GDDR3 interface) and memory bandwidth of 102 GB/sec[2]. The GPU is connected to an Intel core i7 920 CPU at 2.67 Ghz with 8 cores, which uses a motherboard Asus P6T7 WS SuperComputer.

[2]http://www.nvidia.com/object/product_tesla_c1060_us.html

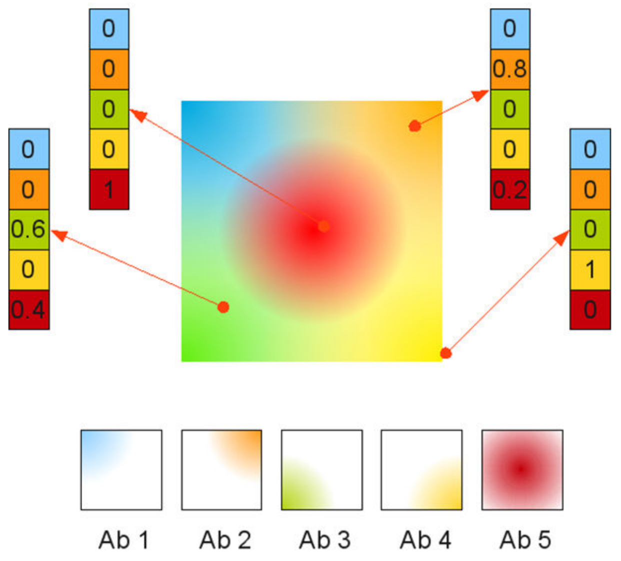


Figure 7. Synthetic hyperspectral image used in experiments.

It is important to emphasize that our GPU version of PPI provides exactly the same results as the serial version of the same algorithm, implemented using the Intel C/C++ compiler and optimized using several compilation flags to exploit data locality and avoid redundant computations. Hence, the only difference between the serial and parallel algorithms is the time they need to complete their calculations.

B. Results with synthetic data

First, we evaluate the parallel performance of our GPU version of the PPI algorithm in the NVidia Tesla C1060 GPU using a synthetic hyperspectral image. The synthetic image was created using five pure spectral signatures (endmembers) obtained from the U.S. Geological Survey (USGS) mineral library available online[3]. The pure signatures were arranged at the vertices of the image and at the central pixel. Signature abundance decreases linearly away from the specified points, as shown in Fig. 7. The five abundance maps (one per endmember) displayed in Fig. 7 have been assigned so that the contributions of the components add to one for every pixel. The resulting image, with a size of 100×100 pixels, contains pure pixels, but mixtures of 2 and 3 endmembers per pixel. Random noise was added to the synthetic scene above to simulate contributions from ambient (clutter) and instrumental sources [2]. Specifically, white gaussian noise was created by using numbers with a standard normal distribution obtained from a random number generator and added to each pixel. For the simulations, we consider a signal-to-noise ratio (SNR) in the synthetic hyperspectral scene of 30:1 [20].

For illustrative purposes, Fig. 8 visually represents the PPI images produced by the CPU and the GPU versions of the PPI algorithm for the synthetic hyperspectral image in Fig. 7 with a high proportion of noise (SNR=30:1). In both cases, the number of skewers used in experiments was $K = 15360$ (this number was carefully adjusted to obtain the best compromise results). As shown by Fig. 8, the results obtained by both implementations are highly correlated. The execution time measured after applying the CPU version of the PPI algorithm to the considered synthetic scene on the Intel Core i7 920 CPU (using just one of the available cores) was 244.55 seconds, while the execution time measured on the GPU was only 1.82 seconds. This means that a significant speedup of 133.85x was observed when comparing the processing time measured in the GPU with the processing time measured in the CPU. The C function `clock()` was used for timing the CPU implementation, and the CUDA timer was used for the GPU implementation. For illustrative purposes, Fig. 9 shows the percentage of the total execution time consumed by the main PPI kernel and by the `RandomGPU` kernel used for generating skewers, along with the number of times that each kernel was invoked (in the parentheses). These values were obtained after profiling the implementation using the CUDA Visual Profiler tool. In the figure, the percentage of time for data movements from host (CPU) to device (GPU) and from device to host. It should be noted that two data movements from host to device are needed to transfer the original hyperspectral image and the skewers to the GPU, while only one movement from device to host (final PPI image) is needed. As shown by Fig. 9, the PPI kernel consumes about 99% of the total GPU time, while the `RandomGPU` kernel compraratively occupies a much smaller fraction. Finally, the data movement operations are not significant, which indicates that most of the GPU processing time is invested in the most time-consuming operation, i.e. the calculation of skewer projections and identification of maxima and minima projection values leading to endmember identification.

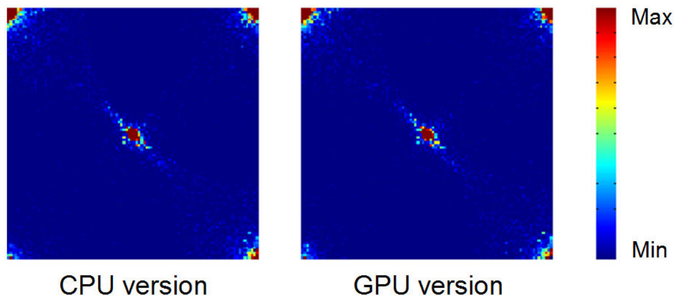


Figure 8. Visual comparison between the PPI images produced by the CPU and the GPU versions of the PPI algorithm for the synthetic hyperspectral image in Fig. 7 with noise.

C. Results with real data

The real hyperspectral scene used in our experiments is the well-known AVIRIS Cuprite data set, available online in reflectance units[4]. This scene has been widely used to validate the performance of endmember extraction algorithms. The portion used in experiments corresponds to a 350×350-pixel subset of the sector labeled as f970619t01p02_r02_sc03.a.rfl

[3]http://speclab.cr.usgs.gov

[4]http://aviris.jpl.nasa.gov/html/aviris.freedata.html

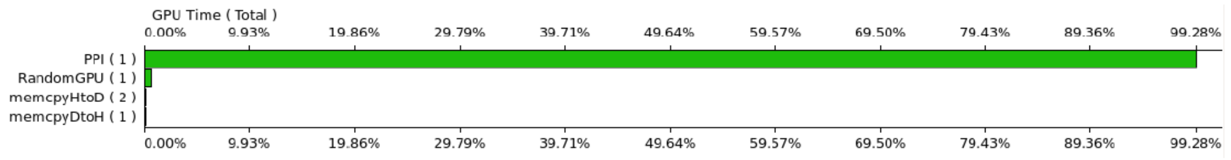


Figure 9. Summary plot describing the percentage of the total GPU time consumed by the different kernels used in the implementation of the PPI in the NVidia Tesla C1060 GPU after processing the synthetic image in Fig. 7.

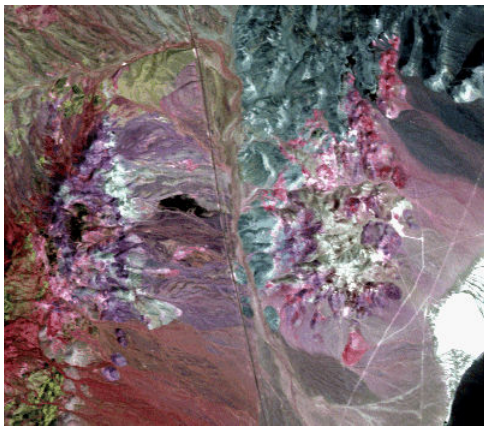


Figure 10. False color composition of the AVIRIS Cuprite scene used in experiments.

in the online data. The scene comprises 224 spectral bands between 0.4 and 2.5 μm, with nominal spectral resolution of 10 nm. Prior to the analysis, bands 1–3, 107-114, 153–169, and 221–224 were removed due to water absorption and low SNR in those bands. For illustrative purposes, Fig. 10 shows a false color composition of this scene, which is well understood mineralogically, and has several exposed minerals of interest including those used in the USGS library considered for the generation of the synthetic data set analyzed in the previous subsection.

In our analysis of the real scene, the number of skewers used in experiments was again set to $K = 15360$. In experiments we observed that the results obtained by both implementations (CPU and GPU) and the commercial ENVI software package were almost identical, hence we do not display the resulting PPI images here. The execution time measured after applying the CPU version of the PPI algorithm to the considered real scene on the Intel Core i7 920 CPU (using just one of the available cores) was 3454.55 seconds, while the execution time measured on the GPU was only 17.59 seconds. This means that a significant speedup of 196.35x was observed when comparing the processing time measured in the GPU with the processing time measured in the CPU. In addition, we also calculated the speedup of our proposed GPU implementation with the PPI algorithm available in the commercial ENVI software package in the same computing environment. It should be noted that the PPI available in ENVI applies several optimizations which include calculation of more than two (i.e. maxima and minima) extreme projections per skewer to save computation time. However, these optimizations are not disclosed and/or documented. In our experiments on the Intel Core i7 920 CPU, the PPI available from ENVI software took 1078.03 seconds to process the AVIRIS Cuprite image using the same number of skewers. This means that the speedup achieved by our GPU implementation with regards to ENVI's PPI was 61.28x. In all cases, the speedup factors achieved were highly relevant, with similar percentage of the total execution time consumed by the main PPI kernel and by the `RandomGPU` kernel as those already reported for the synthetic scene in Fig. 9. This indicates that, as it was already the case in the processing experiments reported for the synthetic scene, most of the GPU processing time is invested in the most time-consuming operations of the PPI, i.e. the calculation of skewer projections and identification of maxima and minima projection values.

However, it is also important to emphasize that the obtained results are not strictly in real time. Since the cross-track line scan time in AVIRIS, a push-broom instrument, is quite fast (8.3 msec to collect 512 full pixel vectors). This introduces the need to process the considered scene in approximately 5 seconds to fully achieve real-time performance. Despite the fact that the current implementation of PPI could only achieve processing times around 17 seconds for the considered AVIRIS real scene, we believe that further optimizations for PPI are possible in future developments by making a better use of GPU computing resources and reducing the granularity of the problem.

Summarizing, our GPU implementation of the PPI algorithm and the experimental results reported in this paper indicate that remotely sensed hyperspectral imaging can greatly benefit from the design of new algorithms and, most importantly, from the development of efficient implementations of such algorithms in specialized hardware devices for better exploitation of high-dimensional data sets, such as those provided by NASA Jet Propulsion Laboratory's AVIRIS instrument. In this case, significant speedups can be obtained using only one GPU device, with very few on-board restrictions in terms of cost, power consumption and size, which are important when defining mission payload in remote sensing missions (defined as the maximum load allowed in the airborne or satellite platform that carries the imaging instrument).

V. Conclusions and future research lines

The ever increasing spatial and spectral resolutions that will be available in the new generation of hyperspectral instruments for remote observation of the Earth anticipates significant improvements in the capacity of these instruments to uncover spectral signals in complex real-world analysis scenarios. Such capacity demands parallel processing techniques which can cope with the requirements of time-critical applications and properly scale with image size, dimensionality and complexity. In order to address such need, in this paper we have developed a new GPU implementations of the well-known PPI algorithm for endmember extraction, which is part of the spectral unmixing chain commonly applied to extract relevance information from hyperspecral scenes. The performance of the proposed parallel algorithm has been evaluated (in terms of the quality of the solutions they provide and their parallel performance) using both synthetic and real hyperspectral data sets collected by NASA Jet Propulsion Laboratory's AVIRIS instrument. The experimental results reported in this paper indicate that remotely sensed hyperspectral imaging can greatly benefit from the development of efficient implementations of endmember extraction algorithms such as the PPI in specialized hardware devices for better exploitation of high-dimensional data sets. In this case, significant speedups are obtained using only one GPU device, with few on-board restrictions in terms of cost and size, which are important when defining mission payload in remote sensing missions. Although the proposed implementation is not strictly in real-time, we expect to achieve this consideration in future developments as it is already the case with other endmember extraction algorithms that we implemented in similar GPU architectures, which are very close to perform in real-time mode. In order to fully substantiate the aforementioned remarks, further experimentation with additional hyperspectral scenes and GPU architectures is highly desirable. Another issue to be explored in future work is the comparison of GPU implementations of hyperspectral imaging algorithms such as the PPI with field programmable gate array (FPGA) implementations of the same algorithms, particularly given the fact that advances if certification of FPGAs for space operation have been more notorious than with GPUs, which are now starting to be seriously considered for airborne and spaceborne Earth observation missions but which still have to deal with some important challenges such as their higher power consumption, radiation-hardening issues, etc. Further implementations in multi-GPU systems are also currently being explored at our laboratory to enhance the PPI and other hyperspectral imaging algorithms in terms of both analysis accuracy and computational performance.

VI. Acknowledgement

This work has been supported by the European Community's Marie Curie Research Training Networks Programme under reference MRTN-CT-2006-035927, Hyperspectral Imaging Network (HYPER-I-NET). This work has also been supported by the Spanish Ministry of Science and Innovation (HYPERCOMP/EODIX project, reference AYA2008-05965-C04-02). Sergio Sánchez is sponsored by research a fellowship with reference PTA2009-2611-P, associated to the aforementioned project. Funding from Junta de Extremadura (local government) under project PRI09A110 is also gratefully acknowledged. The authors thank Robert O. Green at NASA/JPL for making the AVIRIS Cuprite scene available.

References

[1] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for Earth remote sensing," *Science*, vol. 228, pp. 1147–1153, 1985.

[2] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment*, vol. 65, no. 3, pp. 227–248, 1998.

[3] 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*, vol. 113, pp. 110–122, 2009.

[4] J. B. Adams, M. O. Smith, and P. E. Johnson, "Spectral mixture modeling: a new analysis of rock and soil types at the Viking Lander 1 site," *Journal of Geophysical Research*, vol. 91, pp. 8098–8112, 1986.

[5] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 3, pp. 650–663, 2004.

[6] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification.* Norwell, MA: Kluwer, 2003.

[7] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Processing Magazine*, vol. 19, no. 1, pp. 44–57, 2002.

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

[9] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing.* Taylor & Francis: Boca Raton, FL, 2007.

[10] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 9, pp. 2025–2041, 2002.

[11] 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*, vol. 22, pp. 2171–2191, 2001.

[12] T. El-Ghazawi, S. Kaewpijit, and J. L. Moigne., "Parallel and adaptive reduction of hyperspectral data to intrinsic dimensionality," *Cluster Computing*, vol. 1, pp. 102–110, 2001.

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

[14] S. Tehranian and Y. Zhao, "A robust framework for real-time distributed processing of satellite data," *Journal of Parallel and Distributed Computing*, vol. 66, pp. 403–418, 2006.

[15] J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geosci. Remote Sens. Lett.*, vol. 43, pp. 441–445, 2007.

[16] 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*, vol. 4, pp. 1–14, 2009.

[17] A. Paz and A. Plaza, "Gpu implementation of target and anomaly detection algorithms for remotely sensed hyperspectral image analysis," *SPIE Optics and Photonics, Satellite Data Compression, Communication, and Processing Conference*, 2010.

[18] J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping Target Signatures Via Partial Unmixing of Aviris Data," *Proc. JPL Airborne Earth Sci. Workshop*, pp. 23–26, 1995.

[19] C.-I. Chang and A. Plaza, "A fast iterative algorithm for implementation of pixel purity index," *IEEE Geoscience and Remote Sensing Letters*, vol. 3, no. 1, pp. 63–67, 2006.

[20] J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection approach," *IEEE Trans. Geosci. Remote Sens.*, vol. 32, pp. 779–785, 1994.