GPU Implementation of Fully Constrained Linear Spectral Unmixing for Remotely Sensed Hyperspectral Data Exploitation

Sergio Sánchez[a], Gabriel Martín[a], Antonio Plaza[a] and Chein-I Chang[b]

[a]Hyperspectral Computing Laboratory
Department of Technology of Computers and Communications
University of Extremadura, Avda. de la Universidad s/n
10071 Cáceres, Spain

[b]Remote Sensing Signal and Image Processing Laboratory
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250, USA

ABSTRACT

Spectral unmixing is an important task for remotely sensed hyperspectral data exploitation. The spectral signatures collected in natural environments are invariably a mixture of the pure signatures of the various materials found within the spatial extent of the ground instantaneous field view of the imaging instrument. Spectral unmixing aims at inferring such pure spectral signatures, called endmembers, and the material fractions, called fractional abundances, at each pixel of the scene. A standard technique for spectral mixture analysis is linear spectral unmixing, which assumes that the collected spectra at the spectrometer can be expressed in the form of a linear combination of endmembers weighted by their corresponding abundances, expected to obey two constraints, i.e. all abundances should be non-negative, and the sum of abundances for a given pixel should be unity. Several techniques have been developed in the literature for unconstrained, partially constrained and fully constrained linear spectral unmixing, which can be computationally expensive (in particular, for complex high-dimensional scenes with a high number of endmembers). In this paper, we develop new parallel implementations of unconstrained, partially constrained and fully constrained linear spectral unmixing algorithms. The implementations have been developed in programmable graphics processing units (GPUs), an exciting development in the field of commodity computing that fits very well the requirements of on-board data processing scenarios, in which low-weight and low-power integrated components are mandatory to reduce mission payload. Our experiments, conducted with a hyperspectral scene collected over the World Trade Center area in New York City, indicate that the proposed implementations provide relevant speedups over the corresponding serial versions in latest-generation Tesla C1060 GPU architectures.

Keywords: Hyperspectral imaging, spectral unmixing, parallel computing, graphics processing units (GPUs).

1. INTRODUCTION

Spectral mixture analysis (also called *spectral unmixing*) has been an alluring exploitation goal from the earliest days of hyperspectral imaging[1] to our days.[2,3] No matter the spatial resolution, the spectral signatures collected in natural environments are invariably a mixture of the signatures of the various materials found within the spatial extent of the ground instantaneous field view of the imaging instrument.[4] For instance, it is likely that the pixel collected over a vegetation area in Fig. 1 actually comprises a mixture of vegetation and soil. In this case, the measured spectrum may be decomposed into a combination of pure spectral signatures of soil and vegetation, weighted by areal coefficients that indicate the proportion of each *macroscopically* pure signature in the mixed pixel.[5] The availability of hyperspectral imagers with a number of spectral bands that exceeds

Send correspondence to Antonio J. Plaza:
E-mail: aplaza@unex.es; Telephone: +34 927 257000 (Ext. 51662); URL: http://www.umbc.edu/rssipl/people/aplaza

Satellite Data Compression, Communications, and Processing VI, edited by Bormin Huang, Antonio J. Plaza, Joan Serra-Sagristà, Chulhee Lee, Yunsong Li, Shen-En Qian, Proc. of SPIE Vol. 7810, 78100G
 · doi: 10.1117/12.860775

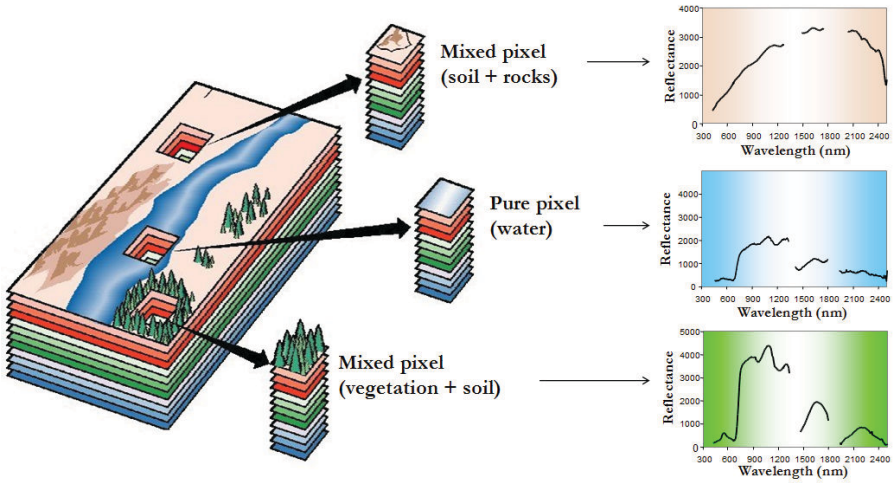


Figure 1. The mixture problem in remotely sensed hyperspectral data analysis.

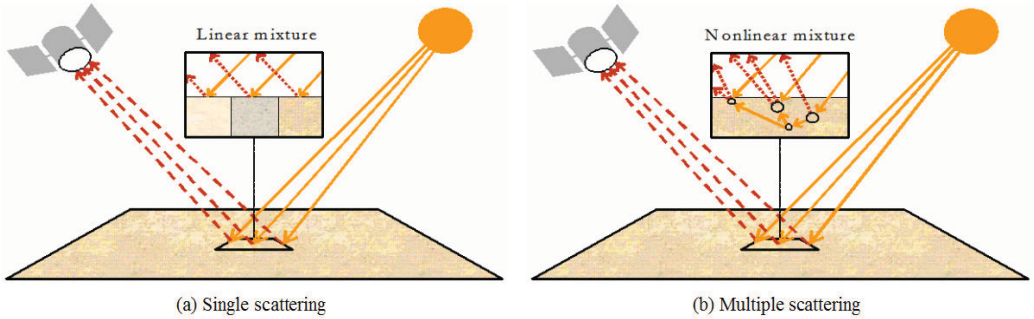


Figure 2. Linear versus nonlinear mixture models: single versus multiple scattering.

the number of spectral mixture components[6] has allowed to cast the unmixing problem in terms of an overdetermined system of equations in which, given a set of pure spectral signatures (called *endmembers*) the actual unmixing to determine apparent pixel *abundance fractions* can be defined in terms of a numerical inversion process.[7]

A standard technique for spectral mixture analysis is *linear* spectral unmixing,[8,9] which assumes that the collected spectra at the spectrometer can be expressed in the form of a linear combination of endmembers weighted by their corresponding abundances. It should be noted that the linear mixture model assumes minimal secondary reflections and/or multiple scattering effects in the data collection procedure, and hence the measured spectra can be expressed as a linear combination of the spectral signatures of materials present in the mixed pixel [see Fig. 2(a)]. Although the linear model has practical advantages such as ease of implementation and flexibility in different applications,[10] *nonlinear* spectral unmixing may best characterize the resultant mixed spectra for certain endmember distributions, such as those in which the endmember components are randomly distributed throughout the field of view of the instrument.[11,12] In those cases, the mixed spectra collected at the imaging instrument is better described by assuming that part of the source radiation is multiply scattered before being collected at the sensor [see Fig. 2(b)].

Although linear spectral unmixing is generally more tractable in computational terms than nonlinear unmixing, it is still a quite computationally expensive process due to the extremely high dimensionality of hyperspectral data cubes. Solving the mixture model generally involves: 1) identifying a collection of endmembers in the image, and 2) estimating their abundance in each pixel. The standard hyperspectral unmixing chain is graphically illustrated by a flowchart in Figure 3. In our work, we do not include the dimensionality reduction step in Fig. 3, which is mainly intended to reduce processing time but often discards relevant information in the spectral

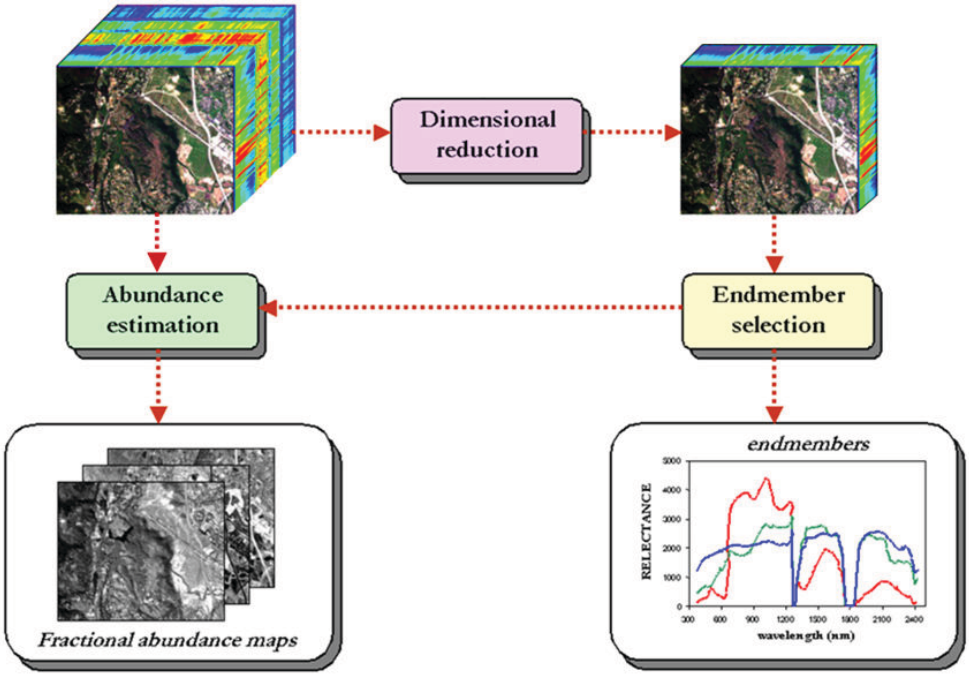


Figure 3. Standard hyperspectral unmixing chain.

domain. Several techniques have been proposed for the endmember extraction and abundance estimation steps of the unmixing chain, but all of them are expensive in computational terms. Although these techniques map nicely to high performance computing systems such as commodity clusters, 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 contamination. In those cases, low-weight integrated components are essential to reduce mission payload. In this regard, the emergence of graphical processing units (GPUs) now offers a tremendous potential to bridge the gap towards real-time analysis of remotely sensed hyperspectral data.[13,14]

In this work, we develop new GPU implementations of unconstrained, partially constrained and fully constrained abundance estimation algorithms in the hyperspectral unmixing chain. The proposed methodologies have been implemented using NVidia's compute device unified architecture (`CUDA`), and tested on the NVidia Tesla C1060 architecture, achieving very significant speedups when compared to optimized implementations of the same codes in a standard CPU.

2. PROBLEM FORMULATION

Let us denote a remotely sensed hyperspectral scene with n bands by $\mathbf{I}$, in which the pixel at the discrete spatial coordinates (i,j) of the scene is represented by a vector $\mathbf{X}(i,j) = [x_1(i,j), x_2(i,j), \cdots, x_n(i,j)] \in \Re^n$, where $\Re$ denotes the set of real numbers in which the pixel's spectral response $x_k(i,j)$ at sensor channels $k = 1, \ldots, n$ is included. Under the linear mixture model assumption, each pixel vector in the original scene can be modeled using the following expression:

$$\mathbf{X}(i,j) = \sum_{z=1}^{p} \Phi_z(i,j) \cdot \mathbf{E}_z + \mathbf{n}(i,j), \tag{1}$$

where $\mathbf{E}_z$ denotes the spectral response of endmember z, $\Phi_z(i,j)$ is a scalar value designating the fractional abundance of the endmember z at the pixel $\mathbf{X}(i,j)$, p is the total number of endmembers, and $\mathbf{n}(i,j)$ is a noise vector. Two physical constrains are generally imposed into the model described in Eq. (1), these are the abundance non-negativity constraint (ANC), i.e., $\Phi_z(i,j) \geq 0$, and the abundance sum-to-one constraint (ASC),

i.e., $\sum_{z=1}^{p} \Phi_z(i,j) = 1$.[8] Once a set of endmembers $\mathbf{E} = \{\mathbf{E}_z\}_{z=1}^{p}$ have been extracted by a certain algorithm,[9] their correspondent abundance fractions $\Phi(i,j) = \{\Phi_z(i,j)\}_{z=1}^{p}$ in a specific pixel vector $\mathbf{X}(i,j)$ of the image $\mathbf{I}$ can be estimated (in least squares sense) by the following unconstrained expression:[10]

$$\hat{\Phi}_{\mathrm{LSU}}(i,j) = (\mathbf{E}^T\mathbf{E})^{-1}\mathbf{E}^T\mathbf{X}(i,j). \tag{2}$$

However, it should be noted that the fractional abundance estimations obtained by means of Eq. (2) do not satisfy the ASC and ANC constraints. Imposing the ASC constraint results in the following optimization problem:

$$\min_{\Phi(i,j)\in\Delta} \left\{ (\mathbf{X}(i,j) - \Phi(i,j)\cdot\mathbf{E})^T (\mathbf{X}(i,j) - \Phi(i,j)\cdot\mathbf{E}) \right\}, \quad \text{subject to: } \Delta = \left\{ \Phi(i,j) \left| \sum_{z=1}^{p} \Phi_z(i,j) = 1 \right. \right\}. \tag{3}$$

Similarly, imposing the ANC constraint results in the following optimization problem:

$$\min_{\Phi(i,j)\in\Delta} \left\{ (\mathbf{X}(i,j) - \Phi(i,j)\cdot\mathbf{E})^T (\mathbf{X}(i,j) - \Phi(i,j)\cdot\mathbf{E}) \right\}, \quad \text{subject to: } \Delta = \{\Phi(i,j) | \Phi_z(i,j) \geq 0 \text{ for all } 1 \leq z \leq p\}. \tag{4}$$

As indicated in previous work,[8,15] a non-negative constrained least squares (NCLS) algorithm can be used to obtain a solution to the ANC-constrained problem described in Eq. (4) in iterative fashion.[16] In order to take care of the ASC constraint, a new endmember signature matrix, denoted by $\mathbf{E}'$, and a modified version of the pixel vector $\mathbf{X}(i,j)$, denoted by $\mathbf{X}'(i,j)$, are introduced as follows:

$$\mathbf{E}' = \begin{bmatrix} \delta\mathbf{M} \\ \mathbf{1}^T \end{bmatrix}, \Phi'(i,j) = \begin{bmatrix} \delta\Phi(i,j) \\ 1 \end{bmatrix}, \tag{5}$$

where $\mathbf{1} = (\underbrace{1, 1, \cdots, 1}_{p})^T$ and δ controls the impact of the ASC constraint. Using the two expressions in (5), a fully constrained estimate can be directly obtained from the NCLS algorithm by replacing $\mathbf{E}$ and $\Phi(i,j)$ used in the NCLS algorithm with $\mathbf{E}'$ and $\Phi'(i,j)$. This fully constrained (i.e. ASC-constrained and ANC-constrained) linear spectral unmixing estimate is referred to by the acronym FCLSU.[8]

3. GPU IMPLEMENTATIONS

This section describes our GPU implementations of abundance estimation algorithms. GPUs can be abstracted in terms of a *stream* model, under which all data sets are represented as streams (i.e., ordered data sets). Algorithms are constructed by chaining so-called *kernels*, which operate on entire streams, taking one or more streams as inputs and producing one or more streams as outputs.[13] Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied. First, we present a GPU implementation of the unconstrained unmixing method, called GPU-LSU. Then, we describe our parallel version of the NCLS algorithm, called GPU-NCLS. Finally, we describe the parallel version of FCLSU which is referred to as GPU-FCLSU. In all cases, code examples illustrating the most relevant kernels constructed in the GPU implementations are given to better understand our designs.

```
__global__ void Unmixing(float *d_image_vector, float *d_image_unmixed,
float *d_compute_matrix, int num_lines, int num_samples, int num_bands,
int num_endmembers)
{
// d_image_vector is the structure that stores the hyperspectral image in the device
// d_image_unmixed is the unmixed image
// idx is the identification number of the thread

      __shared__ float compute_matrix[num_bands];
      int idx =  blockDim.x * blockIdx.x+threadIdx.x;
      float processed_pixel[num_bands];

      for (int t = 0; t < num_bands; t++) {
      processed_pixel[t] = d_image_vector[idx+(num_lines*num_samples*t)];
      }

      for (int iter = 0; iter < num_endmembers; iter++) {
            if (threadIdx.x == 0){
                  for (int i = 0; i < num_bands; i++) {
                  compute_matrix[i] = d_compute_matrix[iter*num_bands+i];
                  }
            }
            syncthreads();
            for (int k = 0; k < num_bands; k++) {
                  d_image_unmixed[idx+(num_lines*num_samples*iter)] +=
                  compute_matrix[k]*processed_pixel[k];
            }
      }
}
```

Figure 4. `CUDA` kernel `Unmixing` that computes unconstrained abundances in each pixel of the hyperspectral image.

3.1 GPU-LSU

Our GPU version of the unconstrained abundance estimation algorithm assumes that a set of endmembers is available and produces a set of endmember abundance maps as follows. The first step in the GPU-LSU is to calculate a so-called compute matrix $\left(\mathbf{E}^T\mathbf{E}\right)^{-1}\mathbf{E}^T$, where $\mathbf{E} = \{\mathbf{E}_z\}_{z=1}^{p}$ is formed by the p endmembers previously extracted. This compute matrix will be multiplied by all pixel vectors $\mathbf{X}(i,j)$ in the original image. In our implementation, the compute matrix is calculated in the CPU mainly due to two reasons: 1) its computation is relatively fast, and 2) once calculated, the compute matrix remains the same throughout the whole execution of the code. The compute matrix is now multiplied by each pixel $\mathbf{X}(i,j)$ in the hyperspectral image, thus obtaining a set of abundance vectors $\Phi(i,j)$, each containing the fractional abundances of the p endmembers in each pixel. This is accomplished in the GPU by means of the `Unmixing` kernel illustrated in Fig. 4, in which the outcome of the unmixing process (i.e. the p endmember abundance maps) are stored in `d_image_unmixed`.

3.2 GPU-NCLS and GPU-FCLSU

Our GPU version of the ANC-constrained abundance estimation algorithm (called GPU-NCLS) is approached from the perspective of a distance minimization problem where the distance between the measured pixel or spectra and the estimate is the smallest. For this purpose, we resort to the image space reconstruction algorithm (ISRA),[15] an iterative algorithm and an example of an algorithm satisfying only the ANC constraint. In other words, this algorithm guarantees convergence in a finite number of iterations and positive values in the abundance estimation results for any input set of endmembers. For illustrative purposes, Fig. 5 shows the ISRA kernel, which iteratively calculates the fractional abundance of a set of pre-calculated endmembers $\mathbf{E}_z$ in a pixel $\mathbf{X}(i,j)$ independently of the other pixels and, hence, can be efficiently executed in parallel. Finally, Fig. 6 shows the kernel used to implement the GPU-FCLSU algorithm. This kernel simply scales the abundances provided by GPU-NCLS to provide fully constrained estimates.

```
__global__ void ISRA(float *d_image_vector, float *d_image_unmixed, float
*d_endmembers, float *d_endmembersT, int num_lines, int num_samples, int
num_bands, int N_END, int MAX_ITER)
{
// d_image_vector is the structure that stores the hyperspectral image in the device
// d_image_unmixing is the structure that stores the abundance estimation maps
// idx is the identification number of the thread

      int idx = blockDim.x * blockIdx.x + threadIdx.x;
      float s_end[NUM_BANDS]; float s_endt[NUM_END];
      float l_pixel[NUM_BANDS]; float l_abu[NUM_END];
      float numerator; float denominator; float dot = 0.0;

      //For all pixels
      for(int t=0; t<num_bands; t++){
            l_pixel[t]=d_image_vector[idx+(num_lines*num_samples*t)];
      }

      //Calculate abundances using ISRA for one pixel
      for (int it=0; it<=MAX_ITER; it++){
            numerator=0; denominator=0;

            //For all endmembers
            for(int e=0; e<N_END; e++){

                  //Read an endmember and store it in shared memory
                  if(threadIdx.x==0){
                        for(int i=0; i<num_bands; i++){
                        s_end[i]=d_endmembers[e*num_bands+i];}
                  }
                  syncthreads();

                  //For all bands
                  for(int k=0; k<num_bands; k++){
                        numerator=numerator + s_end[k]*l_pixel[k];

                        if(threadIdx.x==0){
                              for(int i=0; i<num_bands; i++){
                                    s_endt[i]=d_endmembersT[k*N_END+i];
                              }
                        }
                        syncthreads();

                        //Calculate dot product
                        for(int s=0; s<N_END; s++){
                              dot += s_endt[s]*l_abu[s];
                        }
                        denominator += dot*s_end[k];
                        dot=0;}

                  //Calculate a new abundance
                  l_abu[e] *= (numerator/denominator);
                  numerator=0; denominator=0;
            }
            //Store abundance in global memory
            for(int k=0; k<N_END; k++){
            d_image_unmixed[pixel+(num_lines*num_samples*k)]=l_abu[k];
      }
}
```

Figure 5. CUDA kernel ISRA that computes endmember abundances in each pixel of the hyperspectral image imposing the abundance non-negativity (ANC) constraint.

4. EXPERIMENTAL RESULTS

4.1 Hyperspectral image data

The image scene used for experiments in this work was collected by the AVIRIS instrument, which was flown by NASA's Jet Propulsion Laboratory over the World Trade Center area in New York City on September 16, 2001, just five days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex. The full data set selected for experiments consists of 614×512 pixels, 224 spectral bands and

```
__global__ void FCLSU(float *d_image_unmixed, int num_lines,
int num_samples, int N_END)
{
      float s_pixels[Tamanio_Vector];
      float sum;
      float pixel[NUM_END];

      int idx =  blockDim.x * blockIdx.x+threadIdx.x;

      for(int i=0; i<NUM_END; i++){
            pixel[i]=d_imagen_unmixing[idx+(num_lines*num_samples*i)];
      }

      sum=0;
      for (int j=0;j<NUM_END; j++){
            sum=sum+pixel[j];
      }

      for(int k=0; k<NUM_END; k++){
            pixel[k]=pixel[k]/sum;
            d_image_unmixed[idx+(num_lines*num_samples*k)]=pixel[k];
      }
}
```

Figure 6. `CUDA` kernel `FCLSU` that computes endmember abundances in each pixel of the hyperspectral image imposing the abundance non-negativity (ANC) and abundance sum-to-one (ASC) constraints.

a total size of (approximately) 140 MB. The spatial resolution is 1.7 meters per pixel. The leftmost part of Figure 7 shows a false color composite of the data set selected for experiments using the 1682, 1107 and 655 nm channels, displayed as red, green and blue, respectively. Vegetated areas appear green in the leftmost part of Figure 7, while burned areas appear dark gray. Smoke coming from the WTC area (in the red rectangle) and going down to south Manhattan appears bright blue due to high spectral reflectance in the 655 nm channel. Extensive reference information, collected by U.S. Geological Survey (USGS), is available for the WTC scene*. For instance, a USGS thermal map is available for this scene†, showing the target locations of the thermal hot spots at the WTC area which are displayed as bright red, orange and yellow spots at the rightmost part of Figure 7. The map is centered at the region where the towers collapsed.

4.2 GPU architecture

The proposed GPU implementations have 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‡. 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.

4.3 Discussion of results

Before describing our experimental results, it is important to emphasize that our GPU implementations provide exactly the same results as the serial versions of the same algorithms, 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. In our experiments, the number of endmembers to be detected has always been fixed to $p = 30$ after calculating the virtual dimensionality (VD) of the hyperspectral data,[17] where the endmember signatures were extracted directly from the image scene (assumed to comprise sufficient spatial resolution to allow the presence of relatively pure pixels) using the orthogonal subspace projection (OSP) technique.[18]

*http://speclab.cr.usgs.gov/wtc

†http://pubs.usgs.gov/of/2001/ofr-01-0429/hotspot.key.tgif.gif

‡http://www.nvidia.com/object/product_tesla_c1060_us.html

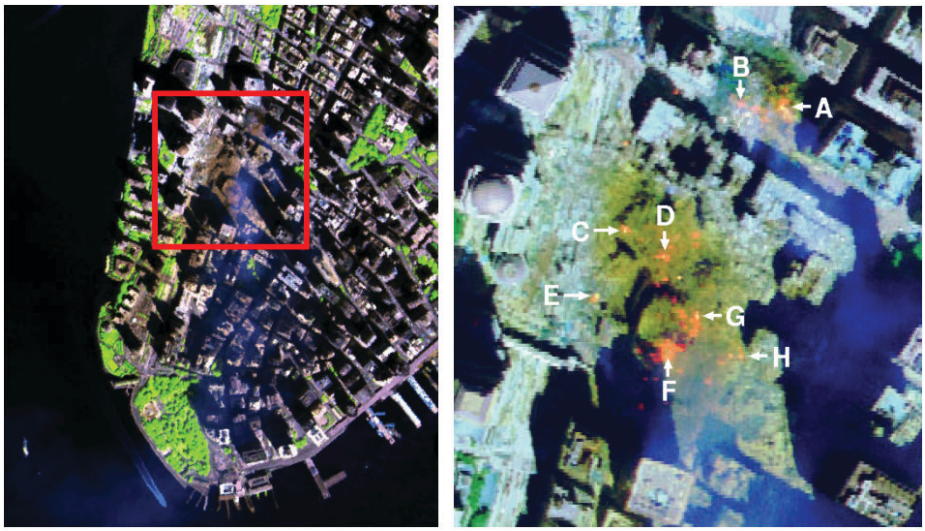


Figure 7. False color composition of an AVIRIS hyperspectral image collected by NASA's Jet Propulsion Laboratory over lower Manhattan on Sept. 16, 2001 (left). Location of thermal hot spots in the fires observed in World Trade Center area (right).

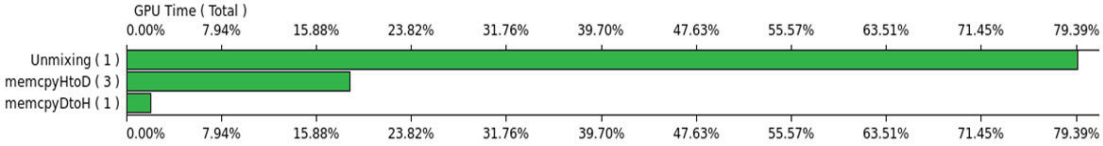


Figure 8. Summary plot describing the percentage of the total GPU time consumed by the different kernels used in the implementation of the GPU-LSU in the NVidia Tesla C1060 GPU.

First, we analyze the parallel performance of GPU-LSU in the NVidia Tesla C1060 GPU. Due to the fact that the hyperspectral image consists of 614×512 pixels, the configuration of the `Unmixing` kernel (which receives the endmembers calculated by GPU-OSP as input) was based on 1228 blocks and 256 threads per block. As a result, in total there are as many threads as pixels in the image. The idea is each thread computes only one multiplication with the compute matrix to calculate the endmember abundances in the pixel. Before executing the `Unmixing` kernel, we need to load in the GPU global memory the hyperspectral image and the compute matrix. For each pixel, we perform the multiplication with the compute matrix by moving such matrix (line by line) to the shared memory in the GPU and performing the needed operations. Each thread only needs 7 registers and each line of the compute matrix has 940 bytes; this amount of resources allow us to achieve full occupancy of the multiprocessors using 4 blocks per multiprocessor (1024 threads for our configuration). With this scheme, we obtain a significant speedup of 81.5x after comparing the GPU time with the CPU time (33.28 seconds for the unmixing in the CPU Intel core i7 versus only 0.48 seconds in the GPU). For illustrative purposes, Fig. 8 shows the percentage of the total execution time employed by each of the `CUDA` kernels obtained after profiling the GPU-LSU implementation. The first bar represents the percentage of time employed by the `Unmixing` kernel, while the second and third bars respectively denote the percentage of time for data movements from host (CPU) to device (GPU) and from device to host. The reason why there are three data movements from host to device is because not only the hyperspectral image but also the compute matrix and the structure where the abundances are stored need to be forwarded to the GPU, which returns the structure with the estimated abundances after the calculation is completed. As shown by Fig. 8, the `Unmixing` kernel consumes about 80% of the total GPU time, while the time for data movements is not significant when compared to the time invested in performing the spectral unmixing operation.

On the other hand, Fig. 9 shows the percentage of the total execution time employed by each of the `CUDA` kernels obtained after profiling the GPU-FCLSU implementation (it should be noted that the GPU-NCLS

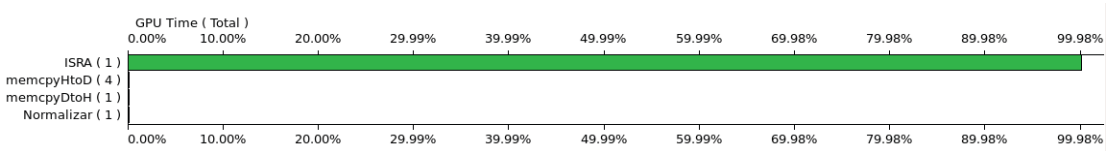


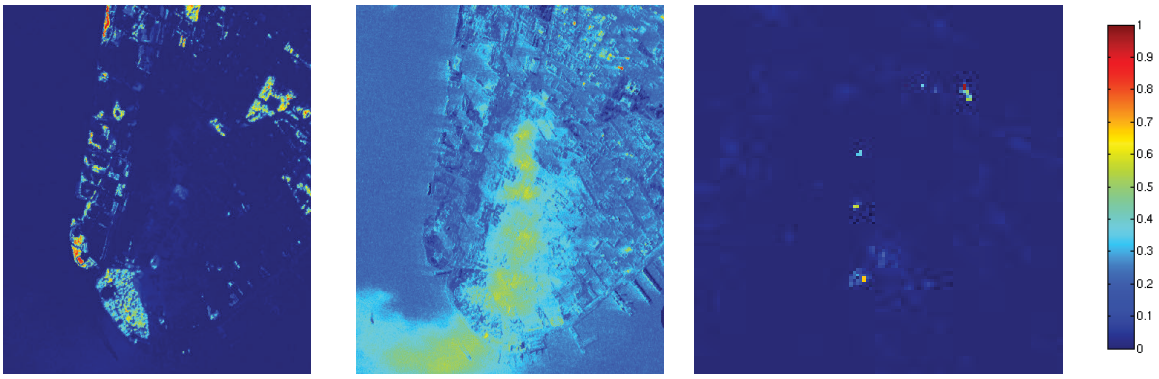
Figure 9. Summary plot describing the percentage of the total GPU time consumed by the different kernels used in the implementation of the GPU-FCLSU in the NVidia Tesla C1060 GPU.

implementation is a subset of the GPU-FCLSU one, with the normalization The first bar represents the percentage of time employed by the `ISRA` kernel, while the second and third bars respectively denote the percentage of time for data movements from host (CPU) to device (GPU) and from device to host. The reason why there are four data movements from host to device is because not only the hyperspectral image but also the endmember matrix, the transposed endmember matrix, and the structure where the abundances are stored need to be forwarded to the GPU, which returns the structure with the estimated abundances after the calculation is completed. The fourth bar in Fig. 8 represents the percentage of time employed by the `FCLSU` kernel, which is very small compared with the `ISRA` kernel that dominates the computations. As shown by Fig. 8, the `ISRA` kernel consumes more than 99% of the total GPU time indicating that the operation is highly parallelizable in the GPU. This is also indicated by the speedup values obtained: 26.4x after comparing the GPU time with the CPU time for the GPU-NCLS (3351.03 seconds for the unmixing in the CPU versus 126.65 seconds in the GPU) and also 26.4x speedup after comparing the GPU time with the CPU time for the GPU-FCLSU (3353.70 seconds for the unmixing in the CPU versus 126.69 seconds in the GPU). Here, it is worth noting that the GPU-NCLS and GPU-FCLSU versions are much more computationally expensive, but the serial versions are also slower than the one for the unconstrained unmixing method.

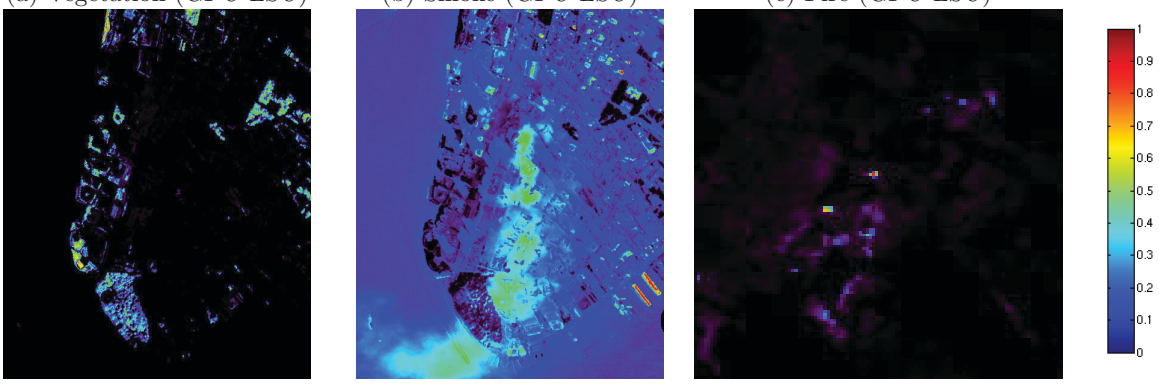
Finally, Fig. 10 shows some of the abundance maps for the most relevant endmembers extracted from the AVIRIS hyperspectral scene after applying the proposed GPU implementations of spectral unmixing algorithms (the fire abundance maps have been rescaled to match the dimensions of the subscene displayed in the rightmost part of Fig. 7, while the vegetation and smoke maps correspond to the scene displayed in the leftmost part of Fig. 7). We recall that endmember abundance maps are the outcome of the unmixing process, where each map reflects the sub-pixel composition of a certain endmember in each pixel of the scene. Specifically, the maps displayed in Fig. 10 correspond to vegetation, smoke and fire. As shown by Fig. 10 there are some scaling differences between the maps estimated by different methods, although we anticipate that the best compromise results are provided by GPU-NCLS (vegetation and fire maps with a clear separation between no abundance at all (0.0 estimated value) and high concentration of the estimated endmember) and GPU-FCLSU (good compromise result for the vegetation endmember). In turn, the GPU-LSU provides maps with fewer 0.0 estimated abundance values. Although these evaluation results are analyzed from a qualitative perspective, further quantitative assessment with simulated hyperspectral scenes are highly desirable in order to fully substantiate these remarks.

5. CONCLUSIONS AND FUTURE RESEARCH LINES

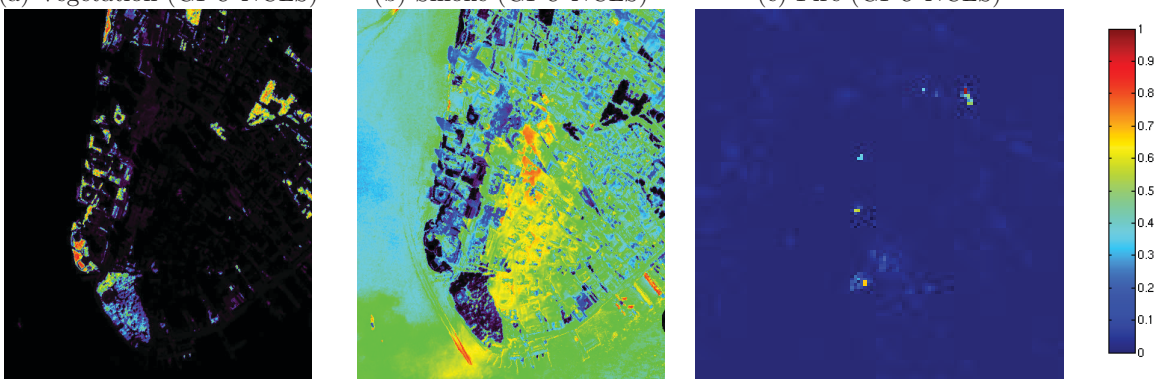
In this paper, we develop new parallel implementations of unconstrained, partially constrained and fully constrained linear spectral unmixing algorithms. The results obtained by our GPU implementations indicate that remotely sensed hyperspectral imaging can greatly benefit from the development of efficient implementations of unmixing 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 the context of a real application with important time constraints (unmixing of hyperspectral data sets collected over the World Trade Center in New York City after the terrorist attacks of September 11th, 2001). In this case, significant speedups can be obtained for the different spectral unmixing implementations tested 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). In the future, we will evaluate our implementations using additional hyperspectral scenes and high performance computing architectures in order to fully extrapolate our promising results to additional case studies and analysis scenarios.



(a) Vegetation (GPU-LSU) (b) Smoke (GPU-LSU) (c) Fire (GPU-LSU)



(a) Vegetation (GPU-NCLS) (b) Smoke (GPU-NCLS) (c) Fire (GPU-NCLS)



(a) Vegetation (GPU-FCLSU) (b) Smoke (GPU-FCLSU) (c) Fire (GPU-FCLSU)

Figure 10. Abundance maps extracted by different GPU implementations for the vegetation (a), smoke (b) and fire (c).

6. ACKNOWLEDGEMENT

This work has been supported by the European Community's Marie Curie Research Training Networks Programme under reference MRTN-CT-2006-035927 (HYPER-I-NET). Funding from the Spanish Ministry of Science and Innovation (HYPERCOMP/EODIX project, reference AYA2008-05965-C04-02) is gratefully acknowledged.

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. M. E. Schaepman, S. L. Ustin, A. Plaza, T. H. Painter, J. Verrelst, and S. Liang, "Earth system science related imaging spectroscopy – an assessment," *Remote Sensing of Environment* **113**, pp. 123–137, 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* **91**, pp. 8098–8112, 1986.
5. N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Processing Magazine* **19**(1), pp. 44–57, 2002.
6. 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* **65**(3), pp. 227–248, 1998.
7. J. E. Ball, L. M. Bruce, and N. Younan, "Hyperspectral pixel unmixing via spectral band selection and dc-insensitive singular value decomposition," *IEEE Geoscience and Remote Sensing Letters* **4**(3), pp. 382–386, 2007.
8. D. Heinz and C.-I. Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing* **39**, pp. 529–545, 2001.
9. 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* **42**(3), pp. 650–663, 2004.
10. C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*, Kluwer Academic/Plenum Publishers: New York, 2003.
11. K. J. Guilfoyle, M. L. Althouse, and C.-I. Chang, "A quantitative and comparative analysis of linear and nonlinear spectral mixture models using radial basis function neural networks," *IEEE Trans. Geosci. Remote Sens.* **39**, pp. 2314–2318, 2001.
12. J. Plaza, A. Plaza, R. Perez, and P. Martinez, "On the use of small training sets for neural network-based characterization of mixed pixels in remotely sensed hyperspectral images," *Pattern Recognition* **42**, pp. 3032–3045, 2009.
13. J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geoscience and Remote Sensing Letters* **43**, pp. 441–445, 2007.
14. 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.
15. A. R. D. Pierro, "On the relation between ISRA and the EM algorithm for positron emission tomography," *IEEE Transactions on Medical Imaging* **12**, pp. 328–333, 1993.
16. C.-I. Chang and D. Heinz, "Constrained subpixel target detection for remotely sensed imagery," *IEEE Transactions on Geoscience and Remote Sensing* **38**, pp. 1144–1159, 2000.
17. C.-I. Chang and Q. Du, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing* **42**(3), pp. 608–619, 2004.
18. J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection," *IEEE Transactions on Geoscience and Remote Sensing* **32**(4), pp. 779–785.