GPU Implementation of Target and Anomaly Detection Algorithms for Remotely Sensed Hyperspectral Image Analysis

Abel Paz[a] and Antonio Plaza[a]

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

ABSTRACT

Automatic target and anomaly detection are considered very important tasks for hyperspectral data exploitation. These techniques are now routinely applied in many application domains, including defence and intelligence, public safety, precision agriculture, geology, or forestry. Many of these applications require timely responses for swift decisions which depend upon high computing performance of algorithm analysis. However, with the recent explosion in the amount and dimensionality of hyperspectral imagery, this problem calls for the incorporation of parallel computing techniques. In the past, clusters of computers have offered an attractive solution for fast anomaly and target detection in hyperspectral data sets already transmitted to Earth. However, these systems are expensive and difficult to adapt to on-board data processing scenarios, in which low-weight and low-power integrated components are essential to reduce mission payload and obtain analysis results in (near) real-time, i.e., at the same time as the data is collected by the sensor. An exciting new development in the field of commodity computing is the emergence of commodity graphics processing units (GPUs), which can now bridge the gap towards on-board processing of remotely sensed hyperspectral data. In this paper, we describe several new GPU-based implementations of target and anomaly detection algorithms for hyperspectral data exploitation. The parallel algorithms are implemented on latest-generation Tesla C1060 GPU architectures, and quantitatively evaluated using hyperspectral data collected by NASA's AVIRIS system 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.

Keywords: Hyperspectral imaging, automatic target and anomaly detection, parallel computing, graphics processing units (GPUs).

1. INTRODUCTION

Hyperspectral imaging[1] is concerned with the measurement, analysis, and interpretation of spectra acquired from a given scene (or specific object) at a short, medium or long distance by an airborne or satellite sensor. Hyperspectral imaging instruments such as the NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS)[2] are now 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.[1] The resulting "image cube" (see Fig. 1) is a stack of images in which each pixel (vector) has an associated spectral signature or *fingerprint* that uniquely characterizes the underlying objects. The resulting data volume typically comprises several GBs per flight.

Automatic target and anomaly detection are very important tasks for hyperspectral data exploitation in many different domains and, specifically, in defense and intelligence applications. During the last few years, several algorithms have been developed for the aforementioned purposes, including the automatic target detection and classification (ATDCA) algorithm[3] or the well-known RX algorithm developed by Reed and Xiaoli for anomaly detection.[4] The ATDCA algorithm finds a set of spectrally distinct target pixels vectors using the

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, 78100R
© 2010 SPIE · CCC code: 0277-786X/10/$18 · doi: 10.1117/12.860213

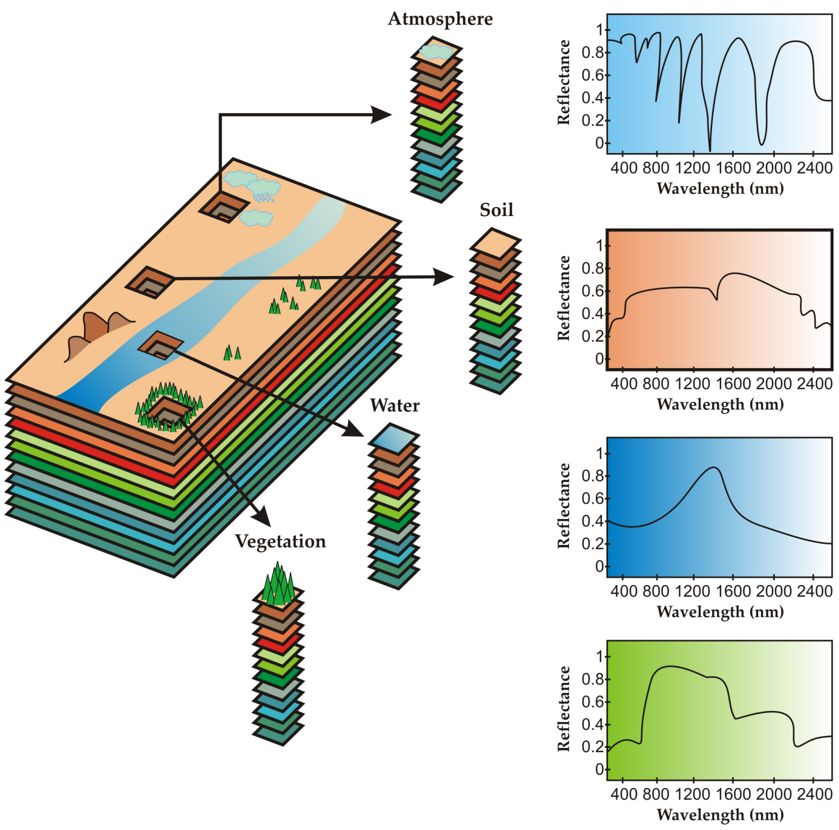


Figure 1. Concept of hyperspectral imaging.

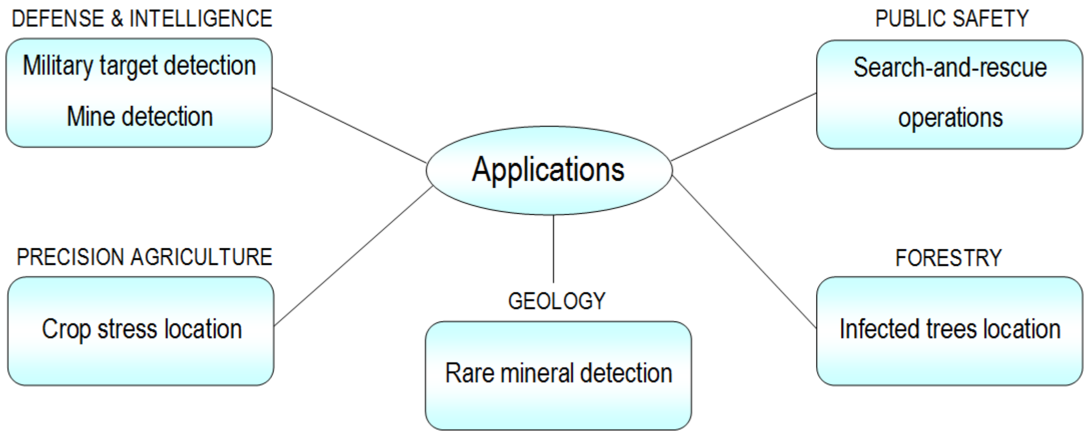


Figure 2. Applications of target and anomaly detection.

concept of orthogonal subspace projection (OSP)[5] in the spectral domain. The RX algorithm is based on the application of a so-called RXD filter, given by the well-known Mahalanobis distance. Many other target/anomaly detection algorithms have also been proposed in the recent literature, using different concepts such as background modeling.[6]

Depending on the complexity and dimensionality of the input scene, the aforementioned algorithms may be computationally very expensive, a fact that limits the possibility of utilizing those algorithms in time-critical applications. In turn, the wealth of spectral information available in hyperspectral imaging data opens ground-breaking perspectives in many applications, including target detection for military and defense/security deployment. Only a few parallel implementations of automatic target and anomaly detection algorithms for hyperspectral data exist in the open literature.[7] However, with the recent explosion in the amount and dimensionality of hyperspectral imagery, parallel processing is expected to become a requirement in most remote sensing missions, including those related with the detection of anomalous and/or concealed targets.

An exciting new development in the field of commodity computing is the emergence of commodity graphic processing units (GPUs), which can now bridge the gap towards on-board processing of remotely sensed hyperspectral data. The speed of graphics hardware doubles approximately every six months, which is much faster than the improving rate of the CPUs (even those made up by multiple cores) which are interconnected in a cluster. Currently, state-of-the-art GPUs deliver peak performances more than one order of magnitude over high-end micro-processors.[8] The ever-growing computational requirements introduced by hyperspectral imaging applications can fully benefit from this type of specialized hardware and take advantage of the compact size and relatively low cost of these units, which make them appealing for onboard data processing at lower costs than those introduced by other hardware devices.[9]

In this paper, we develop and compare several new computationally efficient parallel versions (for GPUs) of two highly representative algorithms for target (ATDCA) and anomaly detection (RX) in hyperspectral scenes. The parallel versions are quantitatively and comparatively analyzed (in terms of target detection accuracy and parallel performance) in the framework of a real defense and security application, focused on identifying thermal hot spots (which can be seen as targets and/or anomalies) 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. The parallel performance of the proposed implementations is assessed using an NVidia$^{\text{TM}}$ GeForce 9800 GX2 GPU.

2. METHODS

In this section we briefly describe the target detection algorithms that will be efficiently implemented in parallel (using different high performance computing architectures) in this work. These algorithms are the ATDCA for automatic target and classification, and the RX for anomaly detection. In the former case, several distance measures are described for implementation of the algorithm.

2.1 ATDCA algorithm

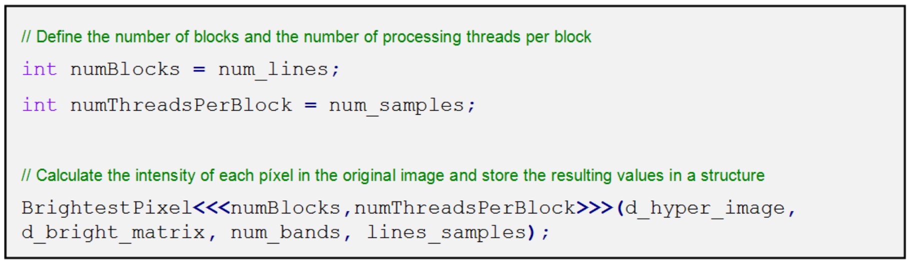
The ATDCA algorithm[3] was developed to find potential target pixels that can be used to generate a signature matrix used in an orthogonal subspace projection (OSP) approach.[5] Let $\mathbf{x}_0$ be an initial target signature (i.e., the pixel vector with maximum length). The ATDCA begins by an orthogonal subspace projector specified by the following expression:

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

which is applied to all image pixels, with $\mathbf{U} = [\mathbf{x}_0]$. It then finds a target signature, denoted by $\mathbf{x}_1$, with the maximum projection in $< \mathbf{x}_0 >^{\perp}$, which is the orthogonal complement space linearly spanned by $\mathbf{x}_0$. A second target signature $\mathbf{x}_2$ can then be found by applying another orthogonal subspace projector $P_{\mathbf{U}}^{\perp}$ with $\mathbf{U} = [\mathbf{x}_0, \mathbf{x}_1]$ to the original image, where the target signature that has the maximum orthogonal projection in $< \mathbf{x}_0, \mathbf{x}_1 >^{\perp}$ is selected as $\mathbf{x}_2$. The above procedure is repeated until a set of target pixels $\{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_t\}$ is extracted, where t is an input parameter to the algorithm.

In addition to the standard OSP approach, we have explored other alternatives in the implementation of ATDCA, given by replacing the $P_{\mathbf{U}}^{\perp}$ operator used in the OSP implementation by one of the distance measures described below:[10, 11]

- The 1-Norm between two pixel vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, defined by $\|\mathbf{x}_i - \mathbf{x}_j\|$.
- The 2-Norm between two pixel vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, defined by $\|\mathbf{x}_i - \mathbf{x}_j\|_2$.
- The Infinity-Norm between two pixel vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, defined by $\|\mathbf{x}_i - \mathbf{x}_j\|_\infty$.



```
// Define the number of blocks and the number of processing threads per block
int numBlocks = num_lines;
int numThreadsPerBlock = num_samples;

// Calculate the intensity of each pixel in the original image and store the resulting values in a structure
BrightestPixel<<<numBlocks,numThreadsPerBlock>>>(d_hyper_image,
d_bright_matrix, num_bands, lines_samples);
```

Figure 3. Portion of code which calls the `CUDA` kernel `BrightestPixel` that computes (in parallel) the brightest pixel in the scene in the G-ATDCA implementation.

- The spectral angle distance (SAD) between two pixel vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, defined by the following expression:[1] $\mathrm{SAD}(\mathbf{x}_i, \mathbf{x}_j) = \cos^{-1}\left(\frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \cdot \|\mathbf{x}_j\|_2}\right)$. As opposed to the previous metric, SAD is invariant in the presence of illumination interferers, which can provide advantages in terms of target and anomaly detection in complex backgrounds.
- The spectral information divergence (SID) between two pixel vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, defined by the following expression:[1] $\mathrm{SID}(\mathbf{x}_i, \mathbf{x}_j) = \mathrm{D}\left(\mathbf{x}_i \| \mathbf{x}_j\right) + \mathrm{D}\left(\mathbf{x}_j \| \mathbf{x}_i\right)$, where $\mathrm{D}\left(\mathbf{x}_i \| \mathbf{x}_j\right) = \sum_{k=1}^{n} p_k \cdot \log(p_k/q_k)$. Here, we define $p_k = x_i^{(k)} / \sum_{k=1}^{n} x_i^{(k)}$ and $q_k = x_j^{(k)} / \sum_{k=1}^{n} x_j^{(k)}$.

2.2 RX algorithm

The RX algorithm has been widely used in signal and image processing.[4] The filter implemented by this algorithm is referred to as RX filter (RXF) and defined by the following expression:

$$\delta^{\mathrm{RXF}}(\mathbf{x}) = (\mathbf{x} - \mu)^T \mathbf{K}^{-1} (\mathbf{x} - \mu), \tag{2}$$

where $\mathbf{x} = \left[x^{(0)}, x^{(1)}, \cdots, x^{(n)}\right]$ is a sample, n-dimensional hyperspectral pixel (vector), μ is the sample mean and $\mathbf{K}$ is the sample data covariance matrix. As we can see, the form of δ^{RXF} is actually the well-known Mahalanobis distance.[12] It is important to note that the images generated by the RX algorithm are generally gray scale images. In this case, the anomalies can be categorized in terms of the value returned by RXF, so that the pixel with higher value of $\delta^{\mathrm{RXF}}(\mathbf{x})$ can be considered the first anomaly, and so on.

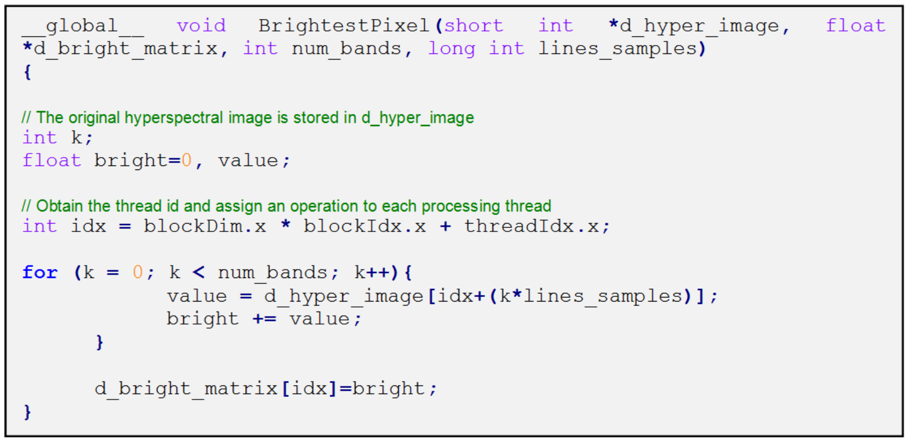
3. PARALLEL IMPLEMENTATIONS FOR GPUS

The first issue that needs to be addressed is how to map a hyperspectral image onto the memory of the GPU. Since the size of hyperspectral images usually exceeds the capacity of such memory, we split them into multiple spatial-domain partitions made up of entire pixel vectors, i.e., each spatial-domain partition incorporates all the spectral information on a localized spatial region and is composed of spatially adjacent pixel vectors. Each spatial-domain partition is further divided into 4-band tiles (called spatial-domain tiles), which are arranged in different areas of a 2-D texture.[8] Such partitioning allows us to map four consecutive spectral bands onto the RGBA color channels of a texture element. Once the procedure adopted for data partitioning has been described, we provide additional details about the GPU implementations of RX and ATDCA algorithms, referred to hereinafter as G-RX and G-ATDCA, respectively.

3.1 G-ATDCA

Our GPU version of the ATDCA algorithm for target detection is given by the following steps:

1. Once the hyperspectral image is mapped onto the GPU memory, a structure (grid) in which the number of blocks equals the number of lines in the hyperspectral image and the number of threads equals the number of samples is created, thus making sure that all pixels in the hyperspectral image are processed in parallel (if this is not possible due to limited memory resources in the GPU, `CUDA` automatically performs several iterations, each of which processes as many pixels as possible in parallel).



```
__global__  void  BrightestPixel(short  int  *d_hyper_image,  float
*d_bright_matrix, int num_bands, long int lines_samples)
{

// The original hyperspectral image is stored in d_hyper_image
int k;
float bright=0, value;

// Obtain the thread id and assign an operation to each processing thread
int idx = blockDim.x * blockIdx.x + threadIdx.x;

for (k = 0; k < num_bands; k++){
            value = d_hyper_image[idx+(k*lines_samples)];
            bright += value;
      }

      d_bright_matrix[idx]=bright;
}
```

Figure 4. `CUDA` kernel `BrightestPixel` that computes (in parallel) the brightest pixel in the scene in the G-ATDCA implementation.

2. Using the aforementioned structure, calculate the brightest pixel $\mathbf{x}_1$ in the original hyperspectral scene by means of a `CUDA` kernel which performs part of the calculations to compute $\mathbf{x}_1 = argmax\{\mathbf{x}^T \cdot \mathbf{x}\}$ after computing (in parallel) the dot product between each pixel vector $\mathbf{x}$ in the original hyperspectral image and its own transposed version $\mathbf{x}^T$. For illustrative purposes, Fig. 3 shows a portion of code which includes the definition of the number of blocks `numBlocks` and the number of processing threads per block `numThreadsPerBlock`, and then calls the `CUDA` kernel `BrightestPixel` that computes the value of $\mathbf{x}_1$. Here, `d_bright_matrix` is the structure that stores the output of the computation $\mathbf{x}^T \cdot \mathbf{x}$ for each pixel. Fig. 4 shows the code of the `CUDA` kernel `BrightestPixel`, in which each different thread computes a different value of $\mathbf{x}^T \cdot \mathbf{x}$ for a different pixel (each thread is given by an identification number `idx`, and there are as many concurrent threads as pixels in the original hyperspectral image). Once all the concurrent threads complete their calculations, the G-ATDCA implementation simply computes the value in `d_bright_matrix` with maximum associated value and obtains the pixel in that position, labeling the pixel as $\mathbf{x}_1$. Although this operation is inevitably sequential, it is performed in the GPU.

3. Once the brightest pixel in the original hyperspectral image has been identified as the first target $\mathbf{U} = \mathbf{x}_1$, the ATDCA algorithm is executed in the GPU by means of another kernel in which the number of blocks equals the number of lines in the hyperspectral image and the number of threads equals the number of samples is created, thus making sure that all pixels in the hyperspectral image are processed in parallel. The concurrent threads find (in parallel) the values obtained after applying the OSP-based projection operator $P_{\mathbf{U}}^{\perp} = \mathbf{I} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T$ to each pixel (using the structure `d_bright_matrix` to store the resulting projection values), and then the G-ATDCA algorithm finds a second target pixel from the values stored in `d_bright_matrix` as follows: $\mathbf{x}_2 = argmax\{(P_{\mathbf{U}}^{\perp}\mathbf{x})^T(P_{\mathbf{U}}^{\perp}\mathbf{x})\}$. The procedure is repeated until a set of t target pixels, $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t\}$, are extracted from the input data. Although in this description we have only referred to the OSP-based operation, the different metrics discussed in section 2.2 have been implemented by devising different kernels which can be replaced in our G-ATDCA implementation in plug and play fashion in order to modify the distance measure used by the algorithm to identify new targets along the process.

3.2 G-RX

Our GPU version of the RX algorithm for anomaly detection is given by the following steps:

1. Once the hyperspectral image is mapped onto the GPU memory, a structure (grid) containing n blocks of threads, each containing n processing threads, is defined using `CUDA`. As a result, a total of $n \times n$ processing threads are available.

```
// Initialization of matrix K
cudaMemset(d_K,0,size2InBytes);

// Calculation of RX filter
RXGPU<<< size, size >>>(d_hyper_image, d_K, lines_samples,
num_samples, num_lines, num_bands);

cudaThreadSynchronize ();
```

Figure 5. Portion of code which calls the `CUDA` kernel `RXGPU` designed to calculate the RX filter (in parallel) in the G-RX implementation.

```
// Calculation of final G-RX result
// numBlocks = num_lines;
// numThreadsPerBlock = num_samples;

RXResult <<< numBlocks, numThreadsPerBlock >>> (d_hyper_image,d_K,
d_result,lines_samples, num_samples, num_lines, num_bands);

cudaThreadSynchronize ();
```

Figure 6. Portion of code which calls the `CUDA` kernel `RXResult` designed to obtain a final set of targets (in parallel) in the G-RX implementation.

2. Using the aforementioned structure, calculate the sample spectral covariance matrix $\mathbf{K}$ in parallel by means of a `CUDA` kernel which performs the calculations needed to compute $\delta^{(\text{RXF})}(\mathbf{x}) = (\mathbf{x}-\mathbf{m})^T\mathbf{K}^{-1}(\mathbf{x}-\mathbf{m})$ for each pixel $\mathbf{x}$. For illustrative purposes, Fig. 5 shows a portion of code which includes the initialization of matrix $\mathbf{K}$ in the GPU memory using `cudaMemset`, a call to the `CUDA` kernel `RXGPU` designed to calculate $\delta^{(\text{RXF})}$, and finally a call to `cudaThreadSynchronize` to make sure that the initiated threads are synchronized. Here, `d_hyper_image` is the original hyperspectral image, `d_K` denotes the matrix $\mathbf{K}$, and `numlines`, `numsamples` and `numbands` respectively denote the number of lines, samples and bands of the original hyperspectral image. It should be noted that the `RXGPU` kernel implements the Gauss-Jordan elimination method for calculating $\mathbf{K}^{-1}$. blue We recall that the entire image data is allocated in the GPU memory, and therefore it is not necessary to partition the data. In fact, this is one of the main advantages of GPUs over clusters of computers (GPUs are shared memory architectures, while clusters are generally distributed memory architectures in which message passing is needed to distribute the workload among the workers). A particularity of the Gauss-Jordan elimination method is that it converts the source matrix into an identity matrix pivoting, where the pivot is the element in the diagonal of the matrix by which other elements are divided in an algorithm. The GPU naturally parallelizes the pivoting operation by applying the calculation at the same time to many rows and columns, and hence the inverse operation is calculated in parallel in the GPU.

3. Once the $\delta^{(\text{RXF})}$ has been computed (in parallel) for every pixel $\mathbf{x}$ in the original hyperspectral image, a final (also parallel) step selects the t pixel vectors with higher associated value of $\delta^{(\text{RXF})}$ (stored in `d_result`), and uses them to form a final set of targets $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t\}$. This is done using the portion of code illustrated in Fig. 6, which calls a `CUDA` kernel `RXResult` which implements this functionality. Here, the number of blocks `numBlocks` equals the number of lines in the hyperspectral image, while the number of threads `numThreadsPerBlock` equals the number of samples, thus making sure that all pixels in the hyperspectral image are processed in parallel (if this is not possible due to limited memory resources in the GPU, `CUDA` automatically performs several iterations, each of which processes as many pixels as possible in parallel).

4. EXPERIMENTAL RESULTS

This section is organized as follows. In subsection 4.1 we describe the AVIRIS hyperspectral data set used in our experiments. Subsection 4.2 describes the NVidia$^{\text{TM}}$ GeForce 8900 GX2 GPU. Subsection 4.3 discusses the target and anomaly detection accuracy of the parallel algorithms when analyzing the hyperspectral data set described

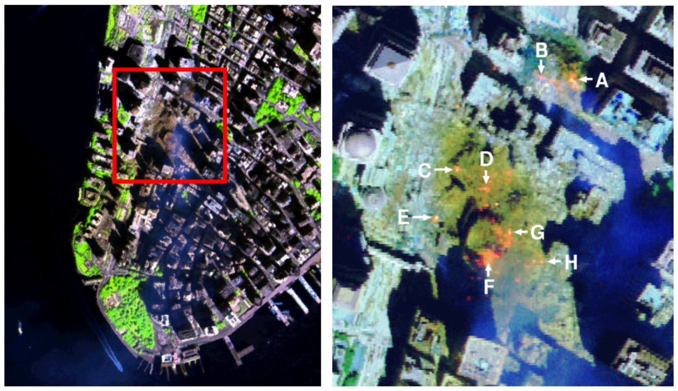


Figure 7. False color composition of an AVIRIS hyperspectral image collected by NASA's Jet Propulsion Laboratory over lower Manhattan on Sept. 16, 2001 (left). Location of thermal hot spots in the fires observed in World Trade Center area, available online: http://pubs.usgs.gov/of/2001/ofr-01-0429/hotspot.key.tgif.gif (right).

in subsection 4.1. Subsection 4.4 describes the parallel performance results obtained after implementing the G-ATDCA and G-RX algorithms on the GPU.

4.1 Data description

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 (WTC) 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 a total size of (approximately) 140 MB. The spatial resolution is 1.7 meters per pixel. The leftmost part of Fig. 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 Fig. 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[*]. In this work, we use a U.S. Geological Survey thermal map[†] which shows the target locations of the thermal hot spots at the WTC area, displayed as bright red, orange and yellow spots at the rightmost part of Fig. 7. The map is centered at the region where the towers collapsed, and the temperatures of the targets range from 700F to 1300F. Further information available from USGS about the targets (including location, estimated size, and temperature) is reported on Table 1. As shown by Table 1, all the targets are sub-pixel in size since the spatial resolution of a single pixel is 1.7 square meters. The thermal map displayed in the rightmost part of Fig. 7 will be used in this work as ground-truth to validate the target detection accuracy of the proposed parallel algorithms and their respective serial versions.

4.2 Parallel computing platforms

The NVidia$^{\mathrm{TM}}$ GeForce 9800 GX2 GPU contains two G92 graphics processors, each with 128 individual scalar processor (SP) cores and 512 MB of fast DDR3 memory[‡]. The SPs are clocked at 1.5 GHz, and each can perform a fused multiply-add every clock cycle, which gives the card a theoretical peak performance of 768 GFlop/s. The GPU is connected to a CPU Intel Q9450 with 4 cores, which uses a motherboard ASUS Striker II NSE (with NVidia$^{\mathrm{TM}}$ 790i chipset) and 4 GB of RAM memory at 1333 MHz. Hyperspectral data are moved to and from the host CPU memory by DMA transfers over a PCI Express bus.

[*]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_geforce_9800_gx2_us.html

Table 1. Properties of the thermal hot spots reported in the rightmost part of Fig. 7.

Hot spot	**Latitude (North)**	**Longitude (West)**	**Temperature (Kelvin)**	**Area (Square meters)**
'A'	$40^o42'47.18''$	$74^o00'41.43''$	1000	0.56
'B'	$40^o42'47.14''$	$74^o00'43.53''$	830	0.08
'C'	$40^o42'42.89''$	$74^o00'48.88''$	900	0.80
'D'	$40^o42'41.99''$	$74^o00'46.94''$	790	0.80
'E'	$40^o42'40.58''$	$74^o00'50.15''$	710	0.40
'F'	$40^o42'38.74''$	$74^o00'46.70''$	700	0.40
'G'	$40^o42'39.94''$	$74^o00'45.37''$	1020	0.04
'H'	$40^o42'38.60''$	$74^o00'43.51''$	820	0.08

Table 2. Spectral angle values (in degrees) between target pixels and known ground targets for G-ATDCA and G-RX.

Algorithm	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
G-ATDCA (OSP)	$9{,}17^o$	$13{,}75^o$	$0{,}00^o$	$0{,}00^o$	$20{,}05^o$	$28{,}07^o$	$21{,}20^o$	$21{,}77^o$
G-ATDCA (1-Norm)	$9{,}17^o$	$24{,}06^o$	$0{,}00^o$	$16{,}04^o$	$37{,}82^o$	$42{,}97^o$	$38{,}39^o$	$35{,}52^o$
G-ATDCA (2-Norm)	$9{,}17^o$	$24{,}06^o$	$0{,}00^o$	$16{,}04^o$	$37{,}82^o$	$42{,}97^o$	$38{,}39^o$	$25{,}78^o$
G-ATDCA (∞-Norm)	$8{,}59^o$	$22{,}35^o$	$0{,}00^o$	$13{,}75^o$	$27{,}50^o$	$30{,}94^o$	$21{,}20^o$	$26{,}36^o$
G-ATDCA (SAD)	$9{,}17^o$	$22{,}35^o$	$0{,}00^o$	$14{,}32^o$	$38{,}39^o$	$32{,}09^o$	$25{,}21^o$	$29{,}79^o$
G-ATDCA (SID)	$9{,}17^o$	$24{,}06^o$	$0{,}00^o$	$16{,}04^o$	$39{,}53^o$	$32{,}09^o$	$22{,}92^o$	$20{,}05^o$
G-RX	$0{,}00^o$	$12{,}03^o$	$0{,}00^o$	$0{,}00^o$	$18{,}91^o$	$28{,}07^o$	$33{,}80^o$	$40{,}68^o$

4.3 Analysis of target detection accuracy

It is first important to emphasize that our parallel versions of ATDCA and RX provide exactly the same results as the serial versions of the same algorithms, implemented using the Intel C/C++ compiler and optimized via compilation flags to exploit data locality and avoid redundant computations. At the same time, these results were also exactly the same as those achieved by the serial implementation and, hence, the only difference between the considered algorithms (serial and parallel) is the time they need to complete their calculations, which varies depending on the computer architecture in which they are run.

Table 2 shows the spectral angle distance (SAD) values (in degrees) between the most similar target pixels detected by G-RX and G-ATDCA (implemented using different distance metrics) and the pixel vectors at the known target positions, labeled from 'A' to 'H' in the rightmost part of Fig. 7. The lower the SAD score, the more similar the spectral signatures associated to the targets. In all cases, the number of target pixels to be detected was set to $t = 30$ after calculating the virtual dimensionality (VD) of the data.[13] As shown by Table 2, both the G-ATDCA and G-RX extracted targets were similar, spectrally, to the known ground-truth targets. The G-RX was able to perfectly detect (SAD of 0 degrees, represented in the table as 0^o) the targets labeled as 'A', 'C' and 'D' (all of them relatively large in size and with high temperature), while the G-ATDCA implemented using OSP was able to perfectly detect the targets labeled as 'C' and 'D'. Both the G-RX and G-ATDCA had more difficulties in detecting very small targets

4.4 Parallel performance

In this subsection we evaluate the parallel performance of both G-ATDCA and G-RX in the NVidia^{TM} GeForce 9800 GX2 GPU. Table 3 shows the execution times measured after processing the full hyperspectral scene (614×512 pixels and 224 spectral bands) on the CPU and on the GPU, along with the speedup achieved in each case. The C function `clock()` was used for timing the CPU implementation, and the `CUDA` timer was used for the GPU implementation. The time measurement was started right after the hyperspectral image file was read to the CPU memory and stopped right after the results of the target/anomaly detection algorithm were obtained and stored in the CPU memory.

From Table 3, it can be seen that the G-ATDCA implemented using the OSP distance scaled slightly worse than the other implementations. This suggests that the matrix inverse and transpose operations implemented

Table 3. Processing time (seconds) and speedups measured for the CPU and GPU implementations of several target and anomaly detection algorithms.

Algorithm	Processing time (CPU)	Processing time (GPU)	Speedup
G-ATDCA (OSP)	1263,21	370,96	3,40
G-ATDCA (1-Norm)	99,24	9,03	10,98
G-ATDCA (2-Norm)	83,99	9,41	9,28
G-ATDCA (∞-Norm)	109,28	9,05	12,07
G-ATDCA (SAD)	133,63	9,06	14,74
G-ATDCA (SID)	911,85	12,67	71,91
G-RX	1955,15	139,17	14,04

by the $P_{\mathbf{U}}^{\perp}$ orthogonal projection operator can still be optimized for efficient execution in the GPU. In this case, the speedup achieved by the GPU implementation over the optimized CPU implementation was only $3,4$. When the G-ATDCA was implemented using the 1-Norm, 2-Norm and ∞-Norm distances, the speedup increased to values around 10, for a total processing time below 10 seconds in the considered GPU. This can be defined as a significant accomplishment if we take in mind that just one GPU is used to parallelize the algorithm. Table 3 also reveals that the speedups achieved by the GPU implementation were slightly increased when the SAD distance was used to implement the G-ATDCA. This suggests that the spectral angle calculations required for this distance can be efficiently parallelized in the considered GPU (in particular, calculation of cosines in the GPU was very efficient).

It is also clear from Table 3 that the best speedup results were obtained when the SID distance was used to implement the G-ATDCA. Specifically, we measured a speedup of $71,91$ when comparing the processing time measured in the GPU with the processing time measured in the CPU. This is mainly due to the fact that the logarithm operations required to implement the SID distance can be executed very effectively in the GPU. Although the speedup achieved in this case is no less than impressive, the final processing time for the G-ATDCA implemented using this distance is still above two minutes after parallelization, which indicates that the use of the SID distance introduces additional complexity in both the serial and parallel implementations of the ATDCA algorithm. Similar comments apply to the parallel version of G-RX, which also takes more than 2 minutes to complete its calculations after parallelization. This is due to swapping problems in both the serial implementation (i.e., an excessive traffic between disk pages and memory pages was observed, probably resulting from an ineffective allocation of resources in our G-RX implementation). This aspect should be improved in future developments of the parallel G-RX algorithm.

Summarizing, the experiments reported on Table 3 indicate that the considered GPU can significantly increase the performance of the considered algorithms, providing speedup ratios on the order of 10 for G-ATDCA (for most of the considered distances) and on the order of 14 for G-RX, although this algorithm should still be further optimized for more efficient execution on GPUs. When G-ATDCA was implemented using OSP as a baseline distance, the speedup decreased since the parallel matrix inverse and transpose operations are not fully optimized in our GPU implementation. On the other hand, when G-ATDCA was implemented using SID as a baseline distance, the speedup boosted over 71 due to the optimized capability of the considered GPU to compute logarithm-type operations in paralell. Overall, the best processing times achieved in experiments were on the order of 9 seconds. These response times are not strictly in real-time since the cross-track line scan time in AVIRIS, a push-broom instrument,[14] is quite fast (8.3 msec). This introduces the need to process the full image cube (614 lines, each made up of 512 pixels with 224 spectral bands) in about 5 seconds to achieve fully achieve real-time performance. Although the proposed implementations can still be optimized, Table 3 indicates that significant speedups can be obtained in most cases 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 (defined as the maximum load allowed in the airborne or satellite platform that carries the imaging instrument).

5. CONCLUSIONS AND FUTURE RESEARCH

This paper described several parallel algorithms for target and anomaly detection in hyperspectral image analysis. As a case study of specific issues involved in the exploitation of an automatic algorithm for target detection and classification (ATDCA), we have investigated the impact of including several distance measures in the design of different parallel versions of this algorithm. This paper has also developed a new parallel version of a well-known anomaly detection algorithm (RX). The parallel algorithms have been implemented in a NVidia^{TM} GeForce 9800 GX2 GPU. Experimental results, oriented towards analyzing the target/anomaly detection accuracy and parallel performance of the proposed parallel algorithms, have been presented and thoroughly discussed in the context of a real defense and security application: the analysis of hyperspectral data collected by NASA's 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. Our experimental assessment indicates that GPU hardware devices may offer important advantages in defense and security applications that demand a response in real-time, mainly due to the low weight and compact size of these devices, and to their capacity to provide high performance computing at very low costs. Experiments with radiation-hardened GPU devices will be required in future research in order to evaluate the possibility of adapting the proposed parallel algorithms to hardware devices which have been already certified by international agencies.

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. C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification.*, Norwell, MA: Kluwer, 2003.
2. R. O. Green, "Imaging spectroscopy and the airborne visible infrared imaging spectrometer (aviris)," *Remote Sensing of Environment* **65**, pp. 227–248, 1998.
3. H. Ren and C.-I. Chang, "Automatic spectral target recognition in hyperspectral imagery," *IEEE Trans. Aerosp. Electron. Syst.* **39**, pp. 1232–1249, 2003.
4. I. Reed and X.Yu, "Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distrihution.," *IEEE Trans. Acoustics, Speech and Signal Processing* **38**, pp. 1760–1770, 1990.
5. J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection approach," *IEEE Trans. Geosci. Remote Sens.* **32**, pp. 779–785, 1994.
6. D. Manolakis, D. Marden, and G. A. Shaw, "Hyperspectral image processing for automatic target detection applications," *MIT Lincoln Laboratory Journal* **14**, pp. 79–116, 2003.
7. A. Paz, A. Plaza, and S. Blazquez, "Parallel implementation of target and anomaly detection algorithms for hyperspectral imagery," *Proc. IEEE Geosci. Remote Sens. Symp.* **2**, pp. 589–592, 2008.
8. J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geosci. Remote Sens. Lett.* **43**, pp. 441–445, 2007.
9. 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.
10. A. Paz, A. Plaza, and J. Plaza, "Comparative analysis of different implementations of a parallel algorithm for automatic target detection and classification of hyperspectral images," *Proc. SPIE* **7455**, pp. 1–12, 2009.
11. J. C. Tilton, W. T. Lawrence, and A. Plaza, "Utilizing hierarchical segmentation to generate water and snow masks to facilitate monitoring of change with remotely sensed image data," *GIScience and Remote Sensing* **43**, pp. 39–66, 2006.
12. J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis: An Introduction*, Springer, 2006.
13. C.-I. Chang and Q. Du, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.* **42**(3), pp. 608–619, 2004.
14. 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.