GPU Implementation of an Automatic Target Detection and Classification Algorithm for Hyperspectral Image Analysis

Sergio Bernabé, Sebastián López, *Member, IEEE*, Antonio Plaza, *Senior Member, IEEE*, and Roberto Sarmiento

***Abstract*—The detection of (moving or static) targets in remotely sensed hyperspectral images often requires real-time responses for swift decisions that depend upon high computing performance of algorithm analysis. The automatic target detection and classification algorithm (ATDCA) has been widely used for this purpose. In this letter, we develop several optimizations for accelerating the computational performance of ATDCA. The first one focuses on the use of the Gram–Schmidt orthogonalization method instead of the orthogonal projection process adopted by the classic algorithm. The second one is focused on the development of a new implementation of the algorithm on commodity graphics processing units (GPUs). The proposed GPU implementation properly exploits the GPU architecture at low level, including shared memory, and provides coalesced accesses to memory that lead to very significant speedup factors, thus taking full advantage of the computational power of GPUs. The GPU implementation is specifically tailored to hyperspectral imagery and the special characteristics of this kind of data, achieving real-time performance of ATDCA for the first time in the literature. The proposed optimizations are evaluated not only in terms of target detection accuracy but also in terms of computational performance using two different GPU architectures by NVIDIA: Tesla C1060 and GeForce GTX 580, taking advantage of the performance of operations in single-precision floating point. Experiments are conducted using hyperspectral data sets collected by three different hyperspectral imaging instruments. These results reveal considerable acceleration factors while retaining the same target detection accuracy for the algorithm.**

***Index Terms*—Automatic target detection and classification algorithm (ATDCA), commodity graphics processing units (GPUs), Gram–Schmidt (GS) orthogonalization, hyperspectral imaging.**

I. Introduction

HYPERSPECTRAL target detection and identification are very important tasks in remotely sensed hyperspectral data exploitation [1]. Over the last few years, several algorithms have been developed for the purpose of target identification, including the automatic target detection and classification algorithm (ATDCA) [2], an unsupervised fully constrained least squares (UFCLS) algorithm [3], an iterative error analysis (IEA) algorithm [4], or the well-known Reed–Xiaoli (RX) algorithm developed by Reed and Xiaoli for anomaly detection purposes [5]. The ATDCA algorithm finds a set of spectrally distinct target pixel vectors using the concept of orthogonal subspace projection (OSP) [6] in the spectral domain. On the other hand, the UFCLS algorithm generates a set of distinct targets using the concept of least square-based error minimization. The IEA uses a similar approach but with a different initialization condition. The RX algorithm is based on the well-known Mahalanobis distance. Many other target/anomaly detection algorithms use different concepts, such as background modeling and characterization [7]. Quantitative and comparative assessments of target detection algorithms reveal good performance of ATDCA with regard to other approaches in terms of detection accuracy and computational performance [1]. Depending on the complexity and dimensionality of the hyperspectral data, the aforementioned algorithms may be computationally very expensive, a fact that limits the possibility of utilizing those algorithms in time-critical applications [8]. Despite the growing interest in parallel hyperspectral imaging research, only a few parallel implementations of automatic target detection algorithms for hyperspectral data exist in the open literature [9]. With the recent explosion in the amount and dimensionality of hyperspectral imagery, parallel processing is a requirement in many remote sensing missions.

In this letter, we develop the first real-time implementation of the ATDCA algorithm for remotely sensed hyperspectral data exploitation. It is based on two optimizations: 1) use of the Gram–Schmidt (GS) orthogonalization method instead of the OSP process adopted by the classic algorithm, which has already demonstrated its effectiveness when applied to the vertex component analysis endmember extraction algorithm [10], and 2) the development of an efficient implementation of the algorithm on commodity graphics processing units (GPUs), a low-weight hardware platform that offers a tremendous potential to bridge the gap toward real-time analysis of remotely sensed hyperspectral data [11]–[13]. The proposed implementation exploits the GPU architecture at low level, including shared memory, and provides coalesced accesses to memory that lead to very significant speedup factors, thus taking full advantage of the computational power of GPUs. The remainder of this letter is organized as follows. Section II describes the original ATDCA and the adopted GS optimization. Section III describes a new and fully optimized GPU implementation of this algorithm. Section IV evaluates the proposed GPU implementation in terms of target detection accuracy and computational performance. Section V concludes this letter with some remarks and future research lines.

Manuscript received November 25, 2011; revised March 17, 2012; accepted April 27, 2012. This work was supported by the Spanish Ministry of Science and Innovation CEOS-SPAIN project (Reference AYA2011-29334-C02-02) and DREAMS project (Reference TEC2011-28666-C04-04).

S. Bernabé and A. Plaza are with the Hyperspectral Computing Laboratory, University of Extremadura, 10071 Cáceres, Spain (e-mail: sergiobernabe@unex.es; aplaza@unex.es).

S. López and R. Sarmiento are with the Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, 35017 Tafira Baja, Spain (e-mail: seblopez@iuma.ulpgc.es; roberto@iuma.ulpgc.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LGRS.2012.2198790

II. ATDCA and Its GS Optimization

The original ATDCA algorithm is based on OSP concepts and will be referred to hereinafter as ATDCA-OSP. This method is summarized in Algorithm 1, where $\boldsymbol{U}$ is a matrix of spectral signatures, $\boldsymbol{U}^{\mathrm{T}}$ is the transpose of this matrix, and $\boldsymbol{I}$ is the identity matrix.

Algorithm 1 Pseudocode of ATDCA-OSP

1: **INPUTS**: $\boldsymbol{F} \in \boldsymbol{R}^n$ and t;
   % $\boldsymbol{F}$ denotes an n-dimensional hyperspectral image with r pixels, and t denotes the number of targets to be detected
2: $\boldsymbol{U} = [\boldsymbol{x}_0|0|, \ldots, |0|]$;
   % $\boldsymbol{x}_0$ is the pixel vector with maximum length in $\boldsymbol{F}$
3: **for** $i = 1$ to $t-1$ **do**
4: $P_{\boldsymbol{U}}^{\perp} = \boldsymbol{I} - \boldsymbol{U}(\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U})^{-1}\boldsymbol{U}^{\mathrm{T}}$;
   % $P_{\boldsymbol{U}}^{\perp}$ is a vector orthogonal to the subspace spanned by the columns of $\boldsymbol{U}$
5: $\boldsymbol{v} = P_{\boldsymbol{U}}^{\perp}\boldsymbol{F}$;
   % $\boldsymbol{F}$ is projected onto the direction indicated by $P_{\boldsymbol{U}}^{\perp}$
6: $i = \arg\max_{\{1,\ldots,r\}} \boldsymbol{v}[:, i]$;
   % The maximum projection value is found, where r denotes the total number of pixels in the hyperspectral image and the operator ":" denotes "all elements"
7: $\boldsymbol{x}_i \equiv \boldsymbol{U}[:, i+1] = \boldsymbol{F}[:, i]$;
   % The target matrix is updated
8: **end for**
9: **OUTPUT**: $\boldsymbol{U} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}]$;

An optimization of the ATDCA-OSP algorithm consists of using the GS method (instead of the OSP) for orthogonalization purposes. This version, called ATDCA-GS hereinafter, selects a finite set of linearly independent vectors $\boldsymbol{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k\}$ in the inner product space $\boldsymbol{R}^n$ in which the original hyperspectral image $\boldsymbol{F}$ is defined and generates an orthogonal set of vectors $\boldsymbol{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ which spans the same k-*dimensional* subspace of $\boldsymbol{R}^n$ $(k \leq n)$ as $\boldsymbol{A}$. In particular, $\boldsymbol{B}$ is obtained as follows:

$$
\begin{aligned}
\boldsymbol{b}_1 &= \boldsymbol{a}_1, & \boldsymbol{e}_1 &= \frac{\boldsymbol{b}_1}{\| \boldsymbol{b}_1 \|} \\
\boldsymbol{b}_2 &= \boldsymbol{a}_2 - proj_{\boldsymbol{b}_1}(\boldsymbol{a}_2), & \boldsymbol{e}_2 &= \frac{\boldsymbol{b}_2}{\| \boldsymbol{b}_2 \|} \\
\boldsymbol{b}_3 &= \boldsymbol{a}_3 - proj_{\boldsymbol{b}_1}(\boldsymbol{a}_3) - proj_{\boldsymbol{b}_2}(\boldsymbol{a}_3), & \boldsymbol{e}_3 &= \frac{\boldsymbol{b}_3}{\| \boldsymbol{b}_3 \|} \\
\boldsymbol{b}_4 &= \boldsymbol{a}_4 - proj_{\boldsymbol{b}_1}(\boldsymbol{a}_4) \\
&\quad - proj_{\boldsymbol{b}_2}(\boldsymbol{a}_4) - proj_{\boldsymbol{b}_3}(\boldsymbol{a}_4), & \boldsymbol{e}_4 &= \frac{\boldsymbol{b}_4}{\| \boldsymbol{b}_4 \|} \\
&\vdots & &\vdots \\
\boldsymbol{b}_k &= \boldsymbol{a}_k - \sum_{j=1}^{k-1} proj_{\boldsymbol{b}_j}(\boldsymbol{a}_k), & \boldsymbol{e}_k &= \frac{\boldsymbol{b}_k}{\| \boldsymbol{b}_k \|}
\end{aligned} \tag{1}
$$

where the projection operator is defined in (2), in which $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ denotes the inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$

$$
proj_{\boldsymbol{b}}(\boldsymbol{a}) = \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\langle \boldsymbol{b}, \boldsymbol{b} \rangle} \boldsymbol{b}. \tag{2}
$$

The sequence $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$ in (1) represents the set of orthogonal vectors generated by the GS method, and thus, the normalized vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k$ in (1) form an orthonormal set. As far as $\boldsymbol{B}$ spans the same k-dimensional subspace of $\boldsymbol{R}^n$ as $\boldsymbol{A}$, an additional vector $\boldsymbol{b}_{k+1}$ computed by following the procedure stated in (1) is also orthogonal to all the vectors included in $\boldsymbol{A}$ and $\boldsymbol{B}$. This algebraic assertion constitutes the cornerstone of the ATDCA-GS algorithm, whose pseudocode is represented in Algorithm 2.

As it can be observed from Algorithm 2, the computation of the orthogonal projector $P_{\boldsymbol{U}}^{\perp}$ starts by first initializing it to an arbitrary n-dimensional vector (step 6 of Algorithm 2). Then, the GS orthogonal set $\boldsymbol{B}$ is generated from the targets already detected in the image (steps 7–10 of Algorithm 2). The main modification of this method consists in fixing the vector $\boldsymbol{w}$ to $[\boldsymbol{1}, \ldots, \boldsymbol{1}]^{\mathrm{T}}$ rather than generating a random vector at each iteration. As far as the underlying reason for generating a random vector is only to get a nonnull projection, this can also be achieved by fixing $\boldsymbol{w}$ in the way that it has been previously mentioned, which allows the reduction of the computational cost of the ATDCA-GS algorithm by avoiding the generation of random vectors. Finally, $P_{\boldsymbol{U}}^{\perp}$ is updated in steps 11–14 of Algorithm 2 by forcing it to be orthogonal to all the vectors in $\boldsymbol{B}$ and, as a consequence, to all the targets already stored in $\boldsymbol{U}$. At this point, it is important to emphasize that the n components of the orthogonal projector $P_{\boldsymbol{U}}^{\perp}$ could be initialized to any other values, since this would not affect its orthogonality with respect to the targets already extracted from the input hyperspectral image $\boldsymbol{F}$. Last but not least, we emphasize that the ATDCA-GS is ideally suited for hardware implementation as it avoids the inverse operation which is very difficult to implement in hardware and obtains the same functionality with simpler operations.

Algorithm 2 Pseudocode of ATDCA-GS

1: **INPUTS**: $\boldsymbol{F} \in \boldsymbol{R}^n$ and t;
   % $\boldsymbol{F}$ denotes an n-dimensional hyperspectral image with r pixels, and t denotes the number of targets to be detected
2: $\boldsymbol{U} = [\boldsymbol{x}_0|0|, \ldots, |0|]$;
   % $\boldsymbol{x}_0$ is the pixel vector with maximum length in $\boldsymbol{F}$
3: $\boldsymbol{B} = [0|0|, \ldots, |0|]$;
   % $\boldsymbol{B}$ is an auxiliary matrix for storing the orthogonal base generated by the GS process
4: **for** $i = 1$ to $t-1$ **do**
5: $\boldsymbol{B}[:, i] = \boldsymbol{U}[:, i]$;
   % the ith column of $\boldsymbol{B}$ is initialized with the target computed in the last iteration (here, the operator ":" denotes "all elements")
6: $P_{\boldsymbol{U}}^{\perp} = [\boldsymbol{1}, \ldots, \boldsymbol{1}]$;
7: **for** $j = 2$ to i **do**
8: $proj_{\boldsymbol{B}[:,j-1]}(\boldsymbol{U}[:, i]) = \boldsymbol{U}[:, i]^{\mathrm{T}}\boldsymbol{B}[:, j-1]/\boldsymbol{B}[:, j-1]^{\mathrm{T}}\boldsymbol{B}[:, j-1]\boldsymbol{B}[:, j-1]$;
9: $\boldsymbol{B}[:, i] = \boldsymbol{B}[:, i] - proj_{\boldsymbol{B}[:,j-1]}(\boldsymbol{U}[:, i])$;
   % The ith column of $\boldsymbol{B}$ is updated
10: **end for** j
   % The computation of $\boldsymbol{B}$ is finished for the current iteration of the main loop
11: **for** $k = 1$ to i **do**
12: $proj_{\boldsymbol{B}[:,k]}(\boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{B}[:, k]/\boldsymbol{B}[:, k]^{\mathrm{T}}\boldsymbol{B}[:, k]\boldsymbol{B}[:, k]$;
13: $P_{\boldsymbol{U}}^{\perp} = P_{\boldsymbol{U}}^{\perp} - proj_{\boldsymbol{B}[:,k]}(\boldsymbol{w})$;
14: **end for** k

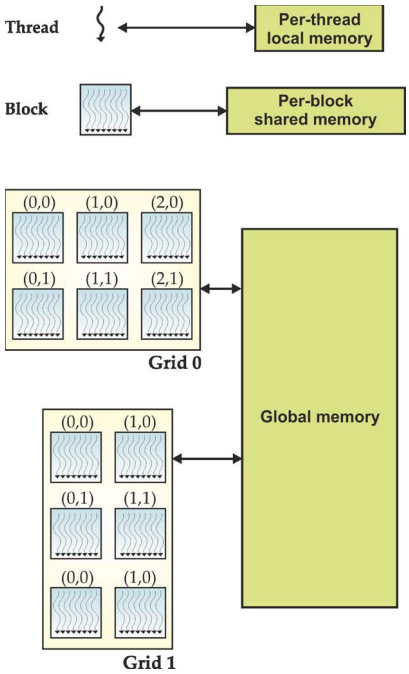

Fig. 1. GPU parallelism at the thread, block, and grid levels.

% The computation of $P_{\boldsymbol{U}}^{\perp}$ is finished for the current iteration of the main loop

15: $\boldsymbol{v} = P_{\boldsymbol{U}}^{\perp}\boldsymbol{F}$;

% $\boldsymbol{F}$ is projected onto the direction indicated by $P_{\boldsymbol{U}}^{\perp}$

16: $i = \arg\max_{\{1,\ldots,r\}} \boldsymbol{v}[:,i]$;

% The maximum projection value is found, where r denotes the total number of pixels in the hyperspectral image

17: $\boldsymbol{x}_i \equiv \boldsymbol{U}[:,i+1] = \boldsymbol{F}[:,i]$;

% The target matrix is updated

18: **end for**

19: **OUTPUT**: $\boldsymbol{U} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}]$;

III. GPU Implementation

In this section, we describe an efficient implementation of ATDCA-GS on GPUs. It should be noted that a GPU implementation of the original ATDCA-OSP algorithm was presented in [9]. In the context of NVIDIA compute unified device architecture (CUDA)[1] adopted for our implementation, GPUs can be abstracted in terms of a stream model, under which all data sets are represented as streams (i.e., ordered data sets) [14]. The architecture of a GPU can be seen as a set of multiprocessors (MPs), where each MP is characterized by a single-instruction multiple-data architecture, i.e., in each clock cycle, each processor executes the same instruction but operating on multiple data streams. Each processor has access to a local shared memory and also to local cache memories in the MP, while the MPs have access to the global GPU (device) memory. Algorithms are constructed by chaining so-called kernels which operate on entire streams and which are executed by an MP, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. The kernels can perform a kind of batch processing arranged in the form of a grid of blocks, where each block is composed by a group of threads which share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory (Fig. 1).

Next, we describe the different steps and architecture-related optimizations carried out in the development of the GPU version of the ATDCA-GS algorithm. The first step is related with the proper arrangement of the hyperspectral data in the local GPU memory. In order to optimize accesses, bearing in mind that the ATDCA-GS algorithm uses the pixel vector as the minimum unit of computation, we store the pixel vectors of the hyperspectral image $\boldsymbol{F}$ by columns. Our arrangement is intended to access consecutive wavelength values in parallel by the processing kernels (coalesced accesses to memory). This means that the ith thread of a block will access the ith wavelength component of a pixel vector of the image. This technique is used to maximize global memory bandwidth and minimize the number of bus transactions. Once the hyperspectral image is mapped onto the GPU memory using the aforementioned strategy, a structure is created in which the number of blocks equals the number of pixel vectors in the hyperspectral image divided by the number of threads per block, where the maximum number of supported threads depends on the considered GPU architecture. A kernel is now used to calculate the brightest pixel $\boldsymbol{x}_0$ in $\boldsymbol{F}$. This kernel computes (in parallel) the dot product between each pixel vector and its own transposed version, retaining the pixel that results in the maximum projection value.

Once the brightest pixel in $\boldsymbol{F}$ has been identified, the pixel is allocated as the first column in matrix $\boldsymbol{U}$. In this way, we ensure that memory accesses are coalesced. The algorithm now calculates the orthogonal vectors through the GS method as detailed in Algorithm 2. This operation is performed in the CPU because this method operates on a small data structure and the results can be obtained very quickly. A new kernel is created, in which the number of blocks equals the number of pixel vectors in the hyperspectral image divided by the number of threads per block, where the maximum number of supported threads depends on the considered GPU architecture. This kernel is now applied to project the orthogonal vector onto each pixel in the image. An important optimization applied at this point involves the effective use of the shared memories, which act as small and very fast cache memories available for the processing elements within the same block. Allocating the data properly in these shared memories is crucial for obtaining good performance in the GPU. In our case, we use these memories to store the most orthogonal vectors obtained at each iteration of ATDCA-GS (this is because these vectors will be accessed every time that the projection onto each pixel of the image is performed). Hence, it is crucial to store these vectors in these small cache memories in order to perform the projection operations much faster and with fewer memory accesses as compared to the case in which these vectors are stored in the main GPU memory. The maximum of all projected pixels is calculated using a separate reduction kernel which also uses shared memory to store each of the projections and obtains the new target $\boldsymbol{x}_1$. The algorithm now extends the target matrix as $\boldsymbol{U} = [\boldsymbol{x}_0\boldsymbol{x}_1]$ and repeats the same process until the desired number of targets (specified by the input parameter t) has been detected. The output of the algorithm is a set of targets $\boldsymbol{U} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}]$.

[1] http://www.nvidia.com/object/cuda_home_new.html

TABLE I
SPECTRAL ANGLE VALUES (IN DEGREES) BETWEEN THE TARGET PIXELS EXTRACTED BY ATDCA-OSP AND ATDCA-GS AND THE KNOWN GROUND TARGETS IN THE AVIRIS WORLD TRADE CENTER SCENE

Version	A	B	C	D	E	F	G	H
ATDCA-OSP	0.00°	14.43°	0.00°	27.38°	20.32°	7.13°	4.15°	31.27°
ATDCA-GS	0.00°	27.16°	0.00°	15.62°	27.81°	3.98°	2.72°	24.26°

TABLE II
SPECTRAL ANGLE VALUES (IN DEGREES) BETWEEN THE TARGET PIXELS EXTRACTED BY ATDCA-OSP AND ATDCA-GS AND THE KNOWN GROUND TARGETS IN THE AVIRIS CUPRITE SCENE

Version	Alunite	Buddingtonite	Calcite	Kaolinite	Muscovite	Average
ATDCA-OSP	4.81°	4.16°	9.52°	10.76°	5.29°	6.91°
ATDCA-GS	5.48°	4.08°	5.87°	11.14°	5.68°	6.45°

IV. EXPERIMENTAL RESULTS

A. Hyperspectral Image Data

Four hyperspectral images are used in our experiments. Two of them were obtained by the National Aeronautics and Space Administration Airborne Visible Infrared Imaging Spectrometer (AVIRIS)[2] over the Cuprite mining district, Nevada, and over the World Trade Center, New York, just 5 days after the terrorist attacks of September 11, 2001. The Cuprite data correspond to a 350 × 350 pixel subset of the sector labeled as f970619t01p02_r02_sc03.a.rfl in the online data, which comprise 188 spectral bands in the range from 400 to 2500 nm and a total size of around 50 MB. The World Trade Center data consist of 614 × 512 pixels, 224 spectral bands, and a total size of (approximately) 140 MB. Another data set collected by the Hyperspectral Digital Imagery Collection Experiment (HYDICE) sensor was used in experiments, which represents a subset of the well-known forest radiance data [1] consisting of 64 × 64 pixels and 169 spectral bands for a total size of 5.28 MB. Finally, we have also used a well-known hyperspectral data set collected by the Reflective Optics Imaging Spectrographic System (ROSIS) over an urban area in the city of Pavia, Italy, which has been also widely used in the literature [8]. It consists of 610 × 340 pixels and 103 spectral bands, for a total size of 40.7 MB.

B. Analysis of Target Detection Accuracy

In this subsection, we compare the performance of ATDCA-GS and ATDCA-OSP implementations using the two AVIRIS scenes for which ground-truth information is available. Table I shows the spectral angle distance (SAD) [1] values (in degrees) between the most similar target pixels detected by ATDCA-OSP and the pixel vectors at the known target positions, labeled from "A" to "H," in the AVIRIS World Trade Center image. The same results are reported for the ATDCA-GS. 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 [15]. As shown by Table I, the ATDCA-OSP and ATDCA-GS extracted targets which were similar, spectrally, to the known ground-truth targets. Both versions were able to perfectly detect the targets labeled as "A" and "C" and had more difficulties in detecting other targets. In the case of targets labeled as "D" to "H," the ATDCA-GS improved the target detection results (lower SAD values in Table I) with regard to the ATDCA-OSP.

Table II shows the SAD values (in degrees) between the most similar target pixels detected by the two considered versions: ATDCA-OSP and ATDCA-GS, and the pixel vectors at the known positions of the target minerals in the AVIRIS Cuprite image. In all cases, the number of target pixels to be detected was set to $t = 19$ after calculating the VD. As shown by Table II, the ATDCA-GS extracted targets which were slightly more similar (on average) to the ground references than those provided by ATDCA-OSP. This indicates that the proposed GS optimization does not penalize the ATDCA algorithm in terms of target detection accuracy.

C. Analysis of Parallel Performance

Two different GPU platforms have been used in our experiments. The first one is the NVIDIA Tesla C1060 GPU,[3] and the second is the NVIDIA GeForce GTX 580 GPU.[4] Both GPUs are connected to an Intel Core i7 920 CPU at 2.67 GHz with eight cores, which uses a motherboard ASUS P6T7 WS SuperComputer. Before analyzing the parallel performance of the proposed GPU implementation, we emphasize that our parallel versions provide exactly the same results as the corresponding serial versions, executed in one of the cores of the i7 920 CPU and implemented using the `GCC` (GNU compiler default) with optimization flag −O3 to exploit data locality and avoid redundant computations. As a result, the only difference between the serial and parallel versions is the time they need to complete their calculations. The reported GPU times are the mean of ten executions in each platform (the measured times were always very similar, with differences—if any—on the order of only a few milliseconds).

Table III reports the processing times obtained for the GPU implementations of ATDCA-OSP and ATDCA-GS on the two considered GPU architectures and for the four considered hyperspectral scenes. It should be noted that the GPU implementation of ATDCA-OSP corresponds to an improvement of the one described in [9] (some kernels were further optimized), while the GPU implementation of ATDCA-GS is the one described in this letter. As shown by Table III, the ATDCA-GS achieved significant speedups in both GPU architectures and offered significant improvements with regard to the previously available GPU implementation of ATDCA-OSP. The slightly lower speedups achieved for the HYDICE image (5.28 MB in size) compared to those obtained for the AVIRIS World Trade Center (140 MB in size) indicate that the ATDCA-GS provides more significant acceleration factors as the amount of data to be processed is larger. The processing times achieved by the GPU implementation of ATDCA-GS are strictly in real time for the AVIRIS data. The cross-track line scan time in AVIRIS, a push-broom instrument, is quite fast (8.3 ms to collect 512 full-pixel vectors). This introduces the need to process the AVIRIS World Trade Center scene (614 × 512 pixels and 224 spectral bands) in less than 5.09 s and the AVIRIS Cuprite scene (350 × 350 pixels and 188 spectral bands) in less than 1.98 s in order to achieve real-time performance. As noted in Table III, all the proposed GPU implementations of ATDCA-GS are well below 1 s in processing time, including the loading times and the data transfer times from CPU to GPU and vice-versa. This represents a significant improvement with regard to previous GPU implementations of ATDCA [9], [13].

[2] http://aviris.jpl.nasa.gov

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

[4] http://www.nvidia.com/object/product-geforce-gtx-580-us.html

TABLE III
PROCESSING TIMES (IN SECONDS) AND SPEEDUPS ACHIEVED BY ATDCA-OSP AND ATDCA-GS WITH OPTIMIZATIONS IN TWO DIFFERENT GPUS

	AVIRIS World Trade Center		AVIRIS Cuprite		ROSIS Pavia University		HYDICE Forest Radiance	
	ATDCA-OSP	**ATDCA-GS**	**ATDCA-OSP**	**ATDCA-GS**	**ATDCA-OSP**	**ATDCA-GS**	**ATDCA-OSP**	**ATDCA-GS**
Serial time	512.1120	7.3230	87.9820	1.5950	20.2672	0.6460	2.2341	0.0331
Time Tesla C1060 GPU	51.4626	0.1663	9.0032	0.0560	2.2840	0.0407	0.4523	0.0045
Time GeForce GTX 580 GPU	10.5747	0.1554	1.9947	0.0456	0.5834	0.0308	0.1837	0.0035
Speedup Tesla C1060 GPU	9.95	44.03	9.77	28.48	8.87	15.87	4.94	7.36
Speedup GeForce GTX 580 GPU	48.43	47.12	44.11	35.01	34.74	20.97	12.16	9.46

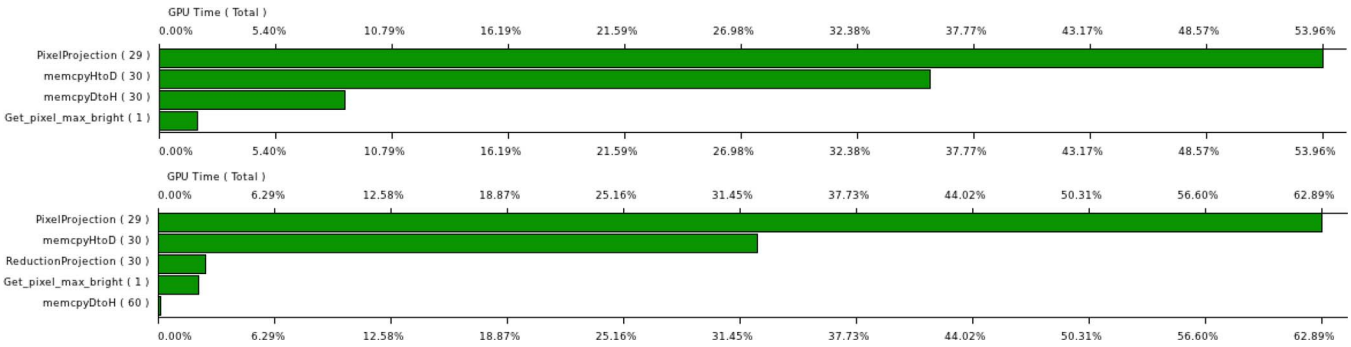

Fig. 2. Summary plot describing the percentage of the total GPU time consumed by memory-transfer operations and by the different kernels used by ATDCA-GS in the NVIDIA GeForce GTX 580 GPU (AVIRIS World Trade Center scene) (top) without and (bottom) with architecture optimizations.

For illustrative purposes, Fig. 2 shows the percentage of the total GPU execution time consumed by memory transfers and by each CUDA kernel (obtained after profiling the ATDCA-GS implementation) along with the number of times that each kernel was invoked (in the parentheses) for the detection of $t = 30$ targets from the AVIRIS World Trade Center scene in the NVIDIA GeForce GTX 580 architecture, with and without the GPU architecture optimizations related with the use of shared memories and coalesced accesses described in Section III. As shown by Fig. 2, the GPU implementation without optimizations uses approximately 55% of the execution time for running the kernels and 45% of the time for memory transfers. On the other hand, the version with optimizations significantly decreases the percentage of time devoted to memory transfers and increases the percentage of time used for running the kernels. This indicates that the proposed implementation does not represent a straightforward parallelization effort but, instead, a careful effort to adapt the GPU architecture to the specific issues involved in hyperspectral data processing.

V. CONCLUSION AND FUTURE RESEARCH

In this letter, we have developed the first real-time implementation of an ATDCA, implemented with GS orthogonalization, on GPU architectures. The proposed implementation has been specifically tailored to specific aspects involved in hyperspectral data processing and makes advanced use of the GPU architecture including considerations such as the arrangement of the data in the GPU local and shared memories in order to ensure coalesced memory accesses and low memory-transfer times. Although the results obtained with a variety of hyperspectral images are very encouraging, GPUs are still far from being exploited in real missions due to power consumption and radiation tolerance issues to be addressed in future developments. Future work will also explore how to merge different kernels used to reduce kernel-invoking time.

REFERENCES

[1] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. New York: Kluwer, 2003.
[2] H. Ren and C.-I. Chang, "Automatic spectral target recognition in hyperspectral imagery," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 39, no. 4, pp. 1232–1249, Oct. 2003.
[3] D. Heinz and C.-I. Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 3, pp. 529–545, Mar. 2001.
[4] R. A. Neville, K. Staenz, T. Szeredi, J. Lefebvre, and P. Hauff, "Automatic endmember extraction from hyperspectral data for mineral exploration," in *Proc. 21st Can. Symp. Remote Sens.*, 1999, pp. 21–24.
[5] I. Reed and X. Yu, "Adaptive multiple-band CFAR detection of an optical pattern with unknown spectral distribution," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 38, no. 10, pp. 1760–1770, Oct. 1990.
[6] J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection," *IEEE Trans. Geosci. Remote Sens.*, vol. 32, no. 4, pp. 779–785, Jul. 1994.
[7] G. Shaw and D. Manolakis, "Signal processing for hyperspectral image exploitation," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 12–16, Jan. 2002.
[8] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*. Boca Raton, FL: Taylor & Francis, 2007.
[9] A. Paz and A. Plaza, "Clusters versus GPUs for parallel automatic target detection in remotely sensed hyperspectral images," *EURASIP J. Adv. Signal Process.*, vol. 2010, pp. 915 639-1–915 639-18, Feb. 2010.
[10] S. Lopez, P. Horstrand, G. M. Callico, J. F. Lopez, and R. Sarmiento, "A low-computational-complexity algorithm for hyperspectral endmember extraction: Modified vertex component analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 9, no. 3, pp. 502–506, May 2012.
[11] H. Yang, Q. Du, and G. Chen, "Unsupervised hyperspectral band selection using graphics processing units," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 4, no. 3, pp. 660–668, Sep. 2011.
[12] E. Christophe, J. Michel, and J. Inglada, "Remote sensing processing: From multicore to GPU," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 4, no. 3, pp. 643–652, Sep. 2011.
[13] S. Sanchez, A. Paz, G. Martin, and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units," *Concurrency Comput. Pract. Exp.*, vol. 23, no. 13, pp. 1538–1557, Sep. 2011.
[14] 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, no. 3, pp. 441–445, Jul. 2007.
[15] Q. Du and C.-I. Chang, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 608–619, Mar. 2004.