GPU Implementation of JPEG2000 for Hyperspectral Image Compression

Milosz Ciznicki[a], Krzysztof Kurowski[a] and [b]Antonio Plaza
[a]Poznan Supercomputing and Networking Center
Noskowskiego 10, 61-704 Poznan, Poland.
[b]Hyperspectral Computing Laboratory,
Department of Technology of Computers and Communications,
University of Extremadura, Avda. de la Universidad s/n.
10071 Cáceres, Spain

ABSTRACT

Hyperspectral image compression has received considerable interest in recent years due to the enormous data volumes collected by imaging spectrometers for Earth Observation. JPEG2000 is an important technique for data compression which has been successfully used in the context of hyperspectral image compression, either in lossless and lossy fashion. Due to the increasing spatial, spectral and temporal resolution of remotely sensed hyperspectral data sets, fast (onboard) compression of hyperspectral data is becoming a very important and challenging objective, with the potential to reduce the limitations in the downlink connection between the Earth Observation platform and the receiving ground stations on Earth. For this purpose, implementation of hyperspectral image compression algorithms on specialized hardware devices are currently being investigated. In this paper, we develop an implementation of the JPEG2000 compression standard in commodity graphics processing units (GPUs). These hardware accelerators are characterized by their low cost and weight, and can bridge the gap towards on-board processing of remotely sensed hyperspectral data. Specifically, we develop GPU implementations of the lossless and lossy modes of JPEG2000. For the lossy mode, we investigate the utility of the compressed hyperspectral images for different compression ratios, using a standard technique for hyperspectral data exploitation such as spectral unmixing. In all cases, we investigate the speedups that can be gained by using the GPU implementations with regards to the serial implementations. Our study reveals that GPUs represent a source of computational power that is both accessible and applicable to obtaining compression results in valid response times in information extraction applications from remotely sensed hyperspectral imagery.

Keywords: Hyperspectral image compression, JPEG2000, commodity graphics processing units (GPUs).

1. INTRODUCTION

Hyperspectral imaging instruments are capable of collecting hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth.[1] For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS), which is able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of reflected light in an area 2 to 12 kilometers wide and several kilometers long, using 224 spectral bands.[2]

One of the main problems in hyperspectral data exploitation is that, during the collection of hyperspectral data, several GBs of multidimensional data volume is generated and set to the ground stations on Earth.[3] The downlink connection between the observation stations and receiving ground stations is limited, so different compression methods are employed.[4] Up to date, lossless compression techniques are the tool of choice, but the best lossless compression ratio reported in hyperspectral image analysis is around 3:1.[4] Due to the increasing spatial, spectral and temporal resolution of remotely sensed hyperspectral data sets, techniques with better

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

High-Performance Computing in Remote Sensing, edited by Bormin Huang, Antonio J. Plaza,
Proc. of SPIE Vol. 8183, 81830H · © 2011 SPIE · CCC code: 0277-786X/11/$18 · doi: 10.1117/12.897386

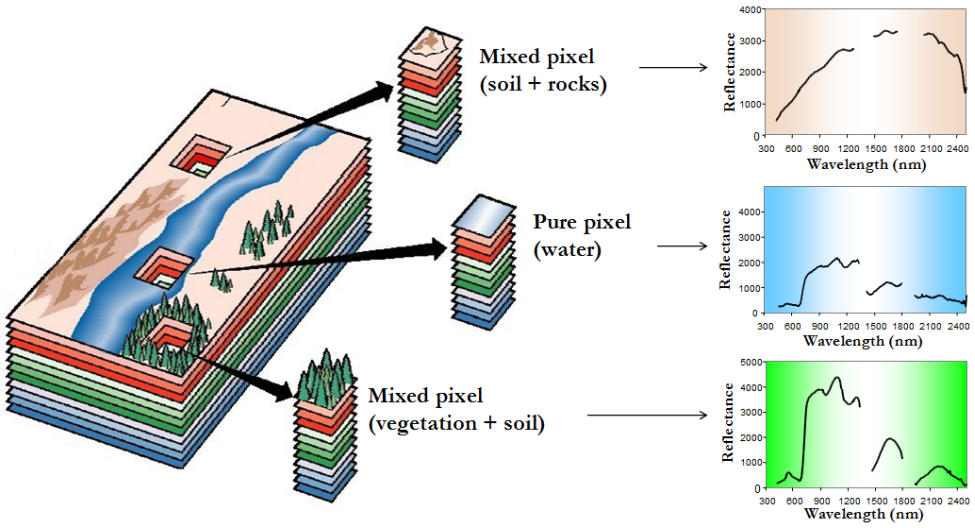

Figure 1. The mixture problem in hyperspectral data analysis.

compression ratios are needed and lossy compression becomes a reasonable alternative.[5] It turns out that JPEG2000[6] has been successfully used in the context of hyperspectral image compression, either in lossless and lossy fashion. Hence, it can be used to evaluate the impact of lossy compression on different techniques for hyperspectral data exploitation.

An important issue that has not been widely investigated in the past is the impact of lossy compression on spectral unmixing applications,[7] which are the tool of choice in order to deal with the phenomenon of mixed pixels,[8] i.e. pixels containing different macroscopically pure spectral substances, as illustrated in Fig. 1. In hyperspectral images, mixed spectral signatures may be collected due to several reasons. First, if the spatial resolution of the sensor is not fine enough to separate different pure signature classes at a macroscopic level, these can jointly occupy a single pixel, and the resulting spectral measurement will be a composite of the individual pure spectra, often called *endmembers* in hyperspectral analysis terminology.[9] Second, mixed pixels can also result when distinct materials are combined into a homogeneous or intimate mixture, and this circumstance occurs independently of the spatial resolution of the sensor.[7]

Although the unmixing chain maps nicely to high performance computing systems such as commodity clusters,[10] these systems are difficult to adapt to on-board processing requirements introduced by applications with real-time constraints 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 such as commodity graphics processing units (GPUs)[11] are essential to reduce mission payload. In this regard, the emergence of GPUs now offers a tremendous potential to bridge the gap towards real-time analysis of remotely sensed hyperspectral data.[12–18]

In this paper we develop an implementation of the JPEG2000 compression standard in commodity graphics processing units (GPUs) for hyperspectral data exploitation. Specifically, we develop GPU implementations of the lossless and lossy modes of JPEG2000. For the lossy mode, we investigate the utility of the compressed hyperspectral images for different compression ratios, using spectral unmixing as a case study. We also investigate the speedups that can be gained by using the GPU implementations with regards to the serial implementations in both the lossless and lossy modes. The remainder of the paper is organized as follows. Section 2 presents the JPEG2000 compression framework. Section 3 presents its GPU implementation. Section 4 first presents the hyperspectral data sets used for evaluation purposes, then briefly introduces the considered hyperspectral unmixing chain, and finally analyzes the proposed GPU implementation of JPEG2000 in terms of both unmixing accuracy (in the lossy mode) and computational performance (in both modes). Section 5 concludes with some remarks and hints at plausible future research.

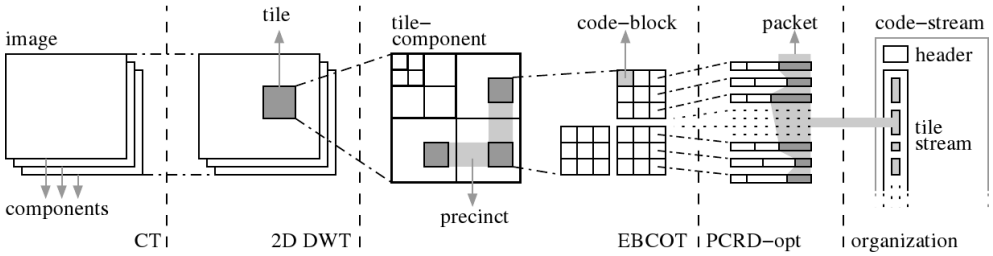

Figure 2. JPEG2000 data partitioning, coding and code-stream organization.

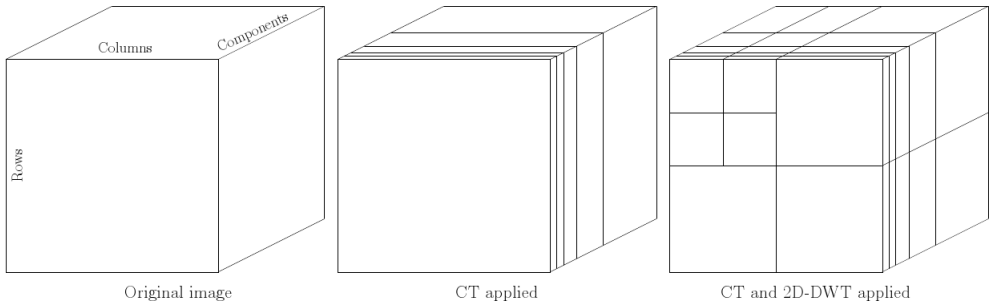

Figure 3. The hybrid scheme for 3D decorrelation. 4 levels for the CT and 2 levels for the ST.

2. OVERVIEW OF JPEG2000

The JPEG2000 standard is divided into several and incremental parts. Part 1[19] defines the core coding system and two basic image file formats. Part 2[20] specifies a set of extensions to the core coding system, such as spectral (inter-component) decorrelation and the use of different wavelet kernels, as well as a more extensible file format. These two first parts are the ones that are going to focus on along this paper. The rest of parts introduce some extensions for different applications. For example, Part 10 (named also JP3D)[21] is concerned with the coding of three-dimensional (3D) data.

Figure 2 shows an overview of the data partitioning made by the core coding system of JPEG2000, and how the elements of a three-component image (such as a color RGB image) are encoded and distributed. The first step in the JPEG200 algorithm, not shown in the figure, is a level offset to guarantee that all the samples are signed. This is a requirement of the transform machinery that is going to be applied. After that, a CT (Component Transform, for example, a RGB to a YUV color transform) removes the inter-component redundancy that could be found in the image. The result of this stage is a new image in other domain, with the same number of components and samples per component. Next, as can be observed in the figure, each component of the image is divided into rectangular areas called tiles. Tiling is useful for compound images because the encoding parameters of each tile can be selected taking into account its characteristics. However, the tiling is rarely used in natural images due to the artifacts produced around the edges of the tiles, having commonly only one tile per component.

JPEG2000 allows the use of different decomposition patterns in the component domain, although the default one is the hybrid scheme (see Fig. 3). In the hybrid decomposition, a dyadic 1D-DWT (Discrete Wavelet Transform) is first applied to the component domain, and then a dyadic 2D-DWT, denoted by ST (Spatial Transform) in the rest of this paper, is applied to each tile-component. Currently, the dyadic DWT is widely used in the processing of scalable image contents because it facilitates the resolution scalability and improves the encoding efficiency, removing the intra-component (spatial) redundancy. In the example of Fig. 2 we can see how four different resolution levels are generated (remarked in blue) when a ST of three iterations is applied to a tile. These resolution levels are commonly referred by positive integer numbers starting from 0 for the highest one, the original image size.

JPEG2000 provides two working modes: lossy and lossless. The first one offers a better encoding efficiency at low bit-rates. When no loss of information is allowed, the lossless mode can be selected. A fixed point (integer) version of the DWT is used in this case. The resultant wavelet coefficients are grouped into rectangular areas (e.g. 64×64 coefficients) called code-blocks, that are encoded independently by the EBCOT (Embedded Block

Coding with Optimal Truncation) algorithm.[22] In order to manage the image information more easily, the code-blocks related to the same rectangular location, within the same resolution level, are grouped into precincts. ROI (Region Of Interest) and the spatial scalabilities are achieved in JPEG2000 by means of the precincts. The compressed bit-streams of the code-blocks can be divided into a specific number of contiguous segments, or quality layers, by the PCRD-opt (Post-Compression Rate-Distortion Optimization) rate-allocation algorithm. The segments of all the code-blocks of a precinct associated to the same quality layer are stored in a packet. The packet is the storing unit in JPEG2000 and it is associated to a quality layer (L), a precinct (P), a resolution level (R), and a tile-component (C). The word formed by this four letters specifies the progression order used to store the image packets, existing five different possibilities: LRCP, RLCP, RPCL, PCRL and CPRL.

The distortion of the decoded image decreases as the amount of decoded data (packets) increases. The LRCP progression provides a fine-grain quality scalability mode. A sequentially decoded RLCP or RPCL image will produce a reconstruction of incremental resolution. The PCRL progression is useful in scan-based systems, like printers. Finally, an CPRL compressed image will be restored, component by component.

The most basic file format defined in the standard, in Part 1, contains only the code-stream of an image (see Fig. 2). This is composed by all the packets of the image and a set of markers with additional information. Markers can be located at any place in the code-stream, however the most important are included in the header. The image files with this format usually have the extension J2C or J2K. Part 1 also defines a more complex file format based on "boxes". This format allows the coder to include additional information such as color palettes or meta-data. All the information is organized in boxes, contiguous segments of data, whose content is identified by a four-bytes code located at its header. It is possible to define a complex hierarchical structure since a box can contain many other boxes. The extension used to identify the image files with this format is JP2.

The box-based structure of the JP2 format is extensible. Just defining new four-bytes identifiers would allow to include new kind of boxes within an image file, maintaining the backward compatibility (an image viewer that does not understand certain box codes it just ignores them). Part 2 defines a new set of boxes with additional and powerful functionalities. For instance, multiple code-streams can be included within a file, as well as a complex composition scheme (animations, transparency masks, geometric transformations, user definable wavelet kernels, multi-component processing (CT), etc.), which will determine how the image decoding and displaying must be performed. The extension JPX is used to identify those files that contain boxes of Part 2. This is the file format used in our experiments.

3. GPU IMPLEMENTATION

GPUs can be abstracted by assuming a much larger availability of processing cores than in standard CPU processing, with smaller processing capability of the cores and small control units associated to each core (see Fig. 4). Hence, the GPU is appropriate for algorithms that need to execute many repetitive tasks with fine grain parallelism and few coordination between tasks. In the GPU, algorithms are constructed by chaining so-called *kernels*, which define the minimum units of computations performed in the cores. 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, as displayed in Fig. 5, 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. There is a maximum number of threads that a block can contain but the number of threads that can be concurrently executed is much larger (several blocks executed by the same kernel can be managed concurrently, at the expense of reducing the cooperation between threads since the threads in different blocks of the same grid cannot synchronize with the other threads). Finally, Fig. 6 shows the architecture of the GPU, which can be seen as a set of multiprocessors. Each multiprocessor is characterized by a single instruction multiple data (SIMD) architecture, i.e., in each clock cycle each processor of the multiprocessor 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 multiprocessor, while the multiprocessors have access to the global GPU (device) memory.

As mentioned in Section 2 the JPEG2000 standard contains several encoding steps which are done in consecutive manner. The first one is the level offset which is performed on samples of components that are unsigned

only. This procedure involves only subtraction of the same quantity from all samples and as a result is bound to memory transfer on GPU. In order to obtain high efficiency every thread on GPU is responsible for calculations of several samples. It occurred that 16×16 blocks of threads where each threads calculates values for four samples gives the best results. After that the component transform is applied to component domain using 1D-DWT. The 1D-DWT can be realized by iteration of filters with rescaling. This kind of implementation has high complexity, need a lot of memory and computational power. The better way is to use the lifting-based wavelet transform. Lifting-based filtering is done by using four lifting steps, which updates alternately odd or even sample values. During this process the spectral vector data is decomposed to the low pass (even) samples and the high pass (odd) samples. The low pass samples contains the most of the information and high pass samples account the residual information. As a result the high pass samples can be discarded during the compression processes and thus reduce file size. In the GPU implementation every thread is responsible for calculating and loading several samples to the shared memory. During the lifting process, all the samples at the edge of the shared memory array depend on samples which were not loaded to the shared memory. Around a data chunk within a thread block, there is a margin of samples that is required in order to calculate the component chunk. The margin of one block overlaps with adjacent blocks. The width of each margin depends on the side of the data chunk. In order to avoid idle threads, data from the margin is loaded by threads within block. Furthermore, the margins of the blocks on the edges of the image are symmetrically extended to avoid large errors at the boundaries. Before the lifting process samples are reordered to access array elements that are adjacent in the global memory. Next encoding step is tiling. Here the components from the hyperspectral dataset may be tiled, which means that they can be divided into several rectangular non-overlapping blocks, called tiles. The tiles at the edges are sometimes smaller, if tile size is not an integral multiple of the component size. A main advantage of tiling is less memory usage and different tiles can be processed parallel. It could be usefull for very large images, which different tiles could be processed independently on separate GPUs. A serious disadvantage of tiling is that artifacts can appear at the edges of the tiles. As hyperspectral data set easily fits into GPU memory no tiling is used during compression.

The subsequent step in compression process is the ST. The ST can be irreversible or reversible. The irreversible transform is implemented by means of the Daubechies filter (lossy transform). The reversible transformation is implemented by means of the Le Gall filter (lossless transform). This enables an intra-component decorrelation that concentrates the image information in a small and very localized area. By itself the ST does not compress image data. It restructures the image information so it is easier to compress it. During the ST the tile data is decomposed into the horizontal and vertical characteristics. This transform is similar to 1D-DWT in nature, but it is applied in the horizontal (rows) and the vertical (columns) directions which forms two-dimensional transform. Similar to the 1D-DWT, the component data block is loaded to the shared memory, however the processing is done on columns and rows during one kernel invocation including data reordering. As a result a number of kernel invocations and calls to the global memory is reduced. In order to get additional performance improvement every thread read and synchronize several samples from global to shared memory. Furthermore, all threads read adjacent sample values from the shared memory to registers, which are needed to correctly compute output samples. When registers are used, each thread from the block is able to calculate two sample values at one time. It gives more speedup, as memory transactions can be overlapped by arithmetic operations. After the lifting procedure samples are scaled and written to the global memory.

Quantization is only applied in the case of lossy compression. After the ST, all the resulting subbands are quantized, that means that floating point numbers are transformed into integers. Quantization is the main source of information loss, and therefore very important to obtain good compression rates. The quantizer maps several values that are in the range of some interval to one integer value. This results in a reduction of the bit-depth, thus compression. It involves only few computations, as a result each threads is responsible for quantization of 16 samples.

After quantization, the integer wavelet coefficients still contain a lot of redundancy and symmetries. This redundancy is removed by entropy coding and so the data is efficiently packed into a minimal size bit-stream. The problem with highly-compressed, entropy coded data is that few bit errors could completely corrupt the image information. This would be a big problem when JPEG2000 data is transmitted over a noisy communication channel, so each wavelet subband in subdivided into small code-blocks with typical sizes of 32×32 or 64×64.

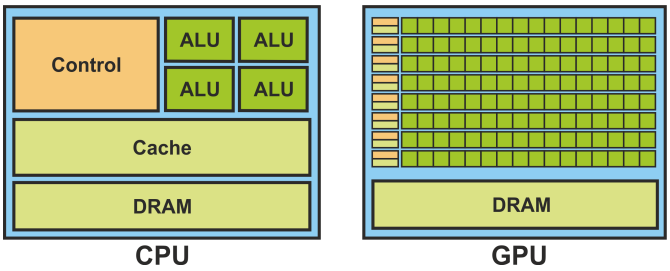

Figure 4. Comparison of CPU versus GPU architecture.

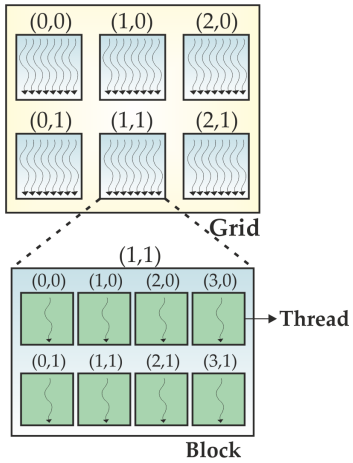

Figure 5. Processing in the GPU: grids made up of blocks with computing threads.

Each of these code-blocks is entropy coded separately which gives potential for parallelization. The process of entropy coding is highly sequential and difficult to efficiently parallelize to more threads, therefore each thread on GPU do entropy coding on whole code-block. However even for small input components it gives enough work to fill all multiprocessors with computations. For instance the input component with size 512×512 and code-blocks size of 32×32 will be spread to $256/8 = 32$ blocks of threads.

The easiest way of rate control would be to change the precision of samples in the quantization process. Naturally, this is only possible for lossy compression. The disadvantage of this way is the high computation consumption, because after every change of precision of the samples, the whole entropy encoding has to be repeated. A much more elegant way is to use the Post-compression rate-distortion algorithm which generates optimal truncation points to minimize the distortion while still obtaining the target bit rate. PCRD algorithm allows to compress hyperspectral data with target bitrate. The algorithm is based on calculation the distortion connected with including next bytes from code-blocks to the output stream. The main idea is to find for the given target output size the total sum of encoded bytes which minimizes the distortion. The calculation of distortion is based on bitplane which is actually encoded. For 16 bit precision hyperspectral component there are 16 bitplanes, from (highest) most significant to (lowest) less significant bitplane. If bit from higher bitplane is skipped during the encoding process, it introduces more distortion. Therefore bits from higher bitplanes have greater chance to be found in output file, because algorithm minimizes summary distortion. Similar to entropy coding each thread on GPU calculates distortions connected with one code-block, because it contains small number of computations.

The last step in compression process is creating and ordering the packets. This basically consists of writing the file and creating the progression order. At the end of the computations all the data have to be saved on the host memory, as a result this step is executed on CPU.

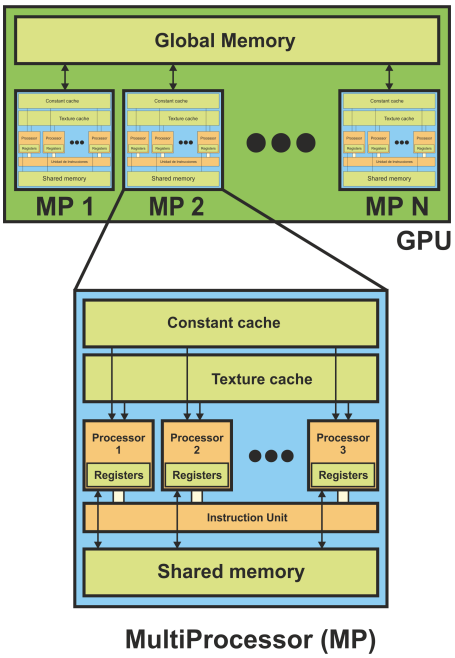

Figure 6. Hardware architecture of the GPU.

4. EXPERIMENTAL RESULTS

4.1 Hyperspectral data set

The hyperspectral data set used in this study is the well-known AVIRIS Cuprite data set, available online* in reflectance units after atmospheric correction. This scene has been widely used to validate the performance of endmember extraction algorithms. The portion used in experiments corresponds to a 350 × 350-pixel subset of the sector labeled as `f970619t01p02_r02_sc03.a.rfl` in the online data. The scene comprises 224 spectral bands between 0.4 and 2.5 μm, with full width at half maximum of 10 nm. Each reflectance value in the scene is represented using 16 bits, for a total image size of approximately 43.92 MB. Prior to the analysis, several bands (specifically, bands 1–2, 105–115 and 150–170) were removed due to water absorption and low SNR in those bands, leaving a total of 188 reflectance channels to be used in the experiments. The Cuprite site is well understood mineralogically,[23,24] and has several exposed minerals of interest including those used in the USGS library considered for the generation of simulated data sets. Five of these laboratory spectra (*alunite*, *buddingtonite*, *calcite*, *kaolinite* and *muscovite*) convolved in accordance with AVIRIS wavelength specifications, will be used to assess endmember signature purity in this work. For illustrative purposes, Fig. 7 shows the image data set considered in experiments and the USGS library signatures.

4.2 Hyperspectral unmixing chain

In order to evaluate the impact of lossy compression on hyperspectral data quality, we follow an exploitation-based approach which consists of applying a spectral unmixing chain to the decompressed hyperspectral data at different quality levels. Let $\mathbf{x}$ be a hyperspectral pixel vector given by a collection of values at different wavelengths. In the context of linear spectral unmixing,[25] such vector can be modeled as:

$$\mathbf{x} \approx \mathbf{E}\mathbf{a} + \mathbf{n} = \sum_{i=1}^{p} \mathbf{e}_i a_i + \mathbf{n}, \tag{1}$$

where $\mathbf{E} = \{\mathbf{e}_i\}_{i=1}^{p}$ is a matrix containing p pure spectral signatures (endmembers), $\mathbf{a} = [a_1, a_2, \cdots, a_p]$ is a p-dimensional vector containing the abundance fractions for each of the p endmembers in $\mathbf{x}$, and $\mathbf{n}$ is a noise

*http://aviris.jpl.nasa.gov/html/aviris.freedata.html

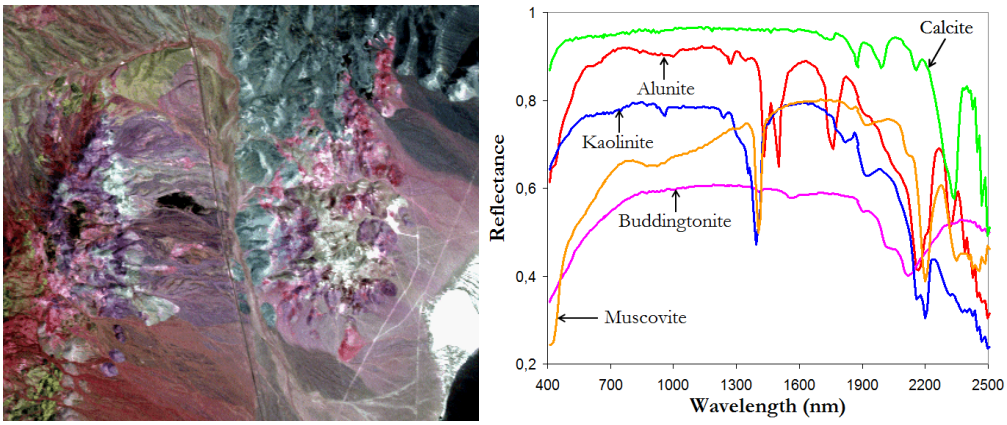

Figure 7. (a) Hyperspectral image over the AVIRIS Cuprite mining district in Nevada. (b) USGS reference signatures used to assess endmember signature purity.

term. Solving the linear mixture model involves: 1) estimating the right subspace for the hyperspectral data, 2) identifying a collection of $\{\mathbf{e}_i\}_{i=1}^{p}$ endmembers in the image, and 3) estimating their abundance in each pixel of the scene. In this work, the dimensional reduction step is performed using principal component analysis (PCA),[26,27] a popular tool for feature extraction in different areas including remote sensing. For the endmember selection part, we rely on the well-known N-FINDR algorithm,[28] which is a standard the hyperspectral imaging community. Finally, abundance estimation is carried out using unconstrained least-squares estimation[25] due to its simplicity. This unmixing chain has been shown to perform in real-time in different GPU platforms.[11] It should be noted that evaluating the accuracy of abundance estimation in real analysis scenarios is very difficult due to the lack of ground-truth information, hence in this work we only evaluate the endmember extraction accuracy of the considered spectral unmixing chain.

4.3 Impact of lossy compression on unmixing accuracy

In this experiment we evaluate the impact of applying lossy JPEG2000-based compression to the considered hyperspectral image in terms of the degradation of unmixing accuracy as we increase the compression ratio. In our experiments, the full scene has been compressed (using different compression ratios) from 0.2 to 2.0 bits per pixel per band (bpppb). This ranges from 80:1 compression ratio (0.2 bpppb) to 2 bpppb (8:1 compression ratio). Our original idea was to compress the hyperspectral data in two different manners: spectral domain and spatial domain. In the spectral domain every pixel (vector) is compressed separately, so the spectral information is utilized. On the other hand compressing pixels separately does not take advantage of redundancy between spatially adjacent pixels. Spatial domain-based compression is done on separate hyperspectral bands. This strategy utilizes spatial data redundancy, but is not specifically intended to preserve spectral properties. Subsequently, in this work we focus mainly on spatial domain decomposition as it exploits spatial information (an important source of correlation in hyperspectral images) and can be applied to the full hyperspectral image without truncation.

The quantitative and comparative algorithm assessment that is performed in this work is intended to measure the impact of compression on the quality of the endmembers extracted by N-FINDR algorithm in the considered unmixing chain. Specifically, the quality of the endmembers extracted from the decompressed scene (i.e., the one obtained after coding/decoding the scene using JPEG2000) is measured in terms of their spectral similarity with regards to the five reference USGS spectral signatures in Fig. 7(b). The spectral similarity between an endmember extracted from the original scene, $\mathbf{e}_i$, and a USGS reference signature $\mathbf{s}_j$, is measured by the spectral angle (SA), a well known metric for hyperspectral data processing[7] which is defined as follows:

$$\mathrm{SA}(\mathbf{e}_i, \mathbf{s}_j) = \cos^{-1} \frac{\mathbf{e}_i \cdot \mathbf{s}_j}{\|\mathbf{e}_i\| \|\mathbf{s}_j\|}. \tag{2}$$

It should be noted that the SA is given by the arc cosine of the spectral angle formed by n-dimensional vectors. As a result, this metric is invariant in the multiplication of $\mathbf{e}_i$ and $\mathbf{s}_j$ by constants and, consequently,

Table 1. Spectral angle values (in degrees) between the pixels extracted by N-FINDR algorithm from the AVIRIS Cuprite scene (using different compression ratios) and the USGS reference signatures.

Compression	Alunite	Buddingtonite	Calcite	Kaolinite	Muscovite
No compression	$4{,}81^o$	$4{,}16^o$	$9{,}52^o$	$10{,}76^o$	$5{,}29^o$
2.0 bpppb	$7{,}44^o$	$7{,}03^o$	$12{,}31^o$	$13{,}45^o$	$8{,}16^o$
1.8 bpppb	$7{,}93^o$	$7{,}58^o$	$12{,}84^o$	$14{,}01^o$	$8{,}57^o$
1.6 bpppb	$8{,}40^o$	$8{,}55^o$	$13{,}36^o$	$14{,}59^o$	$9{,}12^o$
1.4 bpppb	$9{,}22^o$	$9{,}34^o$	$13{,}92^o$	$15{,}26^o$	$9{,}61^o$
1.2 bpppb	$9{,}95^o$	$10{,}06^o$	$14{,}44^o$	$15{,}80^o$	$10{,}23^o$
1.0 bpppb	$10{,}76^o$	$10{,}59^o$	$15{,}20^o$	$16{,}43^o$	$10{,}99^o$
0.8 bpppb	$11{,}17^o$	$11{,}06^o$	$15{,}79^o$	$17{,}12^o$	$11{,}88^o$
0.6 bpppb	$12{,}03^o$	$11{,}94^o$	$16{,}50^o$	$17{,}91^o$	$12{,}60^o$
0.4 bpppb	$13{,}12^o$	$13{,}02^o$	$17{,}61^o$	$19{,}04^o$	$13{,}72^o$
0.2 bpppb	$14{,}31^o$	$14{,}23^o$	$18{,}88^o$	$20{,}11^o$	$13{,}90^o$

Table 2. Real-time compression results obtained on the NVidia GeForce GTX 480 GPU for the AVIRIS Cuprite image using both lossless and lossy compression modes.

	Lossless compression			Lossy compression		
Input image size	Output image size	Ratio	bpppb	Output image size	Ratio	bpppb
43.92 MB	25.00 MB	0.57	9.11	21.97 MB	0.5	8.00

is invariant before unknown multiplicative scalings that may arise due to differences in illumination and angular orientation.[7] For illustrative purposes, Table 1 shows the SA values (in degrees) measured as the compression ratio was increased (the lower the SA, the better the obtained results, with results in the order of 15 degrees considered sufficiently similar in spectral terms[9]). As we can observe in our preliminary assessment, the quality of endmember extraction decreases for higher compression ratios as expected, with values of 1.2 bpppb (around 13:1 compression ratio) still providing acceptable results in terms of spectral similarity for all considered USGS minerals. Further experiments should be conducted in order to assess the impact of additional endmember extraction and abundance estimation algorithms for different compression ratios.

4.4 Analysis of parallel performance

Our GPU implementation was evaluated using the same AVIRIS Cuprite scene addressed in previous subsections. In our experiments, we used a CPU with Intel Core 2 Duo E8400 with 3,00GHz processor and 6GB RAM. The GPU platform used for evaluation purposes was the NVidia GeForce GTX 480, which features 480 processor cores. For the GPU implementation, we used the the compute unified device architecture (`CUDA`) as the development environment.

Our main goal in experiments was to assess the possibility of obtaining compression results in real-time, as the main interest of onboard compression using specialized hardware devices is to reduce the limitations imposed by the downlink connection when sending the data to a control station on Earth. As a result, real-time compression (as the data is collected by the sensor) is highly desirable. It should be noted that the cross-track line scan time in AVIRIS, a push-broom instrument,[2] is quite fast (8.3 milliseconds to collect 512 full pixel vectors). This introduces the need to compress the considered AVIRIS Cuprite scene (350×350 pixels and 188 spectral bands) in less than 1985 milliseconds to fully achieve real-time performance. In our experiments, we observed that the proposed GPU implementation of JPEG2000 was able to compress hyperspectral data sets in valid response time for both lossless and lossy compression. The time measured for lossless compression in the GPU was 1580 milliseconds, while the time measured for lossy compression (with 8 bpppb) in the GPU was 1490 milliseconds. The time difference between the lossless and lossy mode results from compression technique used in JPEG2000. During the lossy mode the quantization is used which considerably reduces the accuracy of the pixels and thus speedups the encoder. Whereas in lossless mode quantization is not used, so all information included in pixels are compressed by the encoder. Therefore duration of lossless compression is an upper bound for compression times with different ratios.

On the other hand, Table 2 shows the compression ratios and number of bpppb's obtained in real-time compression of the AVIRIS Cuprite data set. As it can be seen in Table 2, lossless compression provides approximately 2:1 compression ratio. Due to increasing spatial and spectral resolution of remotely sensed hyperspectral data sets, this ratio is not sufficient for onboard compression. On the other hand, in our experiments we observed that lossy compression (with 8 bpppb) is also able to provide 2:1 compression ratio in real-time. Obviously, further developments are required in order to increase the lossy compression ratios that can be achieved in real-time to fully benefit from lossy compression in order to optimize downlink data transmission.

5. CONCLUSIONS AND FUTURE LINES

In this paper we have developed a GPU implementation of JPEG2000 intended for real-time compression of remotely sensed hyperspectral images, with the ultimate goal of designing a specialized hardware implementation that can be used onboard hyperspectral imaging instruments. Although the power consumption rates of GPUs are still much higher than those provided by other hardware devices such as field programmable gate arrays (FPGAs), we believe in the future potential of GPU hardware accelerators for onboard hyperspectral image compression. Specifically, we have developed GPU implementations for the lossless and lossy compression modes of JPEG2000. For the lossy mode, we investigated the utility of the compressed hyperspectral images for different compression using a standard techniques for hyperspectral data exploitation such as linear spectral unmixing. In both cases we investigated the processing times that can be obtained using the GPU implementations, concluding that real-time performance can be obtained around 2:1 compression ratios in both cases. Further developments are required in order to increase the lossy compression ratios that can be achieved in real-time. In order to achieve this goal, we are planning on using the post-compression rate-distortion (PCRD) algorithm.[29] It allows to optimally truncate codestream depending on given target size. PCRD algorithm calculates distortion associated with truncation points. The process of calculating distortion is based on weights which are connected to the image characteristics. Therefore it is very important to choose suitable weights which reflect the image properties.

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. 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.
3. A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*, Taylor & Francis: Boca Raton, FL, 2007.
4. G. Motta, F. Rizzo, and J. A. Storer, *Hyperspectral data compression*, Springer-Verlag, New York, 2006.
5. Q. Du and C.-I. Chang, "Linear mixture analysis-based compression for hyperspectral image analysis," *IEEE Trans. Geosci. Rem. Sens.* **42**, pp. 875–891, 2004. [doi:10.1109/IGARSS.2000.861638].
6. D. S. Taubman and M. W. Marcellin, *JPEG2000: Image compression fundamentals, standard and practice*, Kluwer, Boston, 2002.
7. N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Processing Magazine* **19**(1), pp. 44–57, 2002.
8. 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.

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. A. Plaza, J. Plaza, A. Paz, and S. Sanchez, "Parallel hyperspectral image and signal processing," *IEEE Signal Processing Magazine* **28**(3), pp. 119–126, 2011.
11. S. Sanchez, A. Paz, G. Martin, and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units," *Concurrency and Computation: Practice and Experience* **23**(12), 2011.
12. C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**(3), 2011.
13. S.-C. Wei and B. Huang, "GPU acceleration of predictive partitioned vector quantization for ultraspectral sounder data compression," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**(3), 2011.
14. H. Yang, Q. Du, and G. Chen, "Unsupervised hyperspectral band selection using graphics processing units," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**(3), 2011.
15. J. A. Goodman, D. Kaeli, and D. Schaa, "Accelerating an imaging spectroscopy algorithm for submerged marine environments using graphics processing units," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**(3), 2011.
16. E. Christophe, J. Michel, and J. Inglada, "Remote sensing processing: From multicore to GPU," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**(3), 2011.
17. J. Mielikainen, B. Huang, and A. Huang, "GPU-accelerated multi-profile radiative transfer model for the infrared atmospheric sounding interferometer," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**(3), 2011.
18. C.-C. Chang, Y.-L. Chang, M.-Y. Huang, and B. Huang, "Accelerating regular LDPC code decoders on GPUs," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**(3), 2011.
19. International Organization for Standardization, "Information Technology - JPEG2000 Image Coding System - Part 1: Core Coding System." ISO/IEC 15444-1:2004, 2004.
20. International Organization for Standardization, "Information Technology - JPEG2000 Image Coding System - Part 2: Extensions." ISO/IEC 15444-2:2004, 2004.
21. International Organization for Standardization, "Information technology - JPEG2000 image coding system: Extensions for three-dimensional data." ISO/IEC 15444-10:2008, 2008.
22. D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Process.* **9**, pp. 1158–1170, 2000. [doi:10.1109/83.847830].
23. R. N. Clark, G. A. Swayze, K. E. Livo, R. F. Kokaly, S. J. Sutley, J. B. Dalton, R. R. McDougal, and C. A. Gent, "Imaging spectroscopy: Earth and planetary remote sensing with the usgs tetracorder and expert systems," *Journal of Geophysical Research* **108**, pp. 1–44, 2003.
24. G. Swayze, R. N. Clark, F. Kruse, S. Sutley, and A. Gallagher, "Ground-truthing AVIRIS mineral mapping at Cuprite, Nevada," *Proc. JPL Airborne Earth Sci. Workshop* , pp. 47–49, 1992.
25. 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.
26. D. A. Landgrebe, *Signal Theory Methods in Multispectral Remote Sensing*, John Wiley & Sons: New York, 2003.
27. J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis: An Introduction*, Springer, 2006.
28. M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral endmember determination in hyperspectral data," *Proceedings of SPIE* **3753**, pp. 266–277, 1999.
29. D. Taubman and M. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*, Springer, Berlin, 2002.