PARALLEL IMPLEMENTATION OF ENDMEMBER EXTRACTION ALGORITHMS USING NVIDIA GRAPHICAL PROCESSING UNITS

Antonio Plaza, Javier Plaza, Sergio Sánchez

Department of Technology of Computers and Communications
Escuela Politécnica de Cáceres, University of Extremadura
Avda. de la Universidad s/n, E-10071 Cáceres, Spain

ABSTRACT

Spectral mixture analysis is an important task for remotely sensed hyperspectral data interpretation. In spectral unmixing, both the determination of spectrally pure signatures (endmembers) and the unmixing process that interprets mixed pixels as combinations of endmembers are computationally expensive procedures. An exciting recent development in the field of commodity computing is the emergence of programmable graphics processing units (GPUs), which are now increasingly being used address the ever-growing computational requirements introduced by hyperspectral imaging applications. In this paper, we develop three new GPU-based implementations of endmember extraction algorithms: the pixel purity index (PPI), a kernel version of the PPI (KPPI), and the automatic morphological endmember extraction (AMEE) algorithm. We also provide a GPU-based implementation of the fully constrained linear spectral unmixing algorithm.

Index Terms— Spectral unmixing, hyperspectral imaging, commodity graphics hardware, endmember extraction

1. INTRODUCTION

Spectral mixture analysis has been an alluring exploitation goal since the earliest days of hyperspectral imaging [1]. No matter the spatial resolution, in natural environments the spectral signature for a nominal pixel is invariably a mixture of the signatures of the various materials found within the spatial extent of the ground instantaneous field view of the sensor. In hyperspectral imagery, the number of spectral bands usually exceeds the number of pure spectral components, called *endmembers* in hyperspectral analysis terminology, and the unmixing problem is cast in terms of an over-determined system of equations in which, given the correct set of endmembers allows determination of the actual endmember abundance fractions through a numerical inversion process. Since each observed spectral signal is the result of an actual mixing process, the driving abundances must obey two constraints. First, all abundances must be non-negative. Second, the sum of abundances for a given pixel must be unity.

Over the last decade, several algorithms have been developed for automatic extraction of endmembers from hyperspectral data sets [2]. A popular approach focused on exploiting the spectral information in the data has been the pixel purity index (PPI) [1]. Other algorithms, such as the automatic morphological endmember extraction (AMEE) [3], integrate the spatial and the spectral information in the data. While hyperspectral imaging algorithms such as PPI or AMEE hold great promise for Earth science image analysis, they also introduce new processing challenges, in particular, for very high-dimensional data sets. From a computational perspective, these algorithms exhibit inherent parallelism at multiple levels: across pixel vectors (coarse grained pixel-level parallelism), across spectral information (fine grained spectral-level parallelism), and even across tasks (task-level parallelism). As a result, they map nicely to massively parallel systems such as clusters [4]. Unfortunately, these systems are expensive and difficult to adapt to onboard data processing scenarios, in which low-weight and low-power integrated components are mandatory to reduce mission payload.

An exciting recent development in the field of commodity computing is the emergence of programmable graphics processing units (GPUs) [5, 6], which bear many features of vector processors. The speed of graphics hardware doubles approximately every six months, which is much faster than the improving rate of the CPU. The ever-growing computational requirements introduced by hyperspectral imaging applications can benefit from this hardware and take advantage of the compact size and relatively low cost of these units, which make them appealing for onboard data processing at much lower costs than those introduced by other hardware devices. In this paper, we develop three new GPU-based implementations of endmember extraction algorithms: the PPI, a kernel version of the PPI, and the spatial-spectral AMEE algorithm. We also provide a GPU-based implementation of the fully constrained linear spectral unmixing algorithm. The algorithms have been implemented on an NVidia GeForce 8800 GTX GPU[1], using NVidia's CUDA[2] programming language.

[1]http://www.nvidia.com/page/geforce_8800.html
[2]http://www.nvidia.com/object/cuda_home.html

2. ALGORITHMS

The algorithms considered in this work are the PPI algorithm, a kernel version of the PPI (called KPPI), and the AMEE algorithm. A brief description of their behavior follows:

- The PPI proceeds by generating a large number of random, N-dimensional unit vectors called "skewers" through the dataset [1]. Every data point is projected onto each skewer, and the data points that correspond to extrema in the direction of a skewer are tallied. The pixels with the highest tallies are considered the final endmembers.
- On the other hand, the AMEE begins by searching spatial neighborhoods around each pixel in the image for the most spectrally pure and mostly highly mixed pixel. This task is performed by using extended mathematical morphology operators of dilation and erosion, respectively [3]. Each spectrally pure pixel is assigned an "eccentricity" value, which is calculated as the spectral angle distance between the most spectrally pure and mostly highly mixed pixel for the given spatial neighborhood. This process is repeated iteratively for larger spatial neighborhoods up to a maximum size that is predetermined. The final endmember set is obtained by applying a threshold to the "eccentricity" image.
- Finally, the KPPI can be seen as a hybrid between the PPI and the AMEE. It begins by searching spatial neighborhoods around each pixel in the image, but instead of using morphological operations of erosion and dilation, it looks for the most spectrally pure pixels in the local neighborhood by applying a variant of the original PPI in each local neighborhood.

3. IMPLEMENTATIONS

3.1. Programming general purpose GPUs

GPUs can be abstracted in terms of a *stream* model, under which all data sets are represented as streams (i.e., ordered data sets). Algorithms are constructed by chaining so-called *kernels*, which operate on entire streams, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. Modern GPU architectures adopt this model and implement a generalization of the traditional rendering pipeline, which consists of two main stages:

1. *Vertex processing*. The input to this stage is a stream of vertices from a 3-D polygonal mesh. Vertex processors transform the 3-D coordinates of each vertex of the mesh into a 2-D screen position, and apply lighting to determine their colors (this stage is fully programmable).

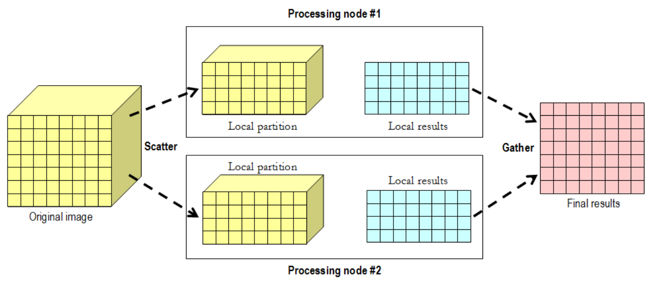

Fig. 1. Spatial-domain decomposition.

2. *Fragment processing*. In this stage, the transformed vertices are first grouped into rendering primitives, such as triangles, and scan-converted into a stream of pixel fragments. These fragments are discrete portions of the triangle surface that corresponds to the pixels of the rendered image. Apart from identifying constituent fragments, this stage also interpolates attributes stored at the vertices, such as texture coordinates, and stores the interpolated values at each fragment. Arithmetical operations and texture lookups are then performed by fragment processors to determine the ultimate color for the fragment. For this purpose, texture memories can be indexed with different texture coordinates, and texture values can be retrieved from multiple textures.

It should be noted that fragment processors currently support instructions that operate on vectors of four RGBA components (Red/Green/Blue/Alpha channels) and include dedicated texture units that operate with a deeply pipelined texture cache. As a result, an essential requirement for mapping non-graphics algorithms onto GPUs is that the data structure can be arranged according to a stream-flow model, in which kernels are expressed as *fragment* programs and data streams are expressed as *textures*. Using C-like, high-level languages such as the compute unified device architecture (CUDA), programmers can write fragment programs to implement general-purpose operations.

3.2. CUDA-based implementations

The algorithms were implemented using NVidia's CUDA, a collection of C extensions and a runtime library. CUDAs functionality primarily allows a developer to write C functions to be executed on the GPU. CUDA also includes memory management and execution configuration, so that a developer can control the number of GPU processors and threads that are to be invoked during a function's execution.

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 [4] made up of entire pixel vectors (see Fig. 1),

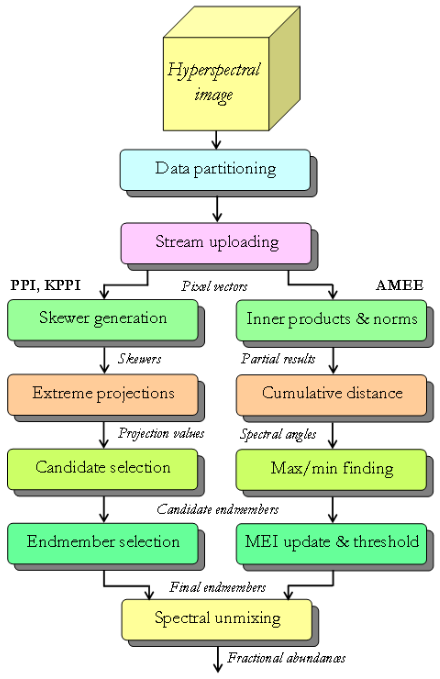

Fig. 2. Flowchart of the stream-based GPU implementations.

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 parition is further divided into 4-band tiles (called spatial-domain tiles), which are arranged in different areas of a 2-D texture. Such partitioning allows us to map four consecutive spectral bands onto the RGBA color channels of a texture element.

Apart from the tiles, we also allocate additional memory to hold other information, such as the skewers which are generated at the host (CPU) and then transmitted to the GPU, and also intermediate results such as dot products, norms, and pointwise distances. Figure 2 shows a flowchart describing the kernels involved in our GPU-based implementations. There are two kernels which are common to the three endmember extraction algorithms: the *data partitioning* stage performs the spatial-domain decomposition of the original hyperspectral image. In the *stream uploading* stage, the spatial-domain partitions are uploaded as a set of tiles onto the GPU memory. The kernels which are specific to the PPI and KPPI implementations can be described as follows:

- *Skewer generation*. This kernel provides the skewers, using NVidia's parallel implementation of the Mersenne twister pseudo-random number generator on the GPU.

- *Extreme projections*. The tiles are input streams to this stage, which obtains all the dot products necessary to compute the required projections. The implementation of this stage is based on a multi-pass kernel that implements an element-wise multiply and add operation, thus producing four partial inner products stored in the RGBA channels of a texture element.

- *Candidate selection*. This kernel uses as inputs the projection values generated in the previous stage, and produces a stream for each pixel vector, containing the relative coordinates of the pixels with maximum and minimum distance after the projection onto each skewer. A complementary kernel is then used to find pixels which have been selected repeatedly during the process.

- *Endmember selection*. For each endmember candidate, this kernel computes the cumulative SAD with all the other candidates. It is based on a single-pass kernel that computes the SAD between two pixel vectors using the dot products and norms produced by the previous stage. A complementary kernel is then used to discard those candidates with cumulative SAD scores below a threshold angle.

It should be noted that both PPI and KPPI use the same kernels. The only difference between these two algorithms is in the way pixels are fed in the *stream uploading* stage. In the case of PPI, the pixels are fed without any spatial arrangement, while in the KPPI the pixels are fed in local neighborhoods according to the kernel size parameter. This strategy is also used in the AMEE implementation, which is based on the following specific kernels:

- *Inner products & norms*. The tiles are input streams to this kernel, which obtains all the inner products and norms necessary to compute the point-wise spectral angle distances involved in the calculations.

- *Cumulative distance*. For each pixel vector, this kernel accumulates the spectral angle with all the neighboring pixels. It is based on a single-pass kernel that computes the spectral angle distance between two pixel vectors using the inner products and norms produced by the previous kernel. Finally, the kernel calculates, for each pixel vector, the cumulative spectral angle between the pixel and all its neighbors.

- *Max/min finding*. Morphological erosion and dilation are finalized at this stage through a kernel that applies minimum and maximum reductions. This kernel uses as inputs the cumulative values generated in the previous stage and produces a stream containing (for each pixel) the relative coordinates of the neighboring pixels with maximum and minimum cumulative distance.

- *Eccentricity update*. This kernel updates the morphological eccentricity scores using the maximum/minimum and point-wise distance streams. A complementary kernel applies a threshold to select a set of final endmembers at the end of the process.

Table 1. Processing time (seconds) and speedups for the dual-core CPU and GPU implementations.

Algorithm	Processing time (CPU)	Processing time (GPU)	Speedup
PPI	493.78	18.93	26.08
KPPI	177.30	19.29	9.19
AMEE	1345.18	52.60	25.57

- *Spectral unmixing*. Finally, this kernel (common to all endmember extraction algorithms) uses as inputs the final endmembers selected in the previous stage and produces the endmember fractional abundances for each pixel by means of an inversion process.

4. EXPERIMENTAL RESULTS

4.1. GPU platform

Our experiments were performed on a 2006-model HP xw8400 workstation based on dual Quad-Core Intel Xeon processor E5345 running at 2.33 GHz with 1.333 MHz bus speed and 3 GB RAM. The computer was equipped with an NVidia GeForce 8800 GTX with 16 multiprocessors, each composed of 8 processors operating at 1350 Mhz. Each multiprocessor has 8192 registers, a 16 KB parallel data cache of fast shared memory, and access to 768 MB of global memory.

4.2. Hyperspectral image experiments

The hyperspectral data set used in our experiments is the well-known AVIRIS Cuprite scene[3]. A comparison in terms of spectral angle distances between the endmembers extracted by the PPI, KPPI and AMEE versus USGS reference signatures (not displayed here for space considerations) revealed that the three considered algorithms were able to extract good candidate endmembers from the input scene. In order to study the scalability of our CPU and GPU-based implementations, we tested them on a subset (614×512 pixels and 224, for a total size of 140 MB). Table 1 shows the execution times and speedups measured for the GPU-based implementations compared to execution in the dual-core CPU of the system in which the GPU is integrated. The speedups achieved by the GPU implementation of the PPI and AMEE algorithms over their respective CPU implementations remained close to 26. It should be noted that the speedup achieved for the GPU implementation of AMEE were independent of the kernel size, even for very large kernels (the results displayed in Table 1 correspond to a kernel size of 5×5 pixels). The GPU implementation of the KPPI was not as effective, probably due to the procedure adopted for incorporating the spatial information into the PPI algorithm. This aspect will be subject to improvements in our future research work.

[3]Available online: http://aviris.jpl.nasa.gov/html/aviris.freedata.html

5. CONCLUSIONS

The emergence of low-weight and low-power specialized hardware devices such as graphics processing units (GPUs), whose evolution in terms of performance and cost is mainly due to the advent of video-game industry, is now progressively bridging the gap towards real-time analysis of high dimensional data in many application domains, including remote sensing. This kind of specialized, on-board processing devices are essential to reduce mission payload and obtain analysis results quickly enough for practical use. In this work, we have presented our experience in computationally efficient implementation of algorithms for endmember extraction and spectral unmixing of remotely sensed hyperspectral data using commodity graphics hardwere. Specifically, three algorithms: PPI, KPPI and AMEE have been implemented on an NVidia GeForce 8800 GTX, obtaining speedups on the order of 26 for the PPI and AMEE, and 9 for the KPPI.

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 also gratefully acknowledged.

7. REFERENCES

[1] J. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of AVIRIS data," in *Summaries of the JPL Airborne Earth Science Workshop*. JPL Publication, 1995, pp. 23–26.

[2] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, pp. 650–663, 2004.

[3] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, pp. 2025–2041, 2002.

[4] A. Plaza, J. Plaza, and D. Valencia, "Impact of platform heterogeneity on the design of parallel algorithms for morphological processing of high-dimensional image data," *Journal of Supercomputing*, vol. 40, pp. 81–107, 2007.

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

[6] Y. Tarabalka, T. V. Haavardsholm, I. Kasen, and T. Skauli, "Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and gpu processing," *Journal of Real-Time Image Processing*, vol. 4, pp. 1–14, 2009.