Parallel Morphological Endmember Extraction Using Commodity Graphics Hardware

Javier Setoain, Manuel Prieto, *Member, IEEE*, Christian Tenllado, Antonio Plaza, *Senior Member, IEEE*, and Francisco Tirado, *Senior Member, IEEE*

***Abstract*—Spatial/spectral algorithms have been shown in previous work to be a promising approach to the problem of extracting image endmembers from remotely sensed hyperspectral data. Such algorithms map nicely on high-performance systems such as massively parallel clusters and networks of computers. Unfortunately, these systems are generally expensive and difficult to adapt to onboard data processing scenarios, in which low-weight and low-power integrated components are highly desirable to reduce mission payload. An exciting new development in this context is the emergence of graphics processing units (GPUs), which can now satisfy extremely high computational requirements at low cost. In this letter, we propose a GPU-based implementation of the automated morphological endmember extraction algorithm, which is used in this letter as a representative case study of joint spatial/spectral techniques for hyperspectral image processing. The proposed implementation is quantitatively assessed in terms of both endmember extraction accuracy and parallel efficiency, using two generations of commercial GPUs from NVidia. Combined, these parts offer a thoughtful perspective on the potential and emerging challenges of implementing hyperspectral imaging algorithms on commodity graphics hardware.**

***Index Terms*—Commodity graphics hardware, endmember extraction, spatial/spectral analysis.**

I. Introduction

ENDMEMBER extraction is a fundamental and crucial task in hyperspectral data exploitation. Over the last decade, several algorithms have been developed for automatic extraction of image endmembers from hyperspectral data sets, including the pixel purity index [1], N-FINDR [2], vertex component analysis [3], or iterative error analysis (IEA) [4]. The previous techniques treat the hyperspectral data not as an image but as an unordered list of spectral measurements, where the spatial coordinates can be randomly permuted without affecting the endmember searching process. As an alternative to purely spectral approaches, the automated morphological endmember extraction (AMEE) algorithm [5] was developed with the purpose of integrating both the spatial and spectral information in the search for endmembers using mathematical morphology concepts [6].

While integrated spatial/spectral developments hold great promise for Earth science image analysis, they also introduce new processing challenges, particularly for very high-dimensional data sets. From a computational standpoint, such algorithms exhibit regular data access patterns and 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 made up of commodity CPUs (e.g., Beowulf clusters). Unfortunately, these systems are generally expensive and difficult to adapt to onboard remote sensing data processing scenarios.

An exciting new development in the field of commodity computing is the emergence of programmable graphics processing units (GPUs). Driven by the increasing demands of the videogame industry, GPUs have evolved from expensive application-specific units into highly parallel and programmable systems. Although the GPU architecture is not necessarily suitable for all kinds of parallel computations, the range of candidate applications is growing far beyond the domain of graphic rendering [7], [8]. Specifically, the ever-growing computational requirements introduced by state-of-the-art hyperspectral imaging algorithms can fully 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 such as field-programmable gate arrays (FPGAs).

In this letter, we describe a GPU-based implementation of the AMEE algorithm, which is used as a representative case study of spatial/spectral developments for hyperspectral analysis. This letter is organized as follows. Section II illustrates how GPUs can be used for nongraphics computations. Section III describes the proposed implementation. Section IV evaluates the proposed GPU-based implementation from the viewpoint of both endmember extraction accuracy (compared to other standard approaches) and parallel performance. Section V concludes with some remarks.

II. GPU Computing Model

Modern GPUs such as the latest NVidia GeForce or ATI Radeon cards implement a generalization of the traditional rendering pipeline for 3-D computer graphics [7]. The inputs to this pipeline are vertices from a 3-D polygonal mesh and a "virtual camera" viewpoint, and the output is a 2-D array of pixels to be displayed on the screen. As shown by Fig. 1, the pipeline consists of several stages, but the bulk of the work is performed by three of them: vertex processing, rasterization, and fragment processing.

The vertex processing stage transforms the 3-D coordinates of each vertex of the input mesh into a 2-D screen position and

Manuscript received November 16, 2006; revised February 6, 2007. This work was supported by the Spanish government through the research contracts CICYT-TIN 2005/5619 and ESP2005-07724-C05-02.

J. Setoain, M. Prieto, C. Tenllado, and F. Tirado are with the ArTeCS Group, Department of Computer Architecture, Complutense University, E-28040 Madrid, Spain.

A. Plaza is with the Department of Computer Science, University of Extremadura, E-10071 Cáceres, Spain.

Digital Object Identifier 10.1109/LGRS.2007.897398

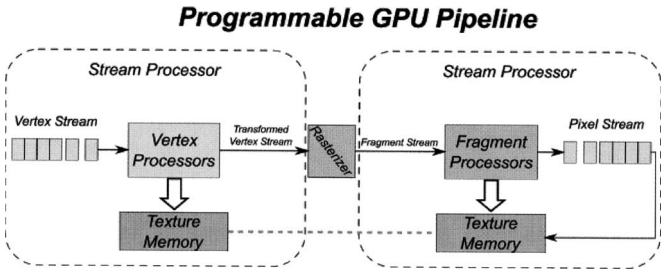

Fig. 1. Overview of the programmable graphics pipeline.

applies lighting to determine their colors. Once transformed, vertices are grouped into rendering primitives such as triangles and scan converted by the rasterizer into a stream of pixel fragments. These fragments are discrete portions of the triangle surface that correspond to the pixels of the rendered image. The fragment processing stage is generally used to modify the color of each fragment with texture mapping or other mathematical operations. The output from the fragment processing stage is finally combined with the existing data stored at the associated 2-D locations in the frame buffer to produce the ultimate colors.

It is also worth noting that parallelism is exploited at different levels within this graphics pipeline [7]. The actual hardware of a modern GPU has hundreds of stages to exploit functional parallelism and increase the throughput. Furthermore, GPUs also incorporate replicated stages to take advantage of the inherent data parallelism of the rendering process. For instance, the vertex and fragment processing stages include several replicated units known as vertex and fragment processors, respectively. On the other hand, the vertex and fragment processors exploit multithreading to hide memory latencies. The overall idea is that for every vertex (fragment) in the incoming stream, the GPU launches a kernel thread executing the vertex (fragment) program. Finally, independent operations or instructions in a vertex (fragment) program may be executed in parallel as well. In particular, current fragment processors explicitly support short-vector instructions that operate on four-element vectors [red/green/blue/alpha (RGBA) channels] in single-instruction multiple-data fashion.

Until only a few years ago, commercial GPUs implemented a fixed rendering pipeline. However, the vertex and fragment processing stages are now entirely programmable, a fact that allows for implementation of nongraphics applications such as remote sensing problems, using C-like high-level languages [8]. For nongraphics applications, the GPU is usually abstracted as a general stream processor that performs computations through the use of streams and kernels. A stream can be defined as an ordered collection of elements to be processed in similar fashion, whereas a kernel is defined as the function used to transform streams. The outcome of a kernel should not depend on the order in which output elements are produced (i.e., it should be possible to process them concurrently), which forces the programmer to explicitly expose data parallelism to the hardware. Therefore, the mapping of a general purpose application onto a GPU involves algorithm transformations to follow a stream-programming model. For matrix-based computations such as those involved in standard image processing operations, it is usually more convenient to express these kernels as fragment programs and abstract the textures as streams [9].

Many processing tasks fit this model well, and although hyperspectral image processing algorithms have not received much attention yet, many researchers have successfully ported a large number of scientific applications onto GPUs [8]. Nevertheless, harnessing the power of a GPU is not straightforward and requires a concerted effort by experts in both the target application (hyperspectral imaging in our case), parallel computing, and 3-D graphics. As a result, the design and development of cost-effective hyperspectral imaging algorithms on GPU platforms represents both a challenge and a highly innovative contribution.

III. GPU-Based Endmember Extraction

A. Morphological Endmember Extraction

Let us denote by $\boldsymbol{f}$ a hyperspectral data set defined on an N-dimensional (N-D) space, where N is the number of channels or spectral bands. The main idea of the AMEE algorithm is to impose an ordering relation in terms of spectral purity in the set of pixel vectors lying within a spatial search window or structuring element around each image pixel vector [6]. To do so, we first define a cumulative distance between one particular pixel $\boldsymbol{f}(x,y)$, i.e., an N-D vector at discrete spatial coordinates (x,y), and all the pixel vectors in the spatial neighborhood given by B (B-neighborhood) as follows [5]:

$$D_B\left(\boldsymbol{f}(x,y)\right) = \sum_{(i,j)\in Z^2(B)} \text{Dist}\left(\boldsymbol{f}(x,y), \boldsymbol{f}(i,j)\right) \quad (1)$$

where (i,j) are the spatial coordinates in the B-neighborhood discrete domain, represented by $Z^2(B)$, and Dist is a pointwise distance measure between two N-D vectors. The choice of Dist is a key topic in the resulting ordering relation. The AMEE algorithm makes use of the Spectral Angle Mapper (SAM), a standard measure in hyperspectral analysis [10]. For illustrative purposes, let us assume that $\boldsymbol{s}_i = (s_{i1}, s_{i2}, \ldots, s_{iN})^T$ and $\boldsymbol{s}_j = (s_{j1}, s_{j2}, \ldots, s_{jN})^T$ are two N-D signatures. Here, the term "spectral signature" does not necessarily imply "pixel vector," and hence, spatial coordinates are omitted from the two previous signatures, although the following argumentation would be the same if pixel vectors were considered. The SAM between $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ is given by

$$\text{SAM}(\boldsymbol{s}_i, \boldsymbol{s}_j) = \cos^{-1}\left(\boldsymbol{s}_i \cdot \boldsymbol{s}_j / \|\boldsymbol{s}_i\| \cdot \|\boldsymbol{s}_j\|\right). \quad (2)$$

It should be noted that SAM is invariant in the multiplication of input vectors by constants and, consequently, is invariant to unknown multiplicative scalings that may arise due to differences in illumination and sensor observation angle.

With the previous definitions in mind, we provide in the following a step-by-step description of the AMEE algorithm which corresponds to the implementation used in [11]. The inputs to the algorithm are hyperspectral data cube $\boldsymbol{f}$, a structuring element B with size of $t \times t$ pixels, a maximum number of algorithm iterations $I_{\max}$, and a maximum number of endmembers to be extracted p. The output is a set of endmembers $\{\boldsymbol{e}_i\}_{i=1}^{q}$, with $q \leq p$.

1) Set $i = 1$, and initialize a morphological eccentricity index score $\text{MEI}(x,y) = 0$ for each pixel.

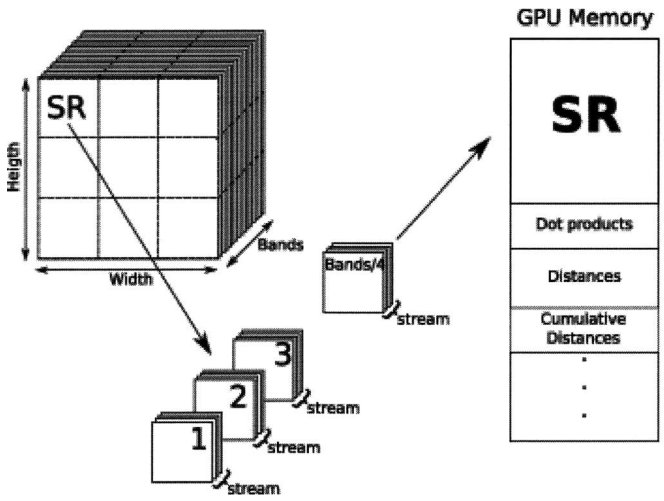

Fig. 2. Mapping of a hyperspectral image onto the GPU memory.

2) Move B through all the pixels of $\boldsymbol{f}$, defining a local spatial search area around each pixel $\boldsymbol{f}(x, y)$, and calculate the maximum and minimum pixel vectors at each B-neighborhood using morphological erosion and dilation [6], respectively, as follows:

$$(\boldsymbol{f} \ominus B)(x, y) = \text{argmin}_{(i,j)\in Z^2(B)} \{D_B [\boldsymbol{f}(x + i, y + j)]\} \quad (3)$$

$$(\boldsymbol{f} \oplus B)(x, y) = \text{argmax}_{(i,j)\in Z^2(B)} \{D_B [\boldsymbol{f}(x + i, y + j)]\} \quad (4)$$

3) Update the MEI at each spatial location (x, y) using $\text{MEI}(x, y) = \text{Dist}[(\boldsymbol{f} \ominus B)(x, y), (\boldsymbol{f} \oplus B)(x, y)]$.
4) Set $i = i + 1$. If $i = I_{\max}$, then go to step 5). Otherwise, set $\boldsymbol{f} = \boldsymbol{f} \oplus B$ and go to step 2).
5) Select the set of p pixel vectors with higher associated MEI scores (called endmember candidates) and form a unique spectral set of $\{\boldsymbol{e}_i\}_{i=1}^{q}$ pixels, with $q \leq p$, by calculating the Dist for all pixel vector pairs.

B. GPU Implementation

We have focused on mapping the first four steps of the AMEE algorithm (i.e., finding the MEI score map) onto a GPU since these steps account for most of the execution time involved in the endmember extraction process and exhibit enough data parallelism.

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 partitions made up of entire pixel vectors [called spatial regions (SRs)], i.e., each SR incorporates all the spectral information on a localized SR and is composed of spatially adjacent pixel vectors.[1] As shown by Fig. 2, SRs are further divided into four-band tiles (called SR 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 (memory) element. Apart from the SR tiles, we also allocate additional memory to hold intermediate information such as inner products, norms, pointwise distances, and cumulative distances.

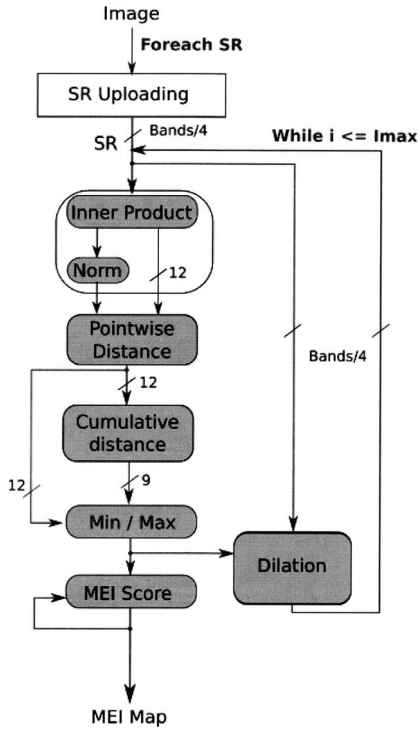

Fig. 3. Flowchart of the proposed stream-based GPU implementation.

Fig. 3 shows a flowchart describing our GPU-based implementation. The stream uploading stage performs the data partitioning and mapping operations described previously, i.e., dividing the image into SRs and uploading them as a set of SR tiles onto the GPU memory. The remaining stages perform the actual computation and comprise the following kernels.

1) Inner products and norms. The SR tiles are input streams to this stage, which obtains all the inner products and norms necessary to compute the required pointwise distances. Keeping in mind that the size of the structuring element is $t \times t$ pixels; it will be necessary to compute $t^2(t^2 + 1)/2$ per pixel. However, taking advantage of the redundancy between adjacent structuring elements, it is possible to reduce this figure to $\lceil (2t - 1)^2/2 \rceil$. As shown by Fig. 4, for $t = 3$, we only need to compute $12 + 1$ inner products per pixel: one product of the vector with itself (to find the norm) and twelve with the pixel vectors within its region of influence (RI).[2] Since streams are actually SR tiles, the implementation of this stage is based on two kernels, denoted as multiply and add (MAD) and 4-to-1 in

[1]There is partial overlapping among spatially adjacent SRs to handle boundary problems. The overlapping width depends on the size of the structuring element and the number of iterations of the morphological algorithm [11]. For a $t \times t$ structuring element and i algorithm iterations, the overlapping width is $\lfloor t/2 \rfloor + (i - 1)$.

[2]The RI of a pixel includes four-connected neighbors to the pixel—the southwest (SW), south (S), southeast (SE), and east (E) neighbors—as well as their respective W, SW, S, SE, and E neighbors within the structuring element. It is worth noting that other alternative definitions for the RI are possible by adopting different connectivity criteria in the selection of neighbors, as far as the chosen RI contains a minimum set of neighbors that cover all the instances.

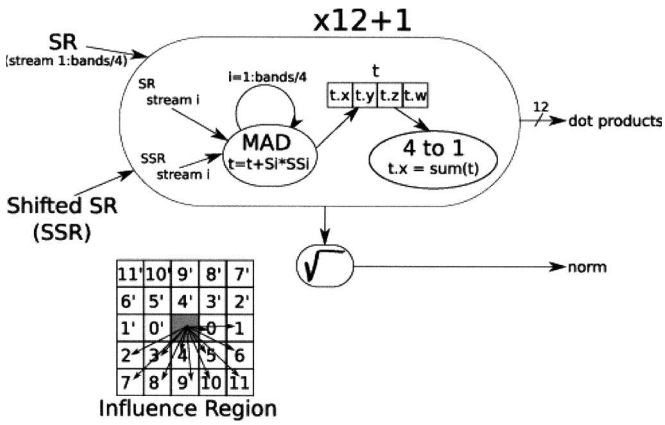

Fig. 4. Kernels involved in the computation of the inner products/norms and definition of an RI for a given pixel defined by a structuring element with $t = 3$.

Fig. 4. The former is a multipass kernel that implements an elementwise MAD operation, thus producing four partial inner products stored in the RGBA channels of a texture element. The latter is a single-pass kernel that computes the final inner products performing the sum reduction of this four-element vector. A third single-pass kernel produces the norm of every pixel vector.

2) Pointwise distance. For each pixel vector, this stage computes the SAM with all the neighbor pixels within its RI. It is based on a single-pass kernel that computes the SAM between two pixel vectors using the inner products and norms produced by the previous stage.
3) Cumulative distance. For each pixel vector, this stage produces t^2 cumulative distance streams for each of the t^2 neighbors defined by the structuring element. It is based on a single-pass kernel that accumulates up to eight pointwise distances.[3]
4) Maximum/minimum finding. Erosion and dilation are finalized at this stage through a kernel that applies minimum and maximum reductions. This kernel uses as inputs the cumulative SAMs generated in the previous stage and produces a stream containing (for each pixel vector) the relative coordinates of the neighboring pixels with maximum and minimum cumulative distance.
5) Dilation. If $i < I_{\max}$, this stage propagates the purest pixels in the current iteration to produce the SR tiles for the next iteration.
6) MEI score update. Finally, this stage updates the MEI scores using the maximum/minimum and pointwise distance streams.

IV. Experimental Results

A. Computing Platforms

The proposed endmember extraction algorithm has been implemented on a state-of-the-art GPU, as well as on an older (three-year old) system in order to account for potential improvements that might be achievable on future generations of GPUs. The generations considered correspond to the NV30 and G70 families (see Table I), and the programs were coded using Cg [8]. For perspective, we have also reported performance results on contemporary Intel CPUs (see Table II). The CPU implementations were developed using the Intel C/C++ compiler and optimized via compilation flags to exploit data locality and avoid redundant computations of common pointwise distances between adjacent structuring elements.

[3]The number of texture indirections that can be performed in a fragment program by hardware is limited to eight in our target GPUs.

TABLE I
Experimental GPU Features

	FX5950 Ultra	7800 GTX
Year	2003	2005
Architecture	NV38	G70
Bus	AGPx8	PCI Express
Video Memory	256MB	256MB
Core Clock	475 MHz	430 MHz
Memory Clock	950 MHz	1.2 GHz GDDR3
Memory Interface	256-bit	256-bit
Memory bandwidth	30.4 GB/s	38.4 GB/s
#Pixel shader processors	4	24
Texture fill rate	3800 MTexels/s	10320 MTexels/s

TABLE II
Experimental CPU Features

	Pentium 4 (Northwood C)	Prescott (6x2)
Year	2003	2005
FSB	800 MHz, 6.4 GB/s	800 MHz, 6.4 GB/s
ICache L1	12KB	12KB
DCache L1	8KB	16KB
L2 Cache	512KB	2M
Memory	1GB	2 GB
Clock	2.8 GHz	3.4 GHz

B. Hyperspectral Data

The hyperspectral data set used in our experiments is the well-known AVIRIS Cuprite scene, available online (in reflectance units) from http://aviris.jpl.nasa.gov/html/aviris.freedata.html. The scene comprises 1939 $\times$ 677 pixels with spatial resolution of 20 m and 204 narrow spectral bands between 0.4 and 2.5 μm, and nominal spectral resolution of 10 nm (20 bands has been removed from the original scene prior to analysis due to low SNR in those bands). The reflectance spectra of ten U.S. Geological Survey (USGS) ground mineral spectra: alunite, buddingtonite, calcite, chlorite, kaolinite, jarosite, montmorillonite, muscovite, nontronite, and pyrophilite (all available from http://speclab.cr.usgs.gov) were used as ground-truth spectra, to illustrate endmember extraction accuracy. Finally, in order to study the scalability of our CPU- and GPU-based implementations, we tested them on different image sizes, where the largest one corresponds to the full 1939 $\times$ 677 pixel scene, whereas the others correspond to cropped portions of the same image.

C. Performance Evaluation

Before empirically investigating the performance of the proposed GPU-based implementation, we first briefly discuss endmember extraction accuracy of the proposed morphological method in comparison with other available approaches. Table III tabulates the SAM spectral similarity scores obtained after comparing USGS library spectra with the corresponding endmembers extracted by standard endmember extraction

TABLE III
SAM-BASED SPECTRAL SIMILARITY SCORES AMONG USGS MINERAL SPECTRA AND ENDMEMBERS PRODUCED BY DIFFERENT ALGORITHMS

	PPI	N-FINDR	VCA	IEA	AMEE ($I_{max} = 1$)	AMEE ($I_{max} = 3$)	AMEE ($I_{max} = 5$)
Alunite	0.084	0.081	0.084	0.084	0.084	0.081	0.079
Buddingtonite	0.106	0.084	0.112	0.094	0.112	0.086	0.081
Calcite	0.105	0.105	0.093	0.110	0.106	0.102	0.093
Chlorite	0.125	0.136	0.096	0.096	0.122	0.110	0.096
Kaolinite	0.136	0.152	0.134	0.134	0.136	0.136	0.106
Jarosite	0.112	0.102	0.112	0.108	0.115	0.103	0.094
Montmorillonite	0.106	0.089	0.120	0.096	0.108	0.105	0.101
Muscovite	0.108	0.094	0.105	0.106	0.109	0.099	0.092
Nontronite	0.102	0.099	0.099	0.099	0.101	0.095	0.090
Pyrophilite	0.094	0.090	0.112	0.090	0.098	0.092	0.079

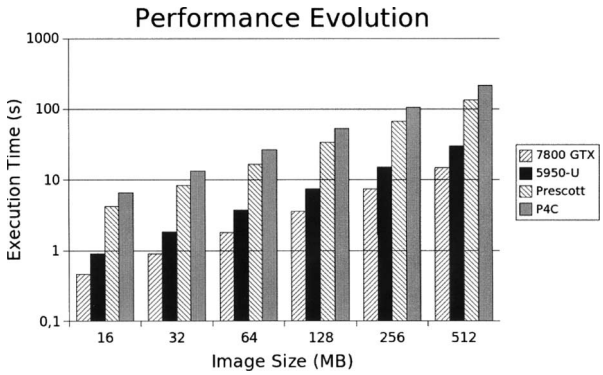

Fig. 5. Performance of the CPU- and GPU-based AMEE implementations for different image sizes ($I_{\max} = 5$).

algorithms and the proposed AMEE algorithm, using different numbers of iterations, ranging from $I_{\max} = 1$ to $I_{\max} = 5$ and a constant structuring element with $t = 3$ (the smaller the scores across the ten minerals considered in Table III, the better the results). The number of endmembers to be extracted in all cases was set to 16 (despite the availability of only ten ground-truth references) after calculating the intrinsic dimensionality of the data [10], which revealed the presence of additional (unknown) endmembers and alterations of available endmembers. Overall, the tables show that AMEE is able to outperform the other methods which rely on using the spectral information alone, particularly as the number of algorithm iterations was increased. In fact, montmorillonite was the only endmember mineral for which N-FINDR and IEA provided better similarity results than AMEE with $I_{\max} = 5$.

On the other hand, Fig. 5 shows the execution times of our CPU- and GPU-based AMEE implementations for different image sizes. First, it is worth noting that the GPU version is able to process the full 1939×677 pixel data cube (about 512 MB in size) in only 14 s (for $I_{\max} = 5$), in spite of the overheads involved in data transfer between the main memory and the GPU. The speedups achieved by the GPU implementation over their CPU counterparts are outstanding: they are in the order of 10 (for a single GPU). Although these factors are already remarkable, we should also highlight that multi-GPU systems or even clusters of GPUs may significantly increase the reported figures. Results in Fig. 5 further demonstrate that the complexity of the implementation scales linearly with the problem size, i.e., doubling the image size simply doubles the execution time. We also remark the relative evolution of GPUs when compared to that of CPUs. The improvement caused by the evolution of Intel CPUs was below 60%, as opposed to the 200% improvement observed for GPU counterparts. This comes at no surprise, since the latest generation has multiplied by six the number of fragment processors as well as increased the onboard memory bandwidth [7].

V. CONCLUSION AND FUTURE LINES OF INVESTIGATION

In this letter, we have explored the viability of using GPUs for efficiently implementing spatial/spectral endmember extraction algorithms. This approach represents a cost-effective alternative to other high-performance systems, such as Beowulf-type clusters, which are expensive and difficult to adapt to onboard processing scenarios. The outstanding speedups reported in experiments, together with the low cost and impressive evolution of GPUs, anticipate a significant impact of these hardware devices in the remote sensing community. In future developments, we will explore additional partitioning strategies to balance the workload between the CPU and the GPU. Further research will also include experiments with multi-GPU systems and clusters of GPUs.

REFERENCES

[1] J. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of aviris data," in *Proc. Summaries JPL Airborne Earth Sci. Workshop*, 1995, pp. 23–26.
[2] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral endmember determination in hyperspectral data," in *Proc. SPIE Imaging Spectrometry V*, 1999, vol. 3753, pp. 266–277.
[3] J. M. P. Nascimento and J. M. B. Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 4, pp. 898–910, Apr. 2005.
[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] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 9, pp. 2025–2041, Sep. 2002.
[6] P. Soille, *Morphological Image Analysis: Principles and Applications*, 2nd ed. Berlin, Germany: Springer-Verlag, 2003.
[7] J. Montrym and H. Moreton, "The GeForce 68000," *IEEE Micro*, vol. 25, no. 2, pp. 41–51, Mar./Apr. 2005.
[8] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A survey of general purpose computation on graphics hardware," in *Proc. Eurographics: State of the Art Reports*, 2005, pp. 21–51.
[9] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 777–786, 2004.
[10] C.-I Chang, *Hyperspectral imaging: Techniques for Spectral Detection and Classification*. New York: Kluwer, 2003.
[11] A. Plaza, D. Valencia, and J. Plaza, "Parallel implementation of endmember extraction algorithms from hyperspectral imagery," *IEEE Geosci. Remote Sens. Lett.*, vol. 3, no. 3, pp. 334–338, Jul. 2006.