# Parallel Hyperspectral Image Processing on Commodity Graphics Hardware

J. Setoain, C. Tenllado, M. Prieto
ArTeCS Group, Complutense University of Madrid, Madrid, Spain
jsetoain@pdi.ucm.es
{tenllado, mpmatias}@dacya.ucm.es

D. Valencia, A. Plaza, J. Plaza
GRNPS Group, University of Extremadura, Caceres, Spain
{davaleco, aplaza, jplaza}@unex.es

## Abstract

*Many recent research efforts have been devoted to the use of commodity hardware for solving computationally-intensive scientific problems. Among such problems, hyperspectral imaging has created new processing challenges in the remote sensing community. Hyperspectral sensors are now capable of collecting hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth. For instance, NASA is continuously gathering high-dimensional image data with hyperspectral sensors such as Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS).*

*The increasing programmability and parallelism of commodity graphics processing units (GPUs) makes them strong candidates for addressing some of these challenges. In this paper, we describe a GPU-based framework for implementation of hyperspectral image processing algorithms which takes advantage of multiple levels of parallelism found in modern GPUs. This framework is inexpensive, uses readily available PC graphics hardware boards, and provides the desired performance at the quality required. Experimental results are presented and discussed in the context of a realistic application, based on hyperspectral data collected by NASA's AVIRIS system.*

## 1. Introduction

Imaging spectroscopy, also known as hyperspectral imaging [2], is a new technique that has gained tremendous popularity in many research areas, including satellite imaging and aerial reconnaissance. Most applications of this emerging technology require timely responses for swift decisions which depend upon high computing performance of algorithm analysis. Examples include target detection for military and defense/security deployment, urban planning and management, risk/hazard prevention and response including wild-land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination. The concept of hyperspectral imaging was introduced when NASA's Jet Propulsion Laboratory developed the Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) system[4] , which covers the wavelength region from 0.4 to $2.5\mu m$ using 224 spectral channels and nominal spectral resolution of 10 nm (see Fig. 1). This sensor routinely collects images hundreds of kilometers long, each of them with hundreds of MBs in size, and this explosion in the amount of collected information has rapidly introduced new processing challenges.

Last-generation hyperspectral image analysis algorithms naturally integrate the wealth spatial and spectral information contained in the data by treating the input volume as an image cube made up of spatially arranged pixel vectors [10]. 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). Taking in mind that data accesses in these algorithms are regular and predictable, they can be nicely mapped onto parallel vector processor systems such as the NEC SX or the Cray SV series, and to high-performance clusters. Unfortunately, these systems are generally expensive and difficult to adapt to onboard remote sensing data processing scenarios, in which low-weight and low-power integrated components are mandatory to reduce mission payload.

An exciting new development is the emergence of programmable graphics processing units (GPUs) [8, 9]. Driven by the ever-growing demands of the game industry, GPUs have evolved from expensive application-specific units into highly parallel and programmable systems. Although, their programming capabilities still pose important challenges (which still make software development a complicated task), current GPUs are general enough to perform computation beyond the domain of graphic rendering [9].

From the viewpoint of a general-purpose programmer, these platforms can be better abstracted by using a stream

Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06)
0-7695-2637-3/06 $20.00 © 2006 IEEE

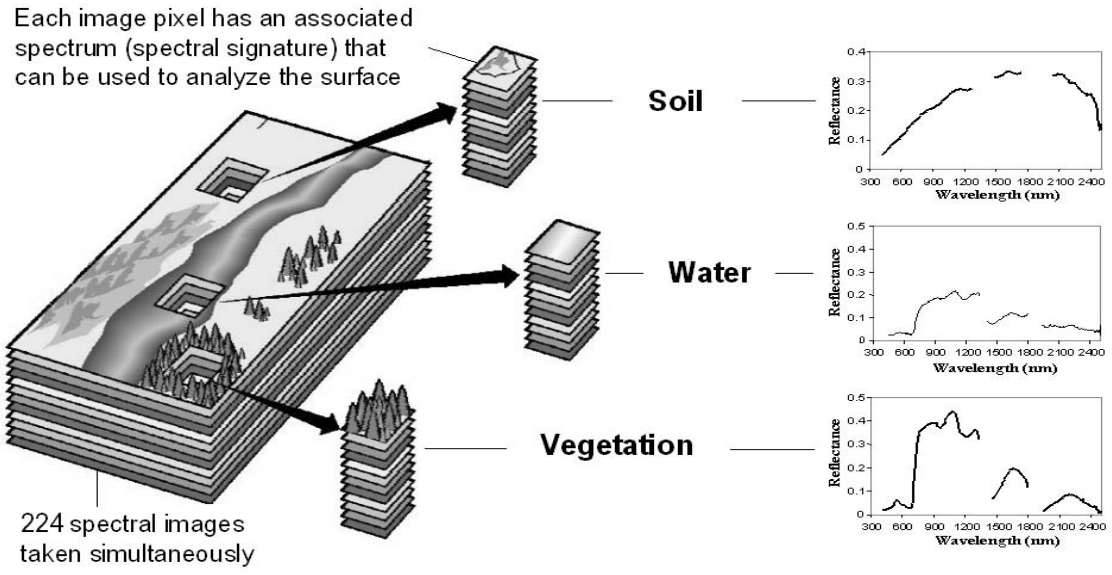



**Figure 1. Concept of hyperspectral imaging using Jet Propulsion Laboratory's AVIRIS system.**

model under which all data portions are represented as streams, which are ordered sets of data, so that applications are constructed by chaining multiple kernels. Kernels operate on entire streams, i.e., taking one or more streams as inputs and producing one or more streams as outputs. They must satisfy one major constraint: their semantic must not depend on the order in which output elements are produced. Thereby, data-level parallelism is exposed to hardware and kernels can operate on streams without any sort of synchronization. Modern GPUs take advantage of these features by replacing the hardware components for complex control flow with extra processing units. Currently, state of the art GPUs are able to deliver peak performances above 300 Gflops, i.e., more than one order of magnitude over high-end microprocessors.

Hyperspectral imaging algorithms can fully benefit from GPU-based hardware and programming models, thus taking advantage of additional features such as the availability of fast and accurate transcendental functions instructions, as well as the compact size and relatively low cost of these units. As mentioned above, low-weight integrated components such GPUs are desirable to reduce payload and data transmission overheads in onboard processing, and to satisfy the extremely high computational requirements sought in many planned and future Earth-observing missions.

In this paper, we will outline a GPU-based implementation of a hyperspectral image classification algorithm, used as a representative case study of techniques that take into account both the spatial and spectral information of the data in simultaneous fashion. The remainder of the paper is organized as follows. Section 2 gives a brief outline of the architecture of modern GPUs. Section 3 develops the proposed algorithm and describe its implementation using a stream processing model. Section 4 provides an assessment of the parallel algorithm from the viewpoint of both classification accuracy and parallel performance on a NVIDIA GeForce 7800GTX. Finally, Section 5 concludes with some remarks and provides hints at plausible future research.

## 2 Commodity Graphic Processing Units Architecture

In this Section we briefly describe the different architectural features of GPUs, which help to understand how to structure algorithms and data representations and map them efficiently onto GPUs. Modern GPUs implement a generalization of the traditional rendering pipeline [6, 8], illustrated in Figure 2. The vertex and fragment processors are the programmable elements of the pipeline, and the programs they execute are called vertex and fragment shaders, respectively. The vertex stage performs operations on the stream of vertices sent to the GPU. Vertex processors transform each of these vertices into a 2D projection space, and apply lighting to determine their colors. This stage is now fully programmable, a fact which allows for custom transformations for special effects. Once transformed, vertices are reassembled into triangles and rasterized 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,

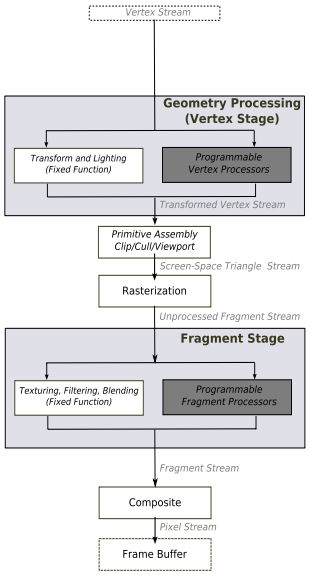



**Figure 2. Programmable Graphics Pipeline**

the rasterization stage also interpolates attributes stored at the vertices, such as texture coordinates, and stores the interpolated values at each fragment.

Fragment processors compute output colors using arithmetic operations and texture lookups. For this purpose, texture memories can be indexed with different texture coordinates and texture values can be retrieved from multiple textures. In order to increase computational efficiency, these processors support short-vector instructions that operate on 4-element vectors (Red/Green/Blue/Alpha channels) in SIMD fashion, and include dedicated texture units that operate with a deeply pipelined texture cache [8]. Furthermore, the latency of data accesses is hidden using texture prefetching and transferring 2D blocks of data from the memory to the texture cache [7]. Finally, results from the fragment processors are combined with the existing data stored at the associated 2D locations in the frame buffer to produce the final colors.

For non-graphics related applications, fragment processors are typically more useful than vertex processors: there are usually more fragment than vertex pipelines, and fragment processors have better memory access performance. However, GPU-based architectures evolve fast and this situation may change in the near future.

To summarize the architecture outline above, it is important to note that mapping applications onto GPU requires structuring the computations in a stream-flow model, in which kernels are expressed as fragment programs and data streams as textures [1].

The programming effort above has already been studied by many researchers, who have successfully ported a large number of scientific applications [9]. Nevertheless, to the best of the authors' knowledge, hyperspectral image processing algorithms have not yet been ported to GPUs. 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.

## 3 GPU-based morphological algorithm

This section first develops a morphological algorithm for hyperspectral image classification. Then, it provides a cost-effective GPU-based implementation based on the design principles outlined in the previous Section.

### 3.1 General Algorithm

Mathematical morphology [13] is a classic non-linear spatial processing technique that provides a remarkable framework to achieve the desired integration of spatial and spectral information in hyperspectral image analysis. Before describing our proposed approach, let us denote by $f$ 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 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 (SE) around each image pixel vector [13]. To do so, we first define a cumulative distance between one particular pixel $f(x,y)$, where $f(x,y)$ denotes an N-D vector at discrete spatial coordinates $(x,y)\varepsilon Z^2$, and all the pixel vectors in the spatial neighborhood given by $B$ ($B$-neighborhood) as:

$$D_B[f(x,y)] = \sum_i \sum_j SID[f(x,y), f(i,j)] \qquad (1)$$

where $(i,j)$ are the spatial coordinates in the $B$-neighborhood and SID is the spectral information divergence, a commonly used distance in remote sensing applications [2]:

$$\begin{aligned} SID[f(x,y), f(i,j)] &= \sum_{l=1}^{N} p_l \cdot \log(\frac{p_l}{q_l}) \\ &+ \sum_{l=1}^{N} q_l \cdot \log(\frac{q_l}{p_l}) \end{aligned} \qquad (2)$$

where:

$$p_l = \frac{f_l(x,y)}{\sum_{k=1}^{N} f_k(x,y)} \qquad (3)$$

$$q_l = \frac{f_l(i,j)}{\sum_{k=1}^{N} f_k(i,j)} \qquad (4)$$

Based on the distance above, we calculate the extended morphological erosion of $f$ by $B$ for each pixel in the input data scene as follows [11]:

$$(f \Theta B)(x,y) = argmin_{(i,j)}\{D_B[f(x+i,y+j)]\} \quad (5)$$

where the *argmin* operator selects the pixel vector is most highly similar, spectrally, to all the other pixels in the $B$-neighborhood. On the other hand, the extended morphological dilation of $f$ by $B$ is calculated as follows [11]:

$$(f \oplus B)(x,y) = argmax_{(i,j)}\{D_B[f(x+i,y+j)]\} \quad (6)$$

where the *argmax* operator selects the pixel vector that is most spectrally distinct to all the other pixels in the $B$-neighborhood. With the above definitions in mind, we provide below an unsupervised classification algorithm for hyperspectral imagery which relies on extended morphological operations:

*Automated Morphological Classification (AMC)*

***Inputs:*** Data cube $f$, structuring element $B$, Number of classes $c$.

***Outputs:*** 2-D matrix which contains a classification label for each pixel $f(x,y)$ in the input image:

1. Initialize a morphological eccentricity index score $MEI(x,y) = 0$ for each pixel.
2. Move $B$ through all the pixels of $f$, defining a local spatial search area around each $f(x,y)$ and calculate the maximum and minimum pixel at each $B$-neighborhood using dilation and erosion respectively. Update the $MEI$ at each pixel using the SID between the maximum and the minimum.
3. Select the set of $c$ pixel vectors in $f$ with higher associated score in the resulting $MEI$ image and estimate the sub-pixel abundance $\alpha_i(x,y)$ of those pixels at $f(x,y)$ using the standard linear mixture model described in [2].
4. Obtain a classification label for each pixel $f(x,y)$ by assigning it to the class with the highest sub-pixel fractional abundance score in that pixel. This is done by comparing all estimated abundance fractions $\{\alpha_1(x,y), \alpha_2(x,y), \ldots, \alpha_c(x,y)\}$ and finding the one with the maximum value, say $\alpha_{i^*}(x,y)$, with $i^* = arg\{max_{1 \leq i \leq c}\{\alpha_i(x,y)\}\}$.

One of the main features of the algorithm above is regularity in the computations. As shown in previous work [12], its computational complexity is $O(p_f \times p_B \times N)$, where $p_f$ is the number of pixels in $f$ and $p_B$ is the number of pixels in $B$. This results in high computational cost in real applications. However, an adequate GPU-based implementation can greatly enhance the computational performance of the algorithm, as explained in the following subsection.

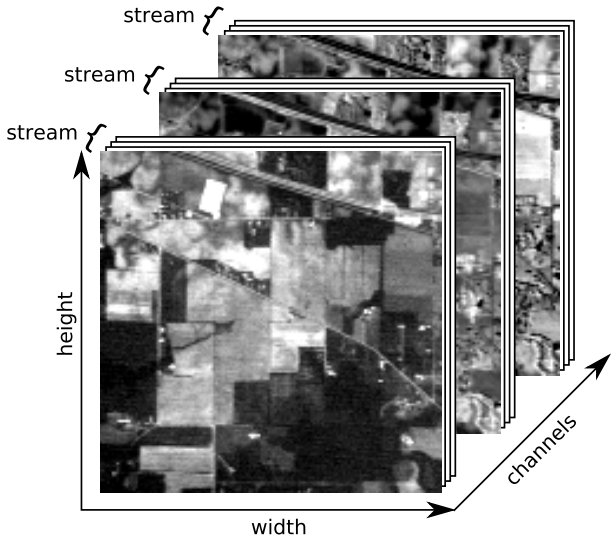



**Figure 3. Hyperspectral image split into streams with four spectral channels.**

### 3.2 GPU Implementation

The first issue that has to be addressed in the description of our GPU-based implementation, from an implementation point of view, is the mapping of hypespectral images onto textures, following the stream programming model described in Section 2. Although a single 3D texture would be the most natural choice, GPUs work better with bidimensional textures. Therefore, we have opted to split every hyperspectral image into a stack of 2D textures, as Figure 3 shows. Furthermore, we have mapped every group of four consecutive channels onto the RGBA color channels of the texture elements, in order to take advantage of the the SIMD capabilities of the fragment processors. In case of a target hyperspectral image that exceeds the capacity of the GPU memory, we split it into multiple chunks made up of entire pixel vectors, i.e. every chunk incorporates all the spectral information on a localized spatial region. Figure 3 represents one of these chunks.

Our stream based implementation of the AMC algorithm, comprises the following stages (Fig. 4):

1. *Stream uploading*
2. *Normalization*
3. *Cumulative distance computation*
4. *Maximum and minimum*
5. *Compute SID*
6. *Stream downloading*

The *Stream Uploading* stage divides the image into chunks and uploads them as streams into the GPU memory. Once the streams have been transferred, *normalization* reduces pixel vectors and obtains normalized values following equations 3 and 4, as Figure 4 illustrates.

The extended morphological erosion and dilation are produced from these normalized streams as described in Section 3.1. Expression (5) indicates that for each pixel vector, we have to compute $B^2$ cumulative SID's, obtained by applying (1) on $B^2$ neighboring pixel vectors. The *Cumulative Distance* stage (See Fig. 4) produces a different *cumulative* stream for each of these $B^2$ neighbors, i.e. *accum0* (shown in figure 4) stores the cumulative distance of *neighbor 0* for all pixels in the incoming chunk, and so forth.

Erosion and Dilation stages are then performed on the *Maximum and Minimum* stage. The kernel produces a new stream, which contains the index of the neighbors with maximum and minimum cumulative distance for each pixel vector. As shown by Figure 4, it uses as inputs the *cumulative* streams generated in the previous stage.

Finally, the *SID Compute* stage, uses the *Maximum/Minimum* stream to compute the SID distance for the pixel vectors selected, completing step 2 of the AMC algorithm.

To conclude this section, we would like to emphasize that every stage in the flowchart described in Figure 4 comprises at least one kernel, although in most cases the stage is implemented using more than one kernel. Furthermore, output streams often loop back to the input until the whole stage completes, as it is indeed the case in the *Cumulative Distance* stage.

## 4 Experimental results

In this section, we provide an assessment of the effectiveness of the proposed GPU-based parallel algorithm on current state-of-the-art GPU platforms. We have also extended our analysis to a *three-years-old* system in order to anticipate potential benefits that may be achieved on future generations of GPUs. The details of both experimental platforms are described in the following Section.

### 4.1 Experimental Platforms

Our study has been performed on two different generations of GPUs corresponding to NV30 and G70 families. As a reference, we also include performance results on contemporary Intel processors. Tables 1 and 2 describe the main features of our experimental platforms. Here, implementations have been built with two different compilers, the GNU-C/C++ compiler (version 4.0, optimizing flags *-O3 -msse*) and the Intel C/C++ compiler with autovectorization capabilities enabled (version 9.0, optimizing flags *-O3 -tpp7 -restrict -xP*). In general, the Intel compiler versions achieve better performance since it is able to vectorize several loops. On the other hand, fragment programs have been hand-coded using Cg [5], and all Cg fragment programs were compiled using the profile fp30, which allows for advanced NVidia fragment programmability [3]. We should also note that our CPU reference implementations have also been hand-tuned to exploit data locality and maximize computation reuse.

**Table 1. Experimental GPU's Features**

| Feature | 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 2. Experimental CPU's Features**

| Feature | Pentium 4 (Northwood M0) | 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 |

### 4.2 Hyperspectral image data

The Indian Pines AVIRIS hyperspectral data set considered in experiments consists of 2166 samples by 614 lines and 216 spectral bands (around 500 MB in size). It was gathered over the Indian Pines test site in North-Western Indiana, a mixed agricultural/forested area, early in the growing season. The data set, represents a very challenging classification scenario, mainly due to the early growth stage of crops at the time of data collection. Discriminating among the major classes under this circumstances is extremely difficult, a fact that has made this scene a universal and extensively used benchmark to validate the classification accuracy of hyperspectral analysis algorithms.

Fig. 5 (left) shows a spectral band at 587 nm of the original scene, and Fig. 5 (right) shows a ground-truth map available for the area, composed of 30 mutually-exclusive land cover classes. Part of these data, including ground-truth, are available on-line (from http://dynamo.ecn.purdue.edu/~biehl/MultiSpec).

### 4.3 Assessment of the GPU-based algorithm

Before empirically investigating the performance of the GPU-based implementation, we first briefly discuss the classification accuracy obtained for the different ground-truth classes available for the AVIRIS Indian Pines scene. For this purpose, Table 3 shows the overall and individual classification accuracies (in percentage) achieved by the proposed parallel algorithm, using a 3x3 structuring element for the construction of morphological operations. As shown by Table 3, the lowest classification accuracies were reported for the buildings class, due to the presence of mixed pixels in this area as a result of the coarse spatial resolution of the scene, and for some of the corn classes due to the early growth stage of crops in these areas which also results in heavily mixed pixels. Quite opposite, very high classification accuracies were reported for macroscopically pure classes such as BareSoil, Concrete/Asphalt or Woods. The measured overall accuracy of 72.35% is a very good classification result given the extremely high complexity of the data set, as reported in previous studies [12].

In order to study the scalability of our parallel GPU-based implementation, we have tested a wide range of image sizes. The largest one corresponds to the full Indian Pines data set, whereas the others are cropped portions of this image. Tables 4 and 5 show the execution times for both the CPU and GPU based implementations. The full Indian Pines data set has been processed in only 12 seconds in spite of the overheads involve in data transfer between main memory and the GPU, which confirms our introspection that GPUs are indeed very suitable for hyperspectral image processing.

Results in Tables 4 and 5 further demonstrate that the complexity of our GPU-based implementations scales linearly with the problem size, i.e., doubling the size doubles the execution time. When the latest-generation platforms were considered, the speedups of the GPU-based implementations over their CPU-based counterparts are outstanding. Using the GNU C/C++ compiler, the speedup remains close to 55 for all the image sizes. Although the Intel compiler reduces this value to 20, this result is still highly acceptable in the context of the considered application.

Finally, we should also note the remarkable relative evolution of both the CPU- and the GPU-based implementations. For instance, our CPU-based implementation on the most recent family of Intel microprocessors only achieved marginal performance improvement (below than 10%) with regards to the previous generation considered in experiments. Along the same period of time, NVidia GPUs have multiplied by six the number of fragment processors, and increased the on-board memory bandwidth. These enhancements together result in a remarkable 400% improvement.

**Table 3. Classification accuracy for each ground-truth class**

| Class | Accuracy (%) |
|---|---|
| BareSoil | 98.05 |
| Buildings | 30.43 |
| Concrete/Asphalt | 96.24 |
| Corn | 99.37 |
| Corn? | 86.77 |
| Corn-EW | 37.01 |
| Corn-NS | 91.50 |
| Corn-CleanTill | 65.39 |
| Corn-CleanTill-EW | 69.88 |
| Corn-CleanTill-NS | 71.64 |
| Corn-CleanTill-NS-Irrigated | 60.91 |
| Corn-CleanTilled-NS? | 70.27 |
| Corn-MinTill | 79.71 |
| Corn-MinTill-EW | 65.51 |
| Corn-MinTill-NS | 69.57 |
| Corn-NoTill | 87.20 |
| Corn-NoTill-EW | 91.25 |
| Corn-NoTill-NS | 44.64 |
| Fescue | 42.37 |
| Grass | 70.15 |
| Grass/Trees | 51.30 |
| Grass/Pasture-mowed | 79.87 |
| Grass/Pasture | 66.40 |
| Grass-runway | 60.53 |
| Hay | 62.13 |
| Hay? | 61.98 |
| Hay-Alfalfa | 83.35 |
| Lake | 83.41 |
| NotCropped | 99.20 |
| Oats | 78.04 |
| Road | 86.60 |
| Woods | 88.89 |
| **Overall:** | **72.35** |

Figure 6 graphically describes these effects and provides some insights on the expected computational power of future hyperspectral imaging developments based on commodity graphics hardware platforms.

## 5 Conclusions

In this paper, we have explored the viability of using commodity graphics hardware for efficiently implementing last-generation hyperspectral image processing algorithms that make use of spatial and spectral information in simultaneous fashion. This approach represents a cost-effective alternative to high performance clusters, which are expensive and difficult to adapt to current mission payload requirements in onboard remote sensing. Experimental results show outstanding speedups, which are expected to produce a significant impact in the remote sensing community.

Despite the impressive figures reported in this work, there is still much room for improvement in our GPU-based

**Table 4. Execution time (in milliseconds) for both the CPU and GPU implementations for different image sizes. Programs where compiled with gcc-4.0**

| Size (MB) | P4 C | Prescott | FX5950 U | 7800 GLX |
|---|---|---|---|---|
| 68 | 91.7453 | 84.0052 | 6.79324 | 1.55211 |
| 136 | 183.32 | 167.852 | 19.572 | 3.067 |
| 205 | 274.818 | 251.427 | 29.2864 | 4.57477 |
| 273 | 367.485 | 336.239 | 39.0221 | 6.0956 |
| 410 | 550.158 | 502.935 | 40.4066 | 9.16738 |
| 547 | 734.243 | 671.157 | 53.9204 | 12.1771 |

**Table 5. Execution time (in milliseconds) for both the CPU and GPU implementations for different image sizes. Programs where compiled with the Intel C/C++ compiler.**

| Size (MB) | P4 C | Prescott | FX5950 U | 7800 GLX |
|---|---|---|---|---|
| 68 | 55.5 | 46.7 | 6.79324 | 1.55211 |
| 136 | 110.7 | 93.2 | 19.572 | 3.067 |
| 205 | 166.2 | 139.7 | 29.2864 | 4.57477 |
| 273 | 222.2 | 186.4 | 39.0221 | 6.0956 |
| 410 | 332.6 | 279.4 | 40.4066 | 9.16738 |
| 547 | 444.1 | 372.8 | 53.9204 | 12.1771 |

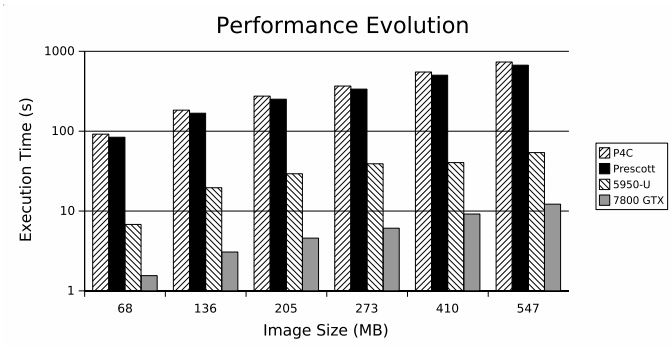



**Figure 6. Performance of the different CPU and GPU implementations . Compilation was done with gcc.**

implementation. In future research, we plan to study additional partitioning strategies to balance the CPU and GPU workloads.

## References


[1] 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.*, 23(3):777–786, 2004.

[2] C.-I. Chang. *Hyperspectral imaging: Techniques for spectral detection and classification*. Kluwer, Academic Publishers, 2003.

[3] M. J. K. Editor. NVIDIA OpenGL Extension Specifications. Available online at http://www.nvidia.com/dev_content/nvopenglspecs/nvOpenGLspecs.pdf, May 2004.

[4] R.-O. G. et al. Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris). *Remote Sensing of Environment*, 65:227–248, 1998.

[5] R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[6] J. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics. Principles and Practice. 2nd Edition in C*. Addison-Wesley, 1996. FOL j 96:1 1.Ex.

[7] Z. S. Hakura and A. Gupta. The design and analysis of a cache architecture for texture mapping. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 108–120, June 1997.

[8] J. Montrym and H. Moreton. The geforce 6800. *IEEE Micro*, 25(2):41–51, 2005.

[9] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, Aug. 2005.

[10] A. Plaza, P. Martinez, R. Perez, and J. Plaza. Spatial/spectral endmember extraction by multidimensional morphological operations. *IEEE Transactions on Geoscience and Remote Sensing*, 40(9):2025–2041, September 2002.

[11] A. Plaza, P. Martinez, J. Plaza, and R. Perez. Dimensionality reduction and classification of hyperspectral image data using sequences of extended morphological transformations. *IEEE Transactions on Geoscience and Remote Sensing*, 43(3):466–479, March 2005.

[12] A. Plaza, D. Valencia, J. Plaza, and P. Martinez. Commodity cluster-based parallel processing of hyperspectral imagery. *Journal of Parallel and Distributed Computing*, 66(3):345–358, March 2006.

[13] P. Soille. *Morphological Image Analysis: Principles and Applications 2nd Ed.* Springer, Berlin, 2003.

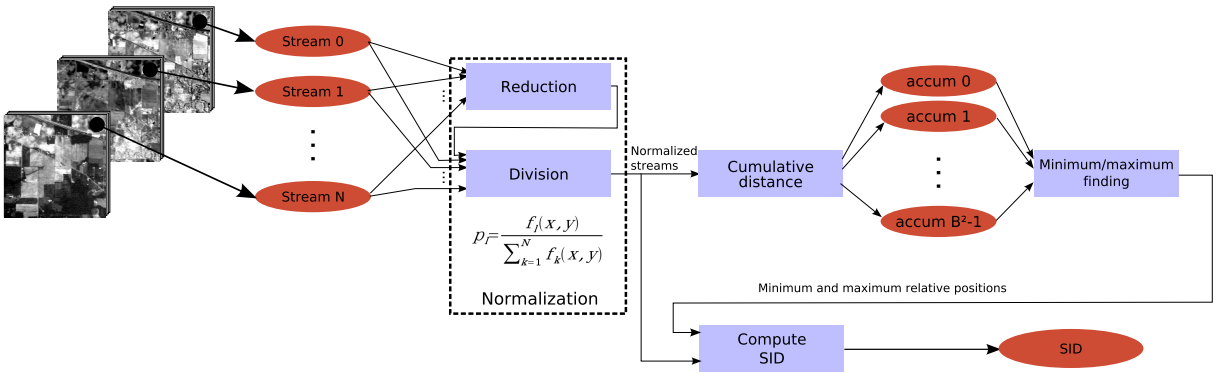
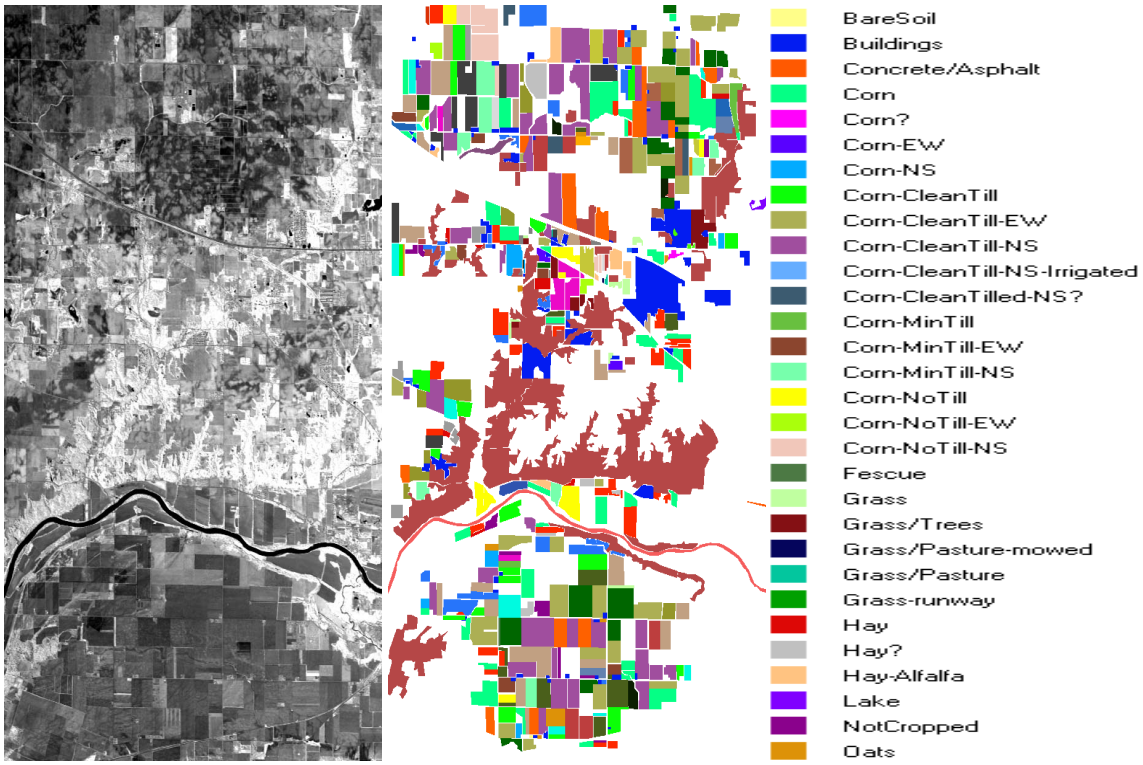



**Figure 4. Stream based AMC algorithm. The image is divided into a series of streams that go through the algorithm stages.**




(a) (b)

**Figure 5. (a) Spectral band at 587 nm wavelength of an AVIRIS scene comprising agricultural and forest features of Indian Pines test site, Indiana. (b) Ground truth map with thirty mutually-exclusive land-cover classes.**