PARALLEL IMPLEMENTATION OF THE SIMPLEX GROWING ALGORITHM FOR HYPERSPECTRAL UNMIXING USING OPENCL

Sergio Bernabé[a], *Guillermo Botella*[a], *José M. R. Navarro*[a], *Carlos Orueta*[a], Manuel Prieto-Matias[a] and Antonio Plaza[b]

[a]Department of Computer Architecture and Automation, Complutense University, 28040 Madrid, Spain
[b]Hyperspectral Computing Laboratory, University of Extremadura, 10003 Cáceres, Spain

ABSTRACT

Many algorithms for spectral unmixing have been proposed in the last years applied on hyperspectral imaging. This process is composed by three stages where the extraction of endmembers is the most consuming step. However, endmember extraction algorithms (EEAs) can be computationally very expensive and its acceleration on parallel architectures is still an interesting and open problem. In this paper, we present a parallel implementation of the simplex growing algorithm for hyperspectral unmixing called P-SGA on different platforms using the OpenCL framework. The proposed implementation exploits the memory hierarchy to accelerate the parts of this method which are more time-consuming. The proposed algorithm is evaluated in terms of both accuracy and computational performance through Monte Carlo simulations using the following architectures: multi-core Xeon CPU, NVidia GeForce GTX 980 GPU and Intel Xeon Phi accelerator. Experiments are conducted using real hyperspectral data set revealing considerable acceleration factors, which satisfies the real-time constraints given by the data acquisition rate.

Index Terms— Hyperspectral imaging, spectral unmixing, high performance computing (HPC), simplex growing algorithm (SGA), OpenCL.

1. INTRODUCTION

Hyperspectral unmixing has been a challenging task in the last decades due to improved high spatial and spectral resolution provided by hyperspectral spectrometers. In this process it is common to collect hundreds or even thousands of nearly contiguous spectral bands of the same area, where the resulting data cube of images typically comprises several GB per flight. This constraint generally requires the use of high performance computing techniques to obtain a fast response in remote sensing applications [1] such as environmental monitoring, mineral detection or military and defense/security purposes.

This work has been supported by the Spanish Ministry of Economy and Competitiveness (MINECO) through the research contract TIN 2012-32180 and the Formación Posdoctoral programme (FPDI-2013-16280).

Previous research studies [2] have shown that the ever-growing computational demands of these applications, which often require real- or near real-time responses, can fully benefit from these emerging computing platforms. Unfortunately, programming heterogeneous systems is a laborious task that often involves a deep knowledge of the underlying architecture. Recently, OpenCL has been used to implement many applications on heterogeneous systems such as graphics processing units (GPUs) [3], multi-core processors [4], the Intel Xeon Phi [5] and other custom devices [6]. Based on these studies, previous works have shown that the use of multi-core and GPUs devices achieve significant acceleration factor over the Intel Xeon Phi device.

In hyperspectral data analysis, one of the main problems is the presence of mixed pure pixels (called endmembers) collected by imaging spectrometers such as the Jet Propulsion Laboratory's Airborne Visible Infra-Red Imagin Spectrometer (AVIRIS). These pixels are highly mixed in nature due to spatial resolution and other phenomena. In this case, several spectrally pure signatures are combined into the same (mixed) pixel. Spetral unmixing [7] is an important technique to solve this problem identifying pure spectral components and their abundance fractions in each (possibly mixed) pixel of the scene. Popular approaches for this purpose have been the linear and nonlinear mixture models (LMM and NLMM respectively). In practice, the LMM is more flexible to adapt it to different analysis scenarios and the most used for the community to unmix remotely sensed hyperspectral data. Recently, several unmixing approaches have been proposed to solve the aforementioned problem using high performance computing systems and OpenCL: multi-core processors [4], commodity graphics processing units [3] and accelerator devices [4].

In this paper, a parallel implementation of the simplex growing algorithm (SGA) for hyperspectral unmixing on different platforms using the OpenCL framework is proposed. Initially, a sequential version was developed in [8] as an alternative to the N-finder algorithm and showed to be a promising endmember extraction technique. The parallel simplex growing algorithm (P-SGA) implementation has been com-

pared with a single-threaded C-based implementation of the SGA algorithm testing the well-known AVIRIS image scene, Cuprite, which has been widely used to study endmember extraction on three different architectures: Intel Xeon processor E5-2695 v3 at 2.30GHz (multi-core CPU), NVidia GeForce GTX 980 (GPU) and Intel Xeon Phi accelerator.

The remainder of this paper is organized as follows. Section 2 briefly describes the original SGA method. Section 3 describes the proposed parallel implementation using the OpenCL framework. Section 4 presents an experimental evaluation of the proposed implementation in terms of both accuracy and parallel performance using real data on three platforms. Finally, Section 5 outlines the conclusions of the paper with some remarks and pointers to future work.

2. DESCRIPTION OF THE METHOD

The initial SGA method for spectral unmixing was originally developed in [8]. This method belongs to the second stage in the unmixing process called endmember extraction algorithms (EEA). This algorithm (see Algorithm 1) assumes the presence of pure pixels on the data, where initially we generate the first endmember by a randomly generated target pixel $\mathbf{t}$ (step 3 in Algorithm 1). Experimental results have shown that different selections for the target pixel $\mathbf{t}$ has not effect on the final set of endmembers. The following endmembers are generated by the volume generation defined by the step 6, where the maximum of all them is selected. This volume generation will be repeated until a desired number of endmembers p. Finally, a set of $\{m_1, m_2, \ldots, m_p\}$ endmembers are obtained.

3. OPENCL IMPLEMENTATION

In this section our mapping of the P-SGA algorithm (see Fig. 1) on different heterogeneous platforms is described. The main goal in this kind of implementations is to use a common code written in OpenCL for the different platforms. This standard is a framework for parallel implementation that allows the execution of parallel programs on heterogeneous platforms. It is currently supported by several hardware devices, such as CPUs, GPUs, DSPs, FPGAs and other accelerators. OpenCL is based on the host-device model, where we have one or more kernels to express the parallelism in our program.

The OpenCL programming model divides a program workload into *work-groups* and *work-items*. We can use a *work-group* with a limited number of *work-items* depending on the selected device. Each *work-group* is executed independently with respect to other *work-groups*. On the other hand, the memory model distinguishes different memory regions: global, local, constant and private memories. Global memory is read-write accessible by all *work-items* across all *work-groups*, and it usually corresponds to the DRAM memory device, which carries a high latency memory access.

Algorithm 1 : Pseudocode of the Simplex Growing Algorithm(SGA)

1: INPUT: $\mathbf{X}, \widehat{p} > 0$
   % $\mathbf{X}$ is $L \times n_s$ matrix with the hyperspectral data set, where L is the number of spectral bands and n_s is the number of pixels.
   % $\widehat{p}$ is an estimated endmember value generated by any algorithm to estimate the number of endmembers such as VD or HySime algorithms.
2: OUTPUT: $\widehat{\mathbf{M}}$
   % $\widehat{\mathbf{M}}$ is $L \times p$ matrix with the spectral signatures for each endmember.
3: $n = 1$
   % $\mathbf{r}$ is an array corresponding to a pixel with all the spectral bands.
4: $\mathbf{m}_n := \arg\left\{\max_{\mathbf{r}} \left[\left|\det\begin{bmatrix}1 & 1\\ \mathbf{t} & \mathbf{r}\end{bmatrix}\right|\right]\right\}$ % First initial endmember pixel
5: $\widehat{\mathbf{M}}_n := [\mathbf{m}_n]$
6: **for** n to $p-1$ **do**
7: $\boldsymbol{V}(\mathrm{m}_1, \ldots, \mathrm{m}_n, \mathbf{r}) := \dfrac{\left|\det\begin{bmatrix}1 & 1 & \ldots & 1 & 1\\ \mathbf{m_1} & \mathbf{m_2} & \ldots & \mathbf{m}_n & \mathbf{r}\end{bmatrix}\right|}{n!}$
8: $\mathbf{m}_{n+1} := \arg\left\{\max_{\mathbf{r}} \left[\boldsymbol{V}(\mathbf{m}_1, \ldots, \mathbf{m}_n, \mathbf{r})\right]\right\}$
9: $\widehat{\mathbf{M}} := [\widehat{\mathbf{M}}, \mathbf{m}_{n+1}]$
10: **end for**

Local memory is a shared read-write memory accessible from all *work-items* of a single *work-group*, and it habitually involves a low latency memory access. Constant memory is a read-only memory that is visible to all *work- items* across all *work-groups*, and private memory, as the name suggests, is only accessible by a single *work-item*.

Before performing any processing on the platform, the hyperspectral image is loaded band by band from the host to device global memory. With this data layout, the access to consecutive pixels in the same spectral band performed by adyacent work-items, can be coalesced into fewer memory transactions. The parallelization consists of the following steps:

1. A matrix of size ($\widehat{p}$+1)*($\widehat{p}$+1), (being $\widehat{p}$ the input parameter which specifies the desired number of endmembers), is built and stored in local memory for each considered endmember. An ad-hoc particular kernel, called *determinant_calculation*, computes the determinant of the previous matrix using an LU decomposition. In this decomposition each *work-item* calculates its value, so that we can obtain a value equal to 0 under the main diagonal once processed each row of the matrix has been processed. In the same kernel, the volume for each determinant is calculated, where each *work-item* obtains its relative value, so it can be divided later on by its factorial in order to obtain the absolute value.

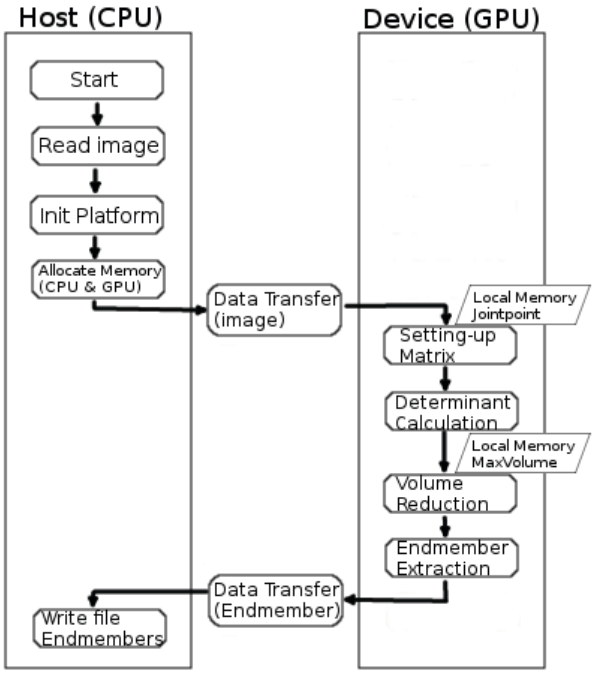

Fig. 1. Flowchart of our mapping of the P-SGA method using OpenCL.

2. Another ad-hoc kernel, called *volume_reduction*, performs a reduction process for all the volume values along two steps. In the first step, each *work-item* compares each element to obtain the maximum value and store it in an array of size $\widehat{p}$, located in the local memory. The second step performs a small reduction in a serial way so the maximum volume is obtained.

3. Once the maximum volume has been calculated, an ad-hoc kernel called *endmember_extraction* is performed in order to extract all the spectral bands from the considered endmember. During the extraction, each *work-item* stores each element in a resultant array between a separation of $\widehat{p}$ elements.

4. These steps are repeated until algorithm reaches up the desired number of endmembers (parameter $\widehat{p}$). Finally, the array with all the obtained endmembers is transferred to the host.

4. EXPERIMENTAL RESULTS

The experiments are carried out using a real hyperspectral scene. The data set used in our experiments is the well-known AVIRIS Cuprite scene, collected by the Airborne Visible Infra-Red Imaging Specrometer (AVIRIS) in the summer of 1997 and available online in reflectance units after atmospheric correction. The portion comprises a relatively large area (350 lines by 350 samples and 20-m pixels) and 224 spectral bands between 0.4 to 2.5 μm and a total size of around 46 MB. Bands 1-3, 105-115, and 150-170 were removed prior to the analysis due to water absorption and low SNR in those bands.

In order to analyze the accuracy of the parallel implementation, the well-known spectral angle distance (SAD) [9] is adopted in Table 1. For this purpose, we have extracted p = 19 endmembers after calculating the virtual dimensionality (VD) [10]. The SAD was performed between the extracted endmembers and ground-truth spectral signatures obtaining very low SAD scores under 6 degrees on average, which indicates that the implementation provides accurate results, since the worst case of SAD is 90 degrees and the best case is 0 degrees.

Table 1. Spectral angle values (in degrees) between the endmembers extracted by SGA and the known ground endmembers in the AVIRIS Cuprite scene.

Alunite	Buddingtonite	Calcite	Kaolinite	Muscovite	Average
8.20°	4.17°	5.87°	10.17°	5.41°	6.76°

In order to evaluate the performance, the proposed P-SGA algorithm has been tested on three different platforms: Intel Xeon processor E5-2695 v3 at 2.30GHz (multi-core CPU), NVidia GeForce GTX 980 (GPU) and Intel Xeon Phi accelerator. The Table 2 summarizes the main features for each platform. Firstly, it is important to emphasize that our parallel method provides exactly the same results, in terms of accuracy, as our single threaded C implementation. On the other hand, both the parallel and the C implementations were compiled using the GNU `gcc`-4.9.2 compiler with the `-O3` optimization flag. Note that for the C implementation, only one of the available CPU cores was used. 10 runs were performed for each experiment and we have reported the mean value (the execution times were very similar, with differences on the order of a few milliseconds).

The performance results with respect to distribution workload are shown in Fig. 2, where bars display execution times for different work-groups sizes (4 to 1024). As can be seen, this parameter is negligible on the CPU platform, but on the other devices, the effects are important on the execution time.

Table 3 shows a breakdown of the execution times after processing real hyperspectral data on the considered platforms. As shown in Table 3, the scene could be processed with significant speedup factor using the GPU, up to 40 times, including both the processing time and the time spent on host/device memory transfers.

5. CONCLUSION AND REMARKS

An OpenCL parallel implementation of the simplex growing algorithm for spectral unmixing is presented. The optimized method exploits the memory hierarchy to accelerate the most

Table 2. OPENCL FEATURE

Device Type	Model	#Cores	Clock Frequence	Global Memory Size	Local Memory Size	Max work-items
CPU	2× Intel Xeon E5-2695	28	2.3 GHz	64 GB	32 KB	8192×8192×8192
GPU	NVIDIA-980GTX	2048	1.126 GHz	4 GB	48 KB	1024×1024×64
Accelerator	Intel Xeon Phi 31S1P	57	1.1 GHz	6 GB	32 KB	8192×8192×8192

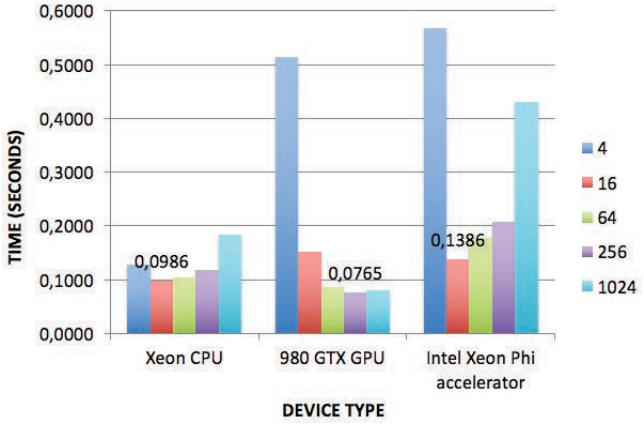

Fig. 2. Execution times achieved on Intel Xeon CPU, NVIDIA 980 GTX GPU and Intel Xeon Phi using different work-group sizes.

Table 3. Processing times (in seconds) and speedups achieved for the proposed implementations in different platforms based on the Algorithm 1 and tested with a real data set.

	AVIRIS Cuprite			
	Serial CPU	**Xeon CPU**	**980 GTX GPU**	**Intel Xeon Phi Accelerator**
Initialization	0.0644	0.4065	0.3024	1.4931
RAM $\rightarrow$ *Device*	–	0.0483	0.0481	0.0566
Compute P-SGA	3.1230	0.0503	0.0284	0.0818
RAM $\leftarrow$ *Device*	–	≈ 0.0000	≈ 0.0000	0.0003
Total Time	3.1230	0.0986	0.0765	0.1386
Speedup	–	31.67x	40.82x	22.53x

time-consuming parts of this method and shows a significant speedup when compared with regard to the sequential version. The performance of the implementation has been evaluated using three different devices: Intel Xeon processor E5-2695 v3 at 2.30GHz (multi-core CPU), NVidia GeForce GTX 980 (GPU) and Intel Xeon Phi accelerator. Experimental results indicate that significant performance could be obtained using the GTX 980 GPU device. Further experimentation with additional large scenes and another high performance computing architectures (low power devices) are desirable in future developments to be used on onboard processing modules in airborne and (particularly) spaceborne Earth observation missions.

6. REFERENCES

[1] A. Plaza and C.-I Chang, *High Performance Computing in Remote Sensing*, Taylor & Francis: Boca Raton, FL, 2007.

[2] 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.

[3] G. M. Callico, S. Lopez, B. Aguilar, J. F. Lopez, and R. Sarmiento, "Parallel implementation of the modified vertex component analysis algorithm for hyperspectral unmixing using opencl," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 8, pp. 3650–3659, 2014.

[4] S. Bernabe, F. D. Igual, G. Botella, C. Garcia, M. Prieto-Matias, and A. Plaza, "Performance portability study of an automatic target detection and classification algorithm for hyperspectral image analysis using opencl," *Proceedings of the SPIE Conference on High-Performance Computing in Remote Sensing*, vol. 9646, pp. 1–9, 2015.

[5] S. Bernabe, G. Martin, J. M. P. Nascimento, J. M. Bioucas-Dias, A. Plaza, G. Botella, and M. Prieto-Matias, "A fast parallel hyperspectral coded aperture algorithm for compressive sensing using opencl," *IEEE 16th International Conference on Computer as a Tool*, pp. 1–6, 2015.

[6] C. Rodriguez-Doñate, G. Botella, C. Garcia, E. Cabal-Yepez, and M. Prieto-Matias, "Early experiences with opencl on fpgas: convolution case study," *IEEE 23rd International Symposium on Field-Programmable Custom Computing Machines*, p. 235, 2015.

[7] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing: geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[8] C.-I Chang, C. Wu, W. Liu, and Y. C. Ouyang, "A new growing method for simplex-based endmember extraction algorithms," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 10, pp. 2804–2819, 2006.

[9] C.-I Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*, Kluwer Academic: New York, 2003.

[10] C.-I Chang and Q. Du, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 608–619, 2004.