Performance-Power Evaluation of an OpenCL Implementation of the Simplex Growing Algorithm for Hyperspectral Unmixing

Sergio Bernabé, Guillermo Botella, José M. R. Navarro, Carlos Orueta, Francisco D. Igual, Manuel Prieto-Matias, and Antonio Plaza, *Fellow, IEEE*

***Abstract*— Over the last few years, several new strategies for spectral unmixing of remotely sensed hyperspectral data have been proposed. Many of them have been developed to solve the most time-consuming and relevant step: endmember extraction. However, unmixing algorithms can be computationally very expensive in terms of processing time and energy consumption, a fact that compromises their use in applications under real-time and energy/power constraints. In this letter, we present a new parallel simplex growing algorithm (SGA) for hyperspectral data which exploits the memory hierarchy with operations in single-precision floating point. Those optimizations accelerate the most time-consuming parts of this method using the open computing language (OpenCL) standard. We have evaluated the performance versus energy consumption using the same open standard for parallel programming over a diverse set of heterogeneous platforms. Experiments have been conducted using real hyperspectral images collected by NASA's Airborne Visible Infrared Imaging Spectrometer and a collection of 24 synthetic hyperspectral images simulated with different sizes and number of endmembers (10–30). Considering the power consumption and OpenCL across all the proposed devices, the analysis presented indicates that the SGA can now be executed in computationally efficient fashion, which was not possible before introducing the parallel implementation described in this letter.**

***Index Terms*— High performance computing (HPC), hyperspectral imaging, open computing language (OpenCL), simplex growing algorithm (SGA), spectral unmixing.**

I. Introduction

SEVERAL techniques have been proposed for spectral unmixing of remotely sensed images in the last decades. This is a challenging task that requires high performance computing (HPC) for processing large images provided by hyperspectral spectrometers [1], particularly in time-critical applications, such as environmental monitoring, mineral detection, or military and defense/security purposes.

Manuscript received June 28, 2016; revised October 17, 2016; accepted November 8, 2016. Date of publication January 18, 2017; date of current version February 23, 2017. This work was supported in part by the EU (FEDER) and the Spanish MINECO, under Grant TIN2015-65277-R, Grant TIN2012-32180 and in part by the Formación Posdoctoral programme under Grant FPDI-2013-16280.

S. Bernabé, G. Botella, J. M. R. Navarro, C. Orueta, F. D. Igual, and M. Prieto-Matias are with the Complutense University, 28040 Madrid, Spain (e-mail: sebernab@ucm.es; gbotella@ucm.es; jmrnavarro@ucm.es; corueta@ucm.es; figual@ucm.es; mpmatias@dacya.ucm.es).

A. Plaza is with the Hyperspectral Computing Laboratory, University of Extremadura, E-10003 Cáceres, Spain (e-mail: aplaza@unex.es).

Color versions of one or more of the figures in this letter are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LGRS.2016.2635585

Previous research studies [2], [3] 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. Many proprietary standards and tools have been designed in order to cover a closed set of architectures, and open computing language (OpenCL) has become a free standard for parallel programming on heterogeneous systems. Since then, it has been adopted by many vendors for all sort of computing devices. The main advantages of implementing OpenCL concern the shorter time to market implementation and the achieved portable codes with acceptable performance. Recently, OpenCL has been used to implement many applications on heterogeneous systems, such as graphics processing units (GPUs) [4], multicore processors [5], the Intel Xeon Phi [6], and other custom devices [7]. Based on these studies, previous works have shown that the use of multicore and GPUs devices achieves 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 Infrared Imaging 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. Spectral unmixing [8] 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 mixture model (LMM) and nonlinear mixture model. In practice, the LMM is more flexible to adapt it to different analysis scenarios and most used for the community to unmix remotely sensed hyperspectral data. Recently, several unmixing approaches have been proposed to solve the aforementioned problem using HPC systems and OpenCL: multicore processors [9], commodity GPUs [10], and accelerator devices [9]. Nevertheless, a detailed assessment in terms of performance and energy consumption using OpenCL has not been conducted in the field of hyperspectral analysis, in which it is particularly important to achieve processing results in real time with low energy cost. However, previous research studies about energy consumption using other

Algorithm 1 : Pseudocode of the SGA

1: INPUT: $\mathbf{Y}$, $\widehat{p} > 0$
   % $\mathbf{Y}$ 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 (samples×lines).
   % $\widehat{p}$ is an estimated value generated by any algorithm to estimate the number of endmembers such as VD [14] or HySime [15] algorithms.
2: OUTPUT: $\widehat{\mathbf{E}}$
   % $\widehat{\mathbf{E}}$ 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: $\mathbf{e}_n := \arg\left\{\max_{\mathbf{r}}\left[\left|\det\begin{bmatrix}1 & 1\\ \mathbf{t} & \mathbf{r}\end{bmatrix}\right|\right]\right\}$ % First initial endmember pixel
5: $\widehat{\mathbf{E}}_n := [\mathbf{e}_n]$
6: **for** n to $p-1$ **do**
7: $\quad V(e_1, \ldots, e_n, \mathbf{r}) := \dfrac{\left|\det\begin{bmatrix}1 & 1 & \ldots & 1 & 1\\ \mathbf{e_1} & \mathbf{e_2} & \ldots & \mathbf{e}_n & \mathbf{r}\end{bmatrix}\right|}{n!}$
8: $\quad \mathbf{e}_{n+1} := \arg\left\{\max_{\mathbf{r}}[V(\mathbf{e_1}, \ldots, \mathbf{e_n}, \mathbf{r})]\right\}$
9: $\quad \widehat{\mathbf{E}} := [\widehat{\mathbf{E}}, \mathbf{e}_{n+1}]$
10: **end for**

parallel programming languages have been proposed in this field [11], [12].

To address these issues, in this letter, we propose a parallel implementation of the simplex growing algorithm (SGA) for hyperspectral unmixing on different platforms using the OpenCL framework. In [13], a sequential version was developed as an alternative to the N-FINDR algorithm and shown to be a promising endmember extraction technique. The same OpenCL SGA (P-SGA) implementation is compared between several heterogeneous architectures over the well-known AVIRIS image scene, cuprite, which has been widely used to study endmember extraction. And also, a collection of 24 synthetic hyperspectral images simulated with different sizes and number of endmembers (10–30) on three different architectures: Intel Xeon processor (multicore), NVidia GeForce GTX (GPU), and Intel Xeon Phi (accelerator). The OpenCL execution on the multicore CPU is considered as the baseline for our experimental study. Experimental results with aggressive optimizations, including operations in single-precision floating point and local memory allowing coalesced accesses to memory, reveal that the accuracy of the proposed parallel implementation is not compromised with a speedup factor of 2 using the GPU platform and including I/O and data transfers between host and device.

II. Simplex Growing Algorithm

The SGA algorithm was proposed in [13] for spectral unmixing. It was an alternative to the N-FINDR algorithm and showed to be a promising endmember extraction technique that belongs to the second stage in the unmixing chain process called endmember extraction algorithm. SGA

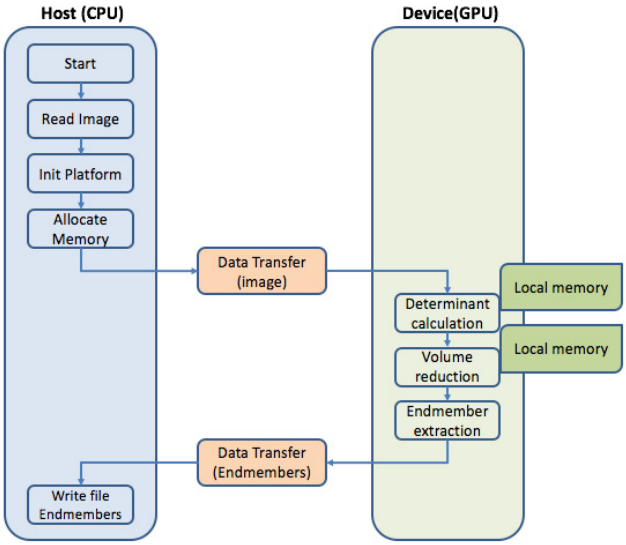

Fig. 1. Block diagram illustrating our mapping of the P-SGA method using OpenCL with local memory.

(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 $\mathbf{t}$ do not effect on the final set of endmembers. The following endmembers are generated by the volume generation defined by step 6, where the maximum of all of them is selected. This volume generation will be repeated until a desired number of endmembers p is obtained. Finally, a set of $\{e_1, e_2, \ldots, e_p\}$ endmembers is obtained.

III. Parallel Implementation

In this section, we show our mapping of the SGA algorithm using a new portable OpenCL code for a diverse set of heterogeneous platforms. As can be seen in Fig. 1, three new kernels have been developed to accelerate the main parts of the algorithm using local memory. We have identified those stages with a profiling of an optimized serial SGA implementation. Those stages are highlighted in green in Algorithm 1. Before explaining the parallel implementation, we will describe the OpenCL framework.

OpenCL is a framework consolidated in the last years as a standard for parallel programming. It is currently supported by several hardware devices, such as CPUs, GPUs, field-programmable gate array, DSPs, and other processors or hardware accelerators. This standard is based on the host-device model, where our program is divided into regions executed on the host and other regions called kernel to express the parallelism in the compute device. The OpenCL standard is a C extension to facilitate the usage of parallelism with vector types, operations, synchronization, and functions to work with *work-items* and *work-groups*. 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, OpenCL defines four different memory spaces for the compute device. Global memory is read–write accessible by all processing

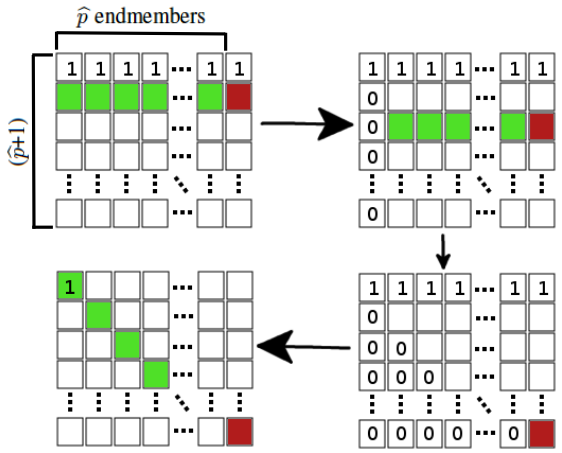

Fig. 2. Parallel LU decomposition where the shaded cells in red and green are stored in global and local memory, respectively.

elements across all *work-groups*, and it usually corresponds to the dynamic random access memory memory device, which carries a high access latency. Local memory is shared by a group of processing elements, and it usually involves 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 parallel processing on the device, a data distribution based on the spectral bands arrangement is considered. With this data layout, the access to consecutive pixels in the same spectral band performed by contiguous work-items, can be coalesced into fewer memory transactions. The parallelization is described as follows.

A matrix of maximum size $(\widehat{p}+1) \times (\widehat{p}+1)$ (being $\widehat{p}$ the input parameter which specifies the desired number of endmembers) is built and stored in local memory for each considered endmember. The first endmember is selected in a random way and replaced by the real as first endmember. Afterward, an *ad hoc* particular kernel, called *determinant_calculation*, computes the determinant of the previous matrix using lower upper (LU) decomposition. We need to calculate a determinant for each sample, processing in parallel as many determinants as existing samples in each *work-group*. In this decomposition (see Fig. 2), 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. For this process, we took into account the endmembers obtained previously. In this way, to look for a new endmember, it is necessary to copy all the previous ones to local memory, so all the possible candidates to become a new endmember are located in the global memory. Due to this strategy, the performance of this calculation is optimized. 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. All obtained values are stored in a transposed matrix of size n_s.

Another *ad hoc* kernel, called *volume_reduction*, performs a reduction process (see Fig. 3) for all the volume values along two steps. In the first step, each *work-item* compares each

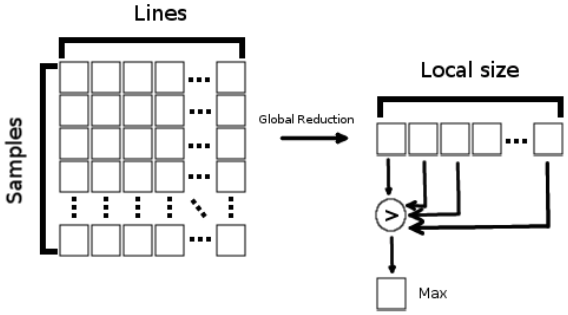

Fig. 3. Reduction process in local memory where local size is the length of the *work-group*.

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.

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 with a stride of $\widehat{p}$ elements.

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

IV. Experiments and Results

A. Real and Synthetic Data Sets

The experiments are carried out using a collection of 24 synthetic hyperspectral images simulated with different sizes (10 000–200 000 pixels) and number of endmembers (10–30). The signatures are obtained from the U.S. Geological Survey library and the scenes are generated using the procedure described in [16] to simulate natural spatial patterns. These images comprise 224 narrow spectral bands between 0.4 and 2.5 μm. On the other hand, we have used the well-known AVIRIS cuprite scene, collected by the AVIRIS in the summer of 1997 and available online in reflectance units after atmospheric correction. The portion comprises a relatively large area (350 lines $\times$ 350 samples and 20-m pixels) and 224 spectral bands between 0.4 and 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 signal-to-noise ratio in those bands prior to the analysis.

B. Accuracy Evaluation

In order to analyze the accuracy of the parallel implementation, the well-known spectral angle distance (SAD) [17] is adopted in Table I. First, it is important to emphasize that our proposed implementation provides exactly the same results, in terms of accuracy, as the single-threaded C-based implementation of the algorithm. For this purpose, we have extracted $p = 19$ endmembers after calculating the virtual dimensionality [14] to estimate the number of endmembers for the AVIRIS cuprite scene. The SAD was performed between the extracted endmembers and ground-truth spectral signatures

TABLE I

SPECTRAL ANGLE VALUES (IN DEGREES) BETWEEN THE ENDMEMBERS EXTRACTED BY SGA AND THE KNOWN GROUND ENDMEMBERS OVER THE AVIRIS CUPRITE SCENE

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

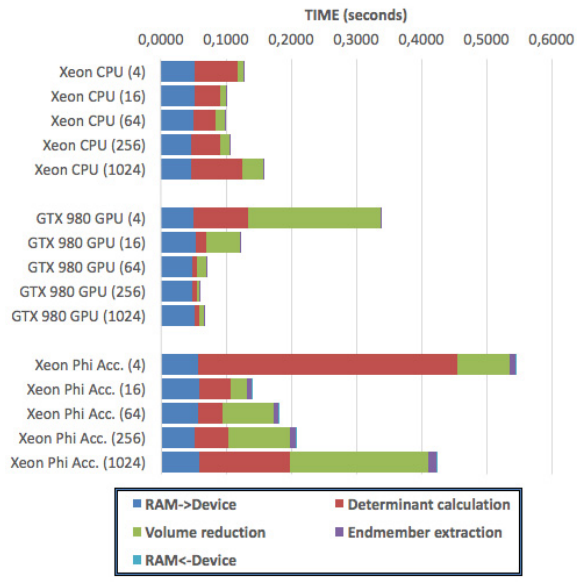

Fig. 4. Execution times achieved on CPU Xeon, NVidia GTX 980 GPU, and Xeon Phi accelerator over the AVIRIS cuprite scene. The serial time executed on CPU Xeon was 3.188 s.

obtaining quite low SAD scores on average, which indicates that the implementation provides accurate results, since the worst case of SAD is 90° and the best case is 0°.

C. Performance Evaluation

In this section, we conduct an experimental evaluation of the computational performance of our optimized SGA implementation. First, the performance evaluation has been obtained on a multicore heterogeneous system equipped with: 2 × Intel Xeon processors E5-2695 v3 at 2.30 GHz with 64 GB of DDR3 RAM memory. An NVidia GeForce GTX 980 GPU with 2048 cores operating at 1.126 GHz and dedicated memory of 4 GB. And finally, an Intel Xeon Phi accelerator 31S1P coprocessor which features 57 cores supporting the execution of four hardware threads (228 hardware threads in total) operating at 1.100 GHz and 8-GB installed RAM memory.

Both the parallel and the CPU implementations were compiled using the GNU `gcc`-4.9.2 compiler with the `-O3` optimization flag for each platform used. Note that for the CPU implementation, only one of the available CPU cores was used. For each experiment, ten runs were performed and we have reported the mean value (the execution times were very similar executions with differences on the order of a few milliseconds). On the one hand, the performance results with respect to distribution work-load are shown in Fig. 4, where bars display a break-down of the execution time for different *work-group* sizes (4–1024) and platforms. The stages represented are the following: *RAM->Device* includes the image transfer from host to device. *Determinant_calculation*, *Volume_reduction*, and *Endmember_extraction* correspond to the execution time for each kernel implemented. Finally, *RAM<-Device* includes the endmember positions transfer time from device to host. As can be seen, *work-group* size is negligible on the CPU platform, but on the other devices, the effects are important on the execution time. Another conclusion is derived from our experiments with *work-group* size equals to 4. On the GPU and Xeon Phi platforms, we have not enough work-items to maximize the computing load and achieve better performance. However, the performance improves on the CPU Xeon, because this size is close to the maximum number of cores.

TABLE II

MEAN EXECUTION TIMES (IN SECONDS) AND JOINTLY SPEEDUP AND OPTIMUM WORK-GROUP SIZE (IN THE PARENTHESES) FOR THE PROPOSED IMPLEMENTATION TESTED ON HETEROGENEOUS PLATFORMS APPLIED TO 24 SYNTHETIC DATA SETS CONTAINING 10, 20, AND 30 ENDMEMBERS

	Xeon CPU		
Size	10 endmembers	20 endmembers	30 endmembers
100×100	0.0072 (4)	0.0131 (16)	0.0198 (64)
100×200	0.0136 (16)	0.0227 (16)	0.0393 (64)
100×300	0.0197 (16)	0.0312 (16)	0.0548 (64)
100×400	0.0255 (16)	0.0410 (16)	0.0705 (64)
100×500	0.0322 (16)	0.0516 (16)	0.0849 (64)
200×500	0.0597 (16)	0.0907 (64)	0.1577 (64)
300×500	0.0834 (16)	0.1313 (64)	0.2271 (64)
400×500	0.1117 (16)	0.1679 (64)	0.3022 (64)
	GTX 980 GPU		
Size	10 endmembers	20 endmembers	30 endmembers
100×100	0.0056 (1.29x, 64)	0.0069 (1.90x, 64)	0.0099 (2.00x,64)
100×200	0.0105 (1.30x, 64)	0.0133 (1.71x, 64)	0.0167 (2.35x, 256)
100×300	0.0146 (1.35x, 256)	0.0178 (1.75x, 256)	0.0245 (2.24x, 256)
100×400	0.0203 (1.26x, 256)	0.0239 (1.72x, 256)	0.0339 (2.08x, 256)
100×500	0.0252 (1.28x, 256)	0.0301 (1.71x, 256)	0.0403 (2.11x, 256)
200×500	0.0501 (1.19x, 256)	0.0592 (1.53x, 256)	0.0802 (1.97x, 256)
300×500	0.0749 (1.11x, 256)	0.0858 (1.53x, 256)	0.1158 (1.96x, 256)
400×500	0.0993 (1.12x, 256)	0.1148 (1.46x, 256)	0.1531 (1.97x, 1024)
	Xeon Phi Accelerator		
Size	10 endmembers	20 endmembers	30 endmembers
100×100	0.0141 (0.51x, 16)	0.0200 (0.66x, 16)	0.0346 (0.57x, 16)
100×200	0.0217 (0.63x, 16)	0.0333 (0.68x, 64)	0.0561 (0.70x, 64)
100×300	0.0296 (0.67x, 16)	0.0462 (0.68x, 64)	0.0773 (0.71x, 64)
100×400	0.0376 (0.68x, 16)	0.0566 (0.72x, 64)	0.0939 (0.75x, 64)
100×500	0.0438 (0.74x, 16)	0.0698 (0.74x, 64)	0.1193 (0.71x, 64)
200×500	0.0773 (0.77x, 16)	0.1275 (0.71x, 16)	0.2471 (0.64x, 16)
300×500	0.1095 (0.76x, 16)	0.1790 (0.73x, 16)	0.3577 (0.63x, 16)
400×500	0.1420 (0.79x, 16)	0.2326 (0.72x, 16)	0.4692 (0.64x, 16)

For illustrative purposes, Table II shows the execution time after processing 24 synthetic data sets on the considered platforms and using an optimum *work-group* size. As can be seen, the performance improves when the number of endmember increases. Moreover, the observed speedups factors are significant for each synthetic scene reaching around two times acceleration considering the largest synthesized image (400 pixel × 500 pixel) based on the NVidia GTX 980 GPU respect to Xeon CPU. To conclude this section, we emphasize that our reported P-SGA processing times are strictly in real-time performance for all the platforms. The cross-track line scan time in AVIRIS, a push-broom instrument, is quite fast (8.3 ms to collect 512 full-pixel vectors). This introduces the need to process the AVIRIS cuprite scene in less than 1.98 s

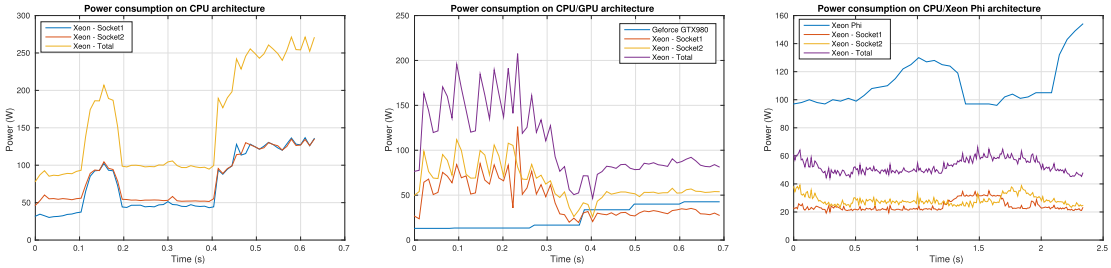

Fig. 5. Power consumption of the P-SGA algorithm executed on the target platforms considering the largest synthesized image.

and the largest synthesized image in less than 3.24 s in order to achieve real-time performance. As noted in Fig. 4 and Table II, both reported times are below 1.98 and 3.24 s to fully achieve real-time performance for this algorithm, for the first time in the literature. These results indicate that our implementation allowed for the first time to meet real-time requirements on a fully operational unmixing chain using this algorithm.

D. Power Consumption Evaluation

In our experiments, the power consumption was measured using the software solution *PowerMeter daemon* (*pmlib*) [18]. Gathering power results periodically from the tools provided by manufacturers, namely, running average power limit for the Intel Xeon CPU, NVidia Management Library for the NVidia GPU, and Intel MicMGMT for the Xeon Phi.

From Fig. 5, a few conclusions can be extracted about the power dissipation. First, if we analyze the graphics, the best platform is CPU/GPU architecture [Fig. 5(b)], which dissipates 134 W on average (20.52 J), and 281 W at most. Compare this, for example, with the 158 W (47.75 J) and 164 W (76.95 J) on average, for CPU and CPU/Xeon Phi architectures, respectively. Second, the best tradeoff solution between performance measured in Mpixel/s and power consumption is achieved by the GPU: 0.006 Mpixel/s/W versus 0.010 and 0.015 Mpixel/s/W for CPU and CPU/Xeon_Phi architectures for a 1024^2 pixel image with 30 endmembers, respectively.

V. Conclusion and Future Research

In this letter, we have developed a new parallel implementation of the SGA for hyperspectral unmixing on different platforms using the OpenCL framework. The obtained results indicate that it is possible to achieve real-time performance by exploiting the memory hierarchy with operations in single-precision floating point using the NVidia GTX 980 GPU. Moreover, the GPU platform achieves the best tradeoff solution ahead of CPU and CPU/Xeon_Phi architectures. Further experimentation with additional large scenes and low power devices will be conducted in the future.

References

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

[2] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, "Deep learning-based classification of hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2094–2107, Jun. 2014.

[3] E. Torti, G. Danese, F. Leporati, and A. Plaza, "A hybrid CPU–GPU real-time hyperspectral unmixing chain," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 945–951, Feb. 2016.

[4] B. Wang, M. Alvarez-Mesa, C. C. Chi, and B. Juurlink, "Parallel H.264/AVC motion compensation for GPUs using OpenCL," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 3, pp. 525–531, Mar. 2015.

[5] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable LDPC decoding on multicores using OpenCL [applications corner]," *IEEE Signal Process. Mag.*, vol. 29, no. 4, pp. 81–109, Jul. 2012.

[6] S. Bernabé *et al.*, "A fast parallel hyperspectral coded aperture algorithm for compressive sensing using OpenCL," in *Proc. IEEE 16th Int. Conf. Comput. Tool*, Sep. 2015, pp. 1–6.

[7] C. Rodriguez-Doñate, G. Botella, C. Garcia, E. Cabal-Yepez, and M. Prieto-Matias, "Early experiences with opencl on FPGAs: Convolution case study," in *Proc. IEEE 23rd Int. Symp. Field-Program. Custom Comput. Mach.*, May 2015, p. 235.

[8] J. M. Bioucas-Dias *et al.*, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 354–379, Apr. 2012.

[9] 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," *Proc. SPIE*, vol. 9646, p. 96460M, Oct. 2015.

[10] G. M. Callicó, S. Lopez, B. Aguilar, J. F. López, and R. Sarmiento, "Parallel implementation of the modified vertex component analysis algorithm for hyperspectral unmixing using OpenCL," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 8, pp. 3650–3659, Aug. 2014.

[11] A. Remón, S. Sánchez, S. Bernabé, E. Quintana-Ortí, and A. Plaza, "Performance versus energy consumption of hyperspectral unmixing algorithms on multi-core platforms," *EURASIP J. Adv. Signal Process.*, vol. 68, pp. 1–15, Dec. 2013.

[12] S. Sánchez, G. León, A. Plaza, and E. S. Quintana-Orti, "Assessing the performance-energy balance of graphics processors for spectral unmixing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2297–2304, Jun. 2014.

[13] C.-I. Chang, C.-C. Wu, W. Liu, and Y.-C. Ouyang, "A new growing method for simplex-based endmember extraction algorithm," *IEEE Trans. Geosci. Remote Sens.*, vol. 44, no. 10, pp. 2804–2819, Oct. 2006.

[14] 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, Mar. 2004.

[15] J. M. Bioucas-Dias and J. M. P. Nascimento, "Hyperspectral subspace identification," *IEEE Trans. Geosci. Remote Sensing*, vol. 46, no. 8, pp. 2435–2445, Aug. 2008.

[16] G. S. Miller, "The definition and rendering of terrain maps," *ACM SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 39–48, Aug. 1986.

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

[18] S. Barrachina *et al.*, "An integrated framework for power-performance analysis of parallel scientific workloads," in *Proc. 3rd Int. Conf. Smart Grids, Green Commun. IT Energy-Aware Technol.*, 2013, pp. 114–119.