Assessing the Performance-Energy Balance of Graphics Processors for Spectral Unmixing

Sergio Sánchez, Germán León, Antonio Plaza, and Enrique S. Quintana-Ortí

***Abstract*—Remotely sensed hyperspectral imaging missions are often limited by onboard power restrictions while, simultaneously, require high computing power in order to address applications with relevant constraints in terms of processing times. In recent years, graphics processing units (GPUs) have emerged as a commodity computing platform suitable to meet real-time processing requirements in hyperspectral image processing. On the other hand, GPUs are power-hungry devices, which result in the need to explore the tradeoff between the expected high performance and the significant power consumption of computing architectures suitable to perform fast processing of hyperspectral images. In this paper, we explore the balance between computing performance and power consumption of GPUs in the context of a popular hyperspectral imaging application, such as spectral unmixing. Specifically, we investigate several processing chains for spectral unmixing and evaluate them on three different GPUs, corresponding to the two latest generations of GPUs from NVIDIA ("Fermi" and "Kepler"), as well as an alternative low-power system more suitable for embedded appliances. Our paper provides some observations about the possibility to use GPUs as effective onboard devices in hyperspectral imaging applications.**

***Index Terms*—Energy consumption, graphics processing units (GPUs), hyperspectral imaging, high-performance computing.**

I. Introduction

HYPERSPECTRAL imaging missions collect a large number of images, corresponding to different wavelength channels, for the same area on the surface of the Earth [1]. Airborne instruments and satellites in operation typically feature a spatial resolution of a few dozens of meters and a revisit time between 3 and 16 days in case of satellite instruments. Combined with fine spectral resolution and extensive earth coverage, this results in vast amounts of data, justifying the adoption of high-performance computational resources for onboard remote sensing that can process this information in near real time in upcoming missions.

For illustrative purposes, Table I displays the spatial and spectral parameters of eight hyperspectral instruments: two airborne (HYDICE[1] and AVIRIS[2]) and six spaceborne (HYPERION,[3] EnMAP,[4] PRISMA,[5] CHRIS,[6] HyspIRI,[7] and IASI[8]). From this list, EnMAP, PRISMA, and HyspIRI are not yet operational. The spatial resolutions are higher for sensors carried by low-altitude platforms and vice versa. The spectral coverage of HYDICE, AVIRIS, HYPERION, EnMAP, PRISMA, and HyspIRI corresponds to the visible, the near-infrared, and the shortwave infrared spectral bands, whereas CHRIS covers the visible bands and IASI covers the midinfrared and the long-infrared bands. The number of bands is approximately 200 for HYDICE, AVIRIS, HYPERION, EnMAP, PRISMA, and HyspIRI, with a spectral resolution of the order of 10 nm. The number of bands for CHRIS is 63 with spectral resolutions of 1.3 and 12 nm (depending on the region of the spectrum) and 8461 for IASI with spectral resolution between 8 and 70 nm. In all cases, the spectral resolution is very high (offering a huge potential to discriminate materials). A summary of the characteristics of several hyperspectral imaging instruments currently in operation, under construction, and missions in a planning stage has been recently compiled [2].

Unfortunately, several factors make the analysis of hyperspectral data often a complex and hard task, calling for sophisticated methods and algorithms. Among these factors, we refer to spectral mixing effects that have been generally approached by identifying a set of spectrally pure signatures in the scene (called *endmembers* in unmixing terminology) and their corresponding abundance fractions in each (mixed) pixel of the scene [3]. An additional important issue is the extremely high dimensionality and size of the data, resulting from the improved spatial, spectral, and temporal resolutions provided by hyperspectral instruments. This demands fast computing solutions that can accelerate the interpretation and efficient exploitation of hyperspectral datasets in various applications [4]. For example, it has been estimated by the NASA's Jet Propulsion Laboratory (JPL) that a volume of 4.5 TB of data will be daily produced by HyspIRI (1630 TB per year), and similar data volume ratios are expected for EnMAP and PRISMA. Unfortunately, this extraordinary amount of information jeopardizes the use of latest-generation hyperspectral instruments in real-time or near real-time applications, due to the prohibitive delays in the delivery of earth observation payload data to ground processing facilities [5]. In this respect, ESA already flagged up in

Manuscript received September 20, 2013; revised February 24, 2014; accepted April 28, 2014. Date of publication June 22, 2014; date of current version August 01, 2014. This work was supported in part by projects CICYT TIN2011-23283 and AYA2011-29334-C02-02, and in part by FEDER.

S. Sánchez and A. Plaza are with the Hyperspectral Computing Laboratory (HyperComp), Department of Technology of Computers and Communications, University of Extremadura, 10071 Cáceres, Spain (e-mail: mitisnio@gmail.com; aplaza@unex.es).

G. León and E. S. Quintana-Ortí are with the Department of Computer Science and Engineering, University Jaume I, 12071 Castellón, Spain (e-mail: leon@uji.es; quintana@uji.es).

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

Digital Object Identifier 10.1109/JSTARS.2014.2322035

[1][Online]. Available: http://rsd-www.nrl.navy.mil/hydice

[2][Online]. Available: http://aviris.jpl.nasa.gov

[3][Online]. Available: http://eo1.usgs.gov

[4][Online]. Available: http://www.enmap.org

[5][Online]. Available: http://www.asi.it/en/flash_en/observing/prisma

[6][Online]. Available: https://earth.esa.int/web/guest/missions/esa-operational-eo-missions/proba

[7][Online]. Available: http://hyspiri.jpl.nasa.gov

[8][Online]. Available:http://smsc.cnes.fr/documentation/IASI/Publications/SPIE_ASPI.pdf

TABLE I
PARAMETERS OF EIGHT HYPERSPECTRAL INSTRUMENTS

Parameter	HYDICE	AVIRIS	HYPERION	EnMAP	PRISMA	CHRIS	HyspIRI	IASI
Altitude (km)	1.6	20	705	653	614	556	626	817
Spatial resolution (m)	0.75	20	30	30	5–30	36	60	V:1–2 km, H: 25 km
Spectral resolution (nm)	7–14	10	10	6.5–10	10	1.3–12	4–12	8–70
Coverage (μm)	0.4–2.5	0.4–2.5	0.4–2.5	0.4–2.5	0.4–2.5	0.4–1.0	0.38–2.5, 7.5–12	3.62–15.5
Number of bands	210	224	220	228	238	63	217	8461
Data cube size	200× 320× 210	512× 614× 224	660× 256× 220	1000× 1000× 228	400× 880× 238	748× 748× 63	620× 512× 210	765× 120× 8461

2011 that "data rates and data volumes produced by payloads continue to increase, while the available downlink bandwidth to ground stations is comparatively stable" [6]. In this context, the design of solutions aimed at taking advantage of the ever increasing dimensionality of remotely sensed hyperspectral images for onboard and near real-time applications has gained significant relevance and momentum during the last decade [7], [8].

In recent years, graphics processing units (GPUs) have evolved into highly parallel, multithreaded, many-core coprocessors with tremendous computational power, consumption, and memory bandwidth [9]. The combined features of general-purpose supercomputing, high parallelism, high memory bandwidth, low cost, compact size, and excellent programmability are now making GPU-based desktop computers an appealing alternative to massively parallel systems made up of commodity CPUs. The exploding GPU capability has attracted more and more scientists and engineers to use it as a cost-effective, high-performance computing platform, including scientists in hyperspectral processing areas. In addition, GPUs can also significantly increase the computational power of cluster-based and distributed systems (indeed, a significant number of the fastest supercomputers in the world are now clusters of GPUs[9]).

Several efforts exploiting GPU technology can already be found in the hyperspectral imaging literature [7], [8], [10]. For instance, only in the area of spectral unmixing, there have been many recent developments. A GPU-based implementation of an automated morphological endmember extraction (AMEE) algorithm for pure spectral signature (endmember) identification is described in [11]. In this case, speedups on the order of $15\times$ were reported. A full spectral unmixing chain comprising the automatic estimation of the number of endmembers, the identification of the endmember signatures, and quantification of endmember fractional abundances has been reported in [12], with speedups superior to $50\times$. Additional efforts toward real-time and onboard hyperspectral target detection and classification using GPUs have also been recently available [13], [14]. It should be noted that, despite the increasing programmability of low-power GPUs such as those available in smartphones, radiation-tolerance, and power consumption issues still prevent the full incorporation of GPUs to spaceborne earth observation missions.

In [15], we analyzed the performance-energy tradeoff of two spectral unmixing methods, for identifying the endmembers and estimating their fractional abundances in hyperspectral images, on a wide variety of multicore architectures, from a low-power digital signal processor (DSP) to conventional, general-purpose multicore processors. However, a detailed assessment of the tradeoff between the (high) computational performance provided by GPUs and their (also high) power consumption remains unexplored and unquantified in the literature. In this paper, we further investigate the performance-energy balance of current high-performance architectures that can meet onboard processing restrictions. Specifically, in this paper, we present several complete processing chains for spectral unmixing, and evaluate them on three different GPUs, corresponding to the two latest generations of graphics processors from NVIDIA ("Fermi" and "Kepler") for the high-end segment, as well as an alternative low-power system more suitable for embedded appliances.

The remainder of the paper is organized as follows. Section II describes the different modules that have been used to design the hyperspectral unmixing chains considered in this paper. Section III describes their parallel implementation in GPU architectures. Section IV presents experimental results in terms of both performance and energy consumption for the considered unmixing chains, using various GPU architectures and a set of representative hyperspectral scenes. Section V concludes the paper with some remarks and hints at plausible future research lines.

II. HYPERSPECTRAL UNMIXING CHAINS

This section describes the hyperspectral unmixing chains that we have considered for our experiments. The chains consist of four main steps: 1) identification of the number of endmembers; 2) dimensionality reduction; 3) endmember extraction; and 4) abundance estimation. In all steps, we have considered two possible approaches except in step 3), which is performed using the N-FINDR algorithm [16]. For step 1), we consider virtual dimensionality (VD) [17] and hyperspectral signal identification by minimum error (HYSIME) [18]. For step 2), we consider the well-known principal component analysis (PCA) and a simultaneous PCA (SPCA) implementation [19]. Finally, for step 4), we consider an unsupervised least squares (ULS) [20] approach to abundance estimation and a nonnegatively constrained abundance estimation algorithm implemented by the image space reconstruction algorithm (ISRA) [21]. These approaches can be combined to form several unmixing chains as illustrated in Fig. 1. In the following, we briefly outline different methods that we have used to construct the aforementioned chains.

A. Estimation of the Number of Endmembers

1) VD Method: The VD concept follows the "pigeon-hole principle." If we represent a signal source by a pigeon and a spectral band by a hole, we can use a spectral band to accommodate one source. Thus, if a distinct spectral signature makes a contribution to the eigenvalue-represented signal energy

[9][Online]. Available: http://www.top500.org

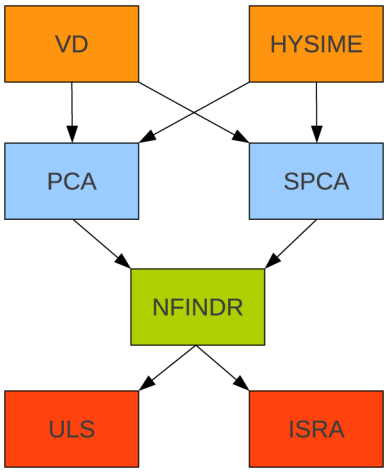

Fig. 1. Spectral unmixing chains considered in this paper.

in one spectral band, then its associated correlation eigenvalue will be greater than its corresponding covariance eigenvalue in this particular band. Otherwise, the correlation eigenvalue would be very close to the covariance eigenvalue, in which case only noise would be present in this particular band. By applying this concept, a Neyman–Pearson detector [17] is introduced to formulate the issue of whether a distinct signature is present or not in each of the spectral bands of the original image as a binary hypothesis testing problem. Here, the decision is made based on an input parameter of the algorithm which is called the false alarm probability, or P_F, which is used to establish the sensitivity of the algorithm in terms of how much error can be tolerated in identification of the actual number of endmembers in the image data. With this interpretation in mind, the issue of determining an estimate for the number of endmembers p is further simplified and reduced to a specific value of P_F that is preset by the Neyman–Pearson detector.

2) HYSIME Method: The HYSIME method consists of two parts: 1) an estimation of the noise present in the original hyperspectral image is obtained and 2) an approach which addresses high correlation exhibited by close spectral bands. The main advantage of this step is that the computational complexity is substantially lower than that of other algorithms for noise estimation in hyperspectral data in the literature. Additional details about this step can be found in [22], but we do not repeat them here for space considerations. A signal subspace identification procedure follows which first computes the noise correlation matrix and then computes the signal correlation matrix. Then, the eigenvectors of the signal correlation matrix are obtained and sorted in ascending order. Finally, a minimization function is applied to obtain an estimate of the number of endmembers p.

B. Dimensionality Reduction Using PCA and SPCA

In this step, the dimensionality of the data is reduced from n to $p-1$, where p is the number of endmembers estimated by VD or HYSIME in Section II-A. For this purpose, we use PCA [19], which orthogonally projects the data into a new coordinate system, defined by the variance of the original data. Thus, the direction that accounts for the greatest variance of the original data will be the first coordinate (the principal component) of the transformed system, the second dimension will be the direction with the second largest variance, and so on. The eigenvalues in the transformed system encase the "weight" of each principal component on the resulting data. By choosing only the eigenvectors corresponding to the largest $p-1$ eigenvalues, the dimensionality of the data is reduced while preserving the maximum information (variance). Simultaneous iteration [23] simply consists of applying the power iteration algorithm to several eigenvectors simultaneously. There are several algorithms for computing the eigendecomposition. The power iteration algorithm [19] is a well known procedure for computing the largest eigenpair of a matrix. Simultaneous iteration is the basis of SPCA and simply consists of applying the power iteration algorithm to several eigenvectors simultaneously. While simultaneous iteration (and power iteration) might not be the most computationally efficient eigendecomposition algorithm available, it is very regular and exhibits ample data-parallelism, which improves its potential for parallelization in a massively parallel architecture such as a GPU.

C. Endmember Extraction Using NFINDR

The NFINDR algorithm [24] is one of the most widely used and successfully applied methods for automatically determining endmembers in hyperspectral image data without using *a priori* information. This algorithm looks for the set of pixels with the largest possible volume by *inflating* a simplex inside the data. The procedure begins with a random initial selection of pixels. Every pixel in the image must be evaluated in order to refine the estimate of endmembers, looking for the set of pixels that maximizes the volume of the simplex defined by selected endmembers. The mathematical definition of the volume of a simplex formed by a set of endmember candidates is proportional to the determinant of the set augmented by a row of ones. The determinant is only defined in case the number of features is $p-1$, p being the number of desired endmembers [25]. Since in hyperspectral data typically $n \gg p$, a transformation that reduces the dimensionality of the input data is required. In this paper, we use the PCA and the SPCA (described in the previous section) for this purpose. The corresponding volume is calculated for every pixel in each endmember position by replacing that endmember and finding the resulting volume. If the replacement results in an increase of volume, the pixel replaces the endmember. This procedure is repeated in iterative fashion until no more endmember replacements occur.

D. Abundance Estimation Using ULS and ISRA

Once a set of endmembers has been estimated using the NFINDR algorithm, an unconstrained p-dimensional estimate of the endmember abundances in a given pixel can be simply obtained by using unconstrained least squares minimization [20]. The main advantages of ULS abundance estimation approach are the simplicity of its implementation and its fast execution. However, under this unconstrained model, the derivation of negative abundances is possible if the model endmembers are not pure or if they are affected by variability caused by spatial or temporal variations [25]. To address this issue, two physical

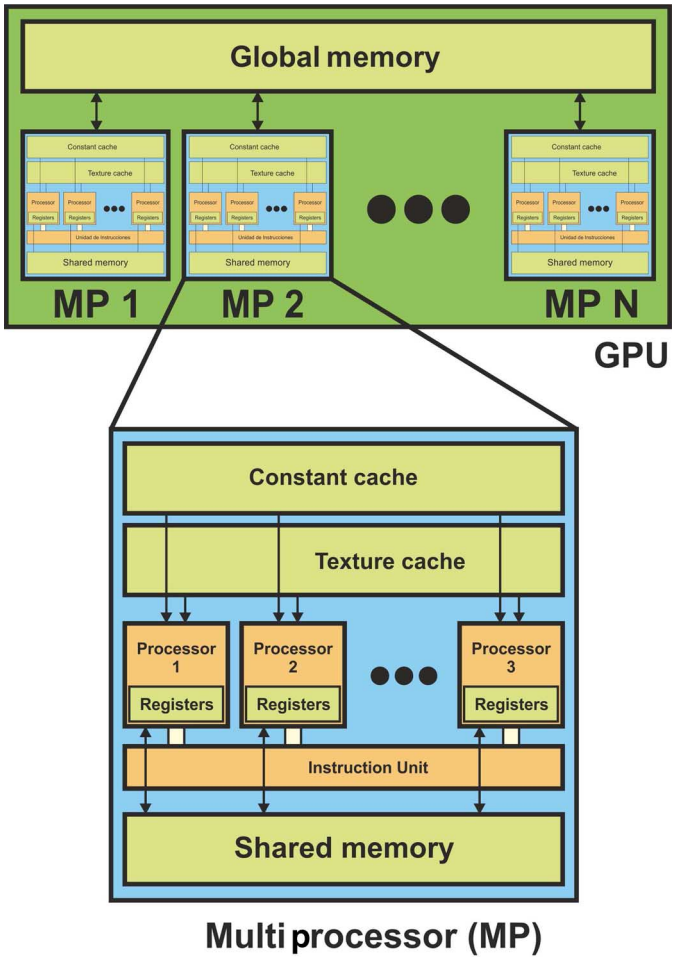

Fig. 2. Schematic overview of a GPU architecture.

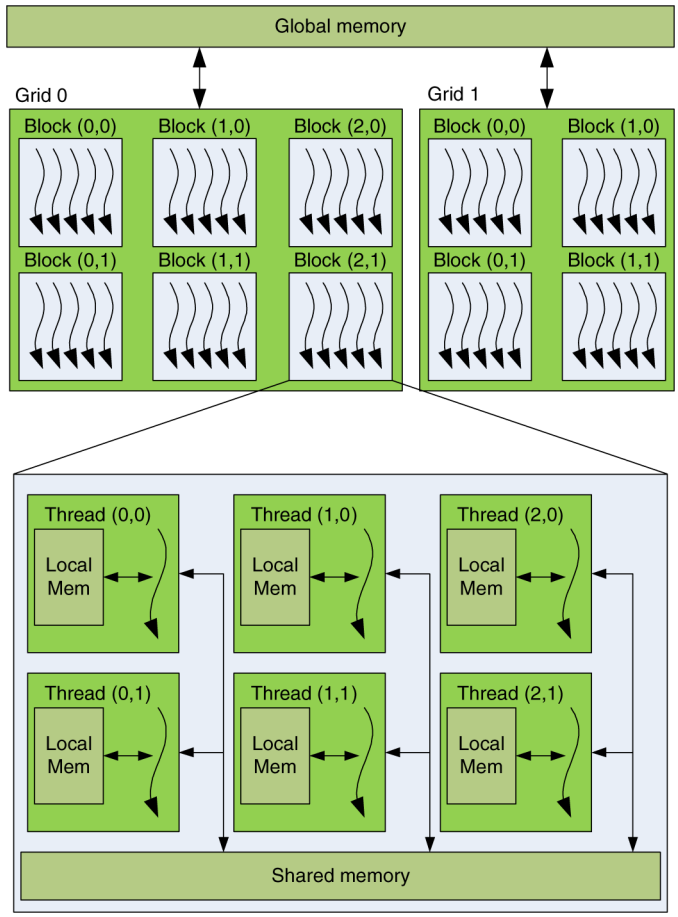

Fig. 3. GPU programming model. Here, the term *Local Mem* refers to *registers*.

constrains can be imposed into the model: 1) the abundance nonnegativity constraint (ANC) and 2) the abundance sum-to-one constraint (ASC). As indicated in [26], a fully constrained estimate can be obtained in least-squares sense. While partially constrained solutions imposing only the ANC have found success in the literature [27], the ASC is nevertheless prone to criticisms because, in a real image, there is a strong signature variability [28] that, at the very least, introduces positive scaling factors varying from pixel to pixel in the signatures present in the mixtures. What we conclude is that the nonnegativity of the endmembers automatically imposes a generalized ASC. For this reason, in this paper, we focus on solutions that do not explicitly impose the ASC constraint but only the ANC constraint. A successful approach for this purpose in different applications is ISRA [21], a multiplicative algorithm for solving ANC problems.

III. Parallelization on GPUs

In the following, we describe the GPU implementations of the methods described in the previous section. The GPU implementations have been carried out using the compute unified device architecture (CUDA), introduced by NVIDIA,[10] and also the NVIDIA CUDA Basic Linear Algebra Subroutines (`cuBLAS`).[11]

[10][Online]. Available: https://developer.nvidia.com/cuda

[11][Online]. Available: https://developer.nvidia.com/cublas

Fig. 2 shows the architecture of a GPU, which can be regarded as a set of multiprocessors (MPs). Each MP is characterized by a single instruction multiple data (SIMD) architecture, i.e., at each clock cycle, processors of an MP execute the same instruction but operating on different data. Each processor has access to a local shared memory and also to local cache memories in the MP, while the MPs have access to the global GPU (device) memory. Unsurprisingly, the programming model for these devices is similar to the architecture lying underneath (see Fig. 3). GPUs can be abstracted in terms of a *stream model*, under which all datasets are represented as streams (i.e., ordered datasets). Algorithms are constructed by chaining the so-called *kernels*, which operate on entire streams and which are executed by a MP, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. The kernels can perform a kind of batch processing arranged in the form of a grid of blocks, where each block is composed by a group of threads that share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. As a result, there are different levels of memory in the GPU for the thread, block, and grid concepts. While the number of threads that can run in a single block is limited, the number of threads that can be concurrently executed is much larger as several blocks can be

executed in parallel. This comes at the expense of reducing the cooperation between threads, since threads in different blocks cannot synchronize among themselves.

In the following, we outline the performance of the different modules used for constructing the unmixing chains discussed in this paper. Since their GPU implementations have been presented in previous contributions, we do not present them in detail here. Additional details about the GPU implementation of VD and HYSIME can be found in [29]. Similarly, details about the GPU implementations of PCA, SPCA, NFINDR, and ULS can be found in [30]. Finally, details about the GPU implementation of ISRA are given in [31]. At this point, it is important to emphasize that faster GPU alternatives exist in the literature for some parts of the unmixing chain. For instance, the GPU implementation of maximum simplex volume analysis (MSVA) [32] provides faster performance in endmember identification than the discussed GPU implementation of NFINDR. Most importantly, MSVA does not require a prior dimensionality reduction such as PCA or SPCA which is needed by our implementation of NFINDR, and this could lead to reduced consumption analysis. In turn, NFINDR is a widely used algorithm for endmember identification purposes, as indicated by the fact that other highly efficient GPU implementations for this algorithm exist in the literature [33]. Although these alternative GPU implementations of MSVA and NFINDR were not available to us at the time of experiments, future works will be focused on testing other possible alternatives for the modules discussed below, which represent widely used choices in the hyperspectral unmixing community. For instance, the vertex component analysis (VCA) algorithm [34] has been recently implemented using a hybrid CPU/GPU approach [35] and could serve as an efficient low-power alternative to NFINDR as well.

A. Estimation of the Number of Endmembers

1) VD Method: This algorithm [17] defines the number of endmembers as signal sources which are determined based on their distinct spectral properties. Once we load the full hyperspectral image $\mathbf{Y}$ (in pixel-by-pixel fashion, being m the number of pixels and l the number of bands) from disk to the main memory of the GPU, the first step is to calculate the covariance matrix $\mathbf{K}_{l\times l}$. For this purpose, we need to calculate the mean value $\overline{\mathbf{Y}}$ of each band of the image and subtract this figure to all the pixels in the same band. To perform this calculation in the GPU, we use a kernel called *meanpixel*, configured with as many blocks as the number of bands l in the hyperspectral image. In each block, all available threads perform a reduction process using shared memory and coalesced memory accesses to add the values of all the pixels in the same band. Once this process is completed, a second thread divides the computed value by the number of pixels in the original image m and the mean value is obtained. The resulting mean values of each band $\overline{\mathbf{Y}}$ are stored in a structure as they will be needed for the calculation of the covariance matrix $\mathbf{K}_{l\times l}$ in the GPU by means of a matrix multiplication operation $(\mathbf{Y}-\overline{\mathbf{Y}})^T(\mathbf{Y}-\overline{\mathbf{Y}})$. This operation is performed using the `cuBLAS` library, specifically, we use the `cublasSgemm` function. The next step is to calculate the correlation matrix $\mathbf{R}_{l\times l}$ in the GPU. To obtain this, we apply a kernel called `correlation`, which launches as many threads as elements in $\mathbf{R}_{l\times l}$, where each thread computes an element of the resulting matrix as follows: $\mathbf{R}_{ij}=\mathbf{K}_{ij}+\overline{\mathbf{Y}}_i\overline{\mathbf{Y}}_j$. Finally, we observe that the remaining steps in the VD calculation (i.e., extraction of correlation-eigenvalues, covariance-eigenvalues, and Neyman–Pearson test for estimation of the number of endmembers p [17]) can be computed very fast in the CPU.

2) HYSIME Method: The HYSIME [22] algorithm is split into two steps: 1) the algorithm removes the noise in the original hyperspectral image and 2) the algorithm estimates the subspace in which the hyperspectral data resides. Once we load the hyperspectral image $\mathbf{Y}$ in GPU memory, we first implement the noise estimation algorithm. For this purpose, the first step is to compute $\hat{\mathbf{R}}$ and its inverse $\mathbf{R}'$. The former is calculated in the GPU by means of standard `cuBLAS` matrix multiplication (using `cublasSgemm`), whereas the latter is implemented in the CPU to avoid the high computational cost of the inverse operation in parallel. The main loop of the algorithm, which has as many iterations as spectral bands are present in the original image, calculates the noise estimation matrix using the two aforementioned matrices. Then, we implement the signal subspace estimation as a second part of the algorithm. The first step in this task is the calculation of the noise correlation matrix $\hat{\mathbf{R}}_n$, which in principle could be obtained from a standard matrix multiplication in `cuBLAS`. However, in the next steps of the algorithm, we only consider the diagonal elements of this matrix, and therefore, it is not necessary to compute the rest of the values. Hence, we use l threads to ensure coalesced accesses during the computation of an element of the diagonal of $\hat{\mathbf{R}}_n$. The second step of the subspace estimation algorithm is to calculate the signal correlation matrix in the GPU. These two matrices are used, via a minimization function, to get the number of different endmember materials p in the CPU.

B. Dimensionality Reduction Using PCA and SPCA

Our GPU version of the PCA transform yields a reduction in the dimensionality of the image, from n spectral bands to $p-1$ ($n \gg p-1$), while retaining the necessary information for conducting the spectral unmixing process. First, we load the hyperspectral image $\mathbf{Y}$ in the main memory of the GPU, where it is centered and normalized to standard deviation of one, to improve the stability of the subsequent computations. Then, we use the sample covariance matrix $\mathbf{K}_{l\times l}$ of the normalized image $\mathbf{Y}$ in the GPU by resorting to the `cuBLAS` library. Specifically, we use the matrix multiplication operation `cublasSgemm` for this purpose. At this point, the implementation depends on whether we aim for the classic PCA or the SPCA. If the classic PCA is applied, we perform a singular value decomposition (SVD) in the CPU and then send the obtained eigenvectors to the GPU in order to perform the final projection and obtain the transformed $p-1$ dimensional data. Quite opposite, if the SPCA is applied, we apply the power iteration algorithm in the GPU and analyze convergence by checking if the direction of eigenvectors did not significantly change from the previous iteration. This operation is very simple and can be calculated in the CPU. If the measured

changes are above a given tolerance threshold, we continue iterating; otherwise, we finalize and provide the final eigenvectors, which are used to obtain the transformed $p-1$ dimensional data.

C. Endmember Extraction Using NFINDR

The NFINDR [24] algorithm looks for the set of pixels with the largest possible volume by *inflating* a simplex inside the data. Prior to implementation on GPU, a set of optimizations were performed. In particular, the most time-consuming computation in the NFINDR algorithm is the calculation of determinants. In particular, the determinant of a nonsingular matrix $\mathbf{V}$ is usually obtained from the factorization $\mathbf{PV}=\mathbf{LU}$ (where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is a unit lower triangular matrix, and $\mathbf{U}$ is an upper triangular matrix) as the product of the diagonal elements of $\mathbf{U}$. This decomposition is known as *Gaussian elimination* or LU factorization (with partial row pivoting) and its computational cost is cubic with respect to the matrix dimension. The repeated volume calculations of the NFINDR algorithm can be reduced by exploiting some basic properties of the LU factorization and matrix determinants. Consider, i.e., the $p\times p$ and $p\times p-1$ matrices

$$V_{\mathbf{M}}^{(1)}=\begin{bmatrix}1 & \dots & 1 & 1\\ \mathbf{e}_2^{(0)} & \dots & \mathbf{e}_p^{(0)} & \mathbf{x}_j\end{bmatrix} \text{ and}$$
$$\bar{V}_{\mathbf{M}}^{(1)}=\begin{bmatrix}1 & \dots & 1\\ \mathbf{e}_2^{(0)} & \dots & \mathbf{e}_p^{(0)}\end{bmatrix} \quad (1)$$

where $\mathbf{M}$ is the reduced version of the hyperspectral image with p components, obtained from the PCA or SPCA described in the previous section. Assume that we have computed the LU factorization (with partial pivoting) $\mathbf{P}_{\mathbf{M}}\bar{\mathbf{V}}_{\mathbf{M}}^{(1)}=\mathbf{L}_{\mathbf{M}}\mathbf{U}_{\mathbf{M}}$. Then, the LU factorization (with partial pivoting) of $\mathbf{V}_{\mathbf{M}}^{(1)}$ is simply given by $\mathbf{P}_{\mathbf{M}}\mathbf{V}_{\mathbf{M}}^{(1)}=[\mathbf{U}_{\mathbf{M}}(\mathbf{L}_{\mathbf{M}}^{-1}\mathbf{P}_{\mathbf{M}}^{\mathbf{T}}\mathbf{x}_{\mathbf{j}})]$. Therefore, the LU factorizations required in the volume calculations of the NFINDR algorithm can be all computed by simply forming the $p\times m$ matrix $\hat{\mathbf{M}}=\begin{bmatrix}\mathbf{1} & \mathbf{1} & \dots & \mathbf{1}\\ & & \mathbf{M}^T & \end{bmatrix}$, where $\mathbf{M}^{\mathbf{T}}=\tilde{\mathbf{M}}^{\mathbf{T}}V$, $\tilde{\mathbf{M}}^{\mathbf{T}}=(\mathbf{M}^{\mathbf{T}}-mean(\mathbf{M}^{\mathbf{T}}))/\sqrt{m-1}$, and m denotes the total number of pixels in the hyperspectral image. Then, we need to compute $\mathbf{L}_{\mathbf{M}}^{-1}\mathbf{P}_{\mathbf{M}}^{\mathbf{T}}\hat{\mathbf{M}}$. This is one of the parts that we accomplished in the GPU by means of a `Volume-Calculation` kernel which obtains the volume of each pixel for one iteration. The m volumes required in the first iteration of the NFINDR algorithm are obtained from the product of the determinant of $\mathbf{U}_{\mathbf{M}}$ times each one of the entries in the last row of $\mathbf{L}_{\mathbf{M}}^{-1}\mathbf{P}_{\mathbf{M}}^{\mathbf{T}}\hat{\mathbf{M}}$. By means of a `ReductionVol` kernel, we get the value of the maximum volume and the coordinates of the pixel that produce such volume. Given that $m\gg p$, this implies a significant reduction of the computational complexity of the original algorithm. In the following, we describe step-by-step how the algorithm has been implemented in GPU.

1) Initialization: First, we form the matrix $\mathbf{V}_{\mathbf{M}}^{-(1)}$ of size $p\times p$ by initializing to ones the first row and setting in each column (from row two) a randomly selected endmember. The determinant of the resulting matrix $\mathbf{V}_{\mathbf{B}}^{-(1)}$ is now calculated and the result is stored in the variable `currentVolume`. Since the dimensions of this matrix are small, the determinant computation can be performed in the CPU.

2) Volume Calculation: Next, we form a vector (of size m) `Volumes`, where at each iteration k, the volume resulting from the replacement of the pixel i with and endmember will be stored. Also, the reduced image $\mathbf{M}$ is modified by adding a first band of ones to produce $\hat{\mathbf{M}}$. At each iteration k, we replace in $\mathbf{V}_{\mathbf{M}}^{-(1)}$ the endmember in position k by the endmember in position p; we also replace the column p by a column of the type $[0,0,\dots,1]^T$. Then, the LU factorization is applied to this matrix and $\mathbf{L}_{\mathbf{M}}$,$\mathbf{U}_{\mathbf{M}}$, and $\mathbf{P}_{\mathbf{M}}$ are obtained. After that, we compute the determinant of $\mathbf{U}_{\mathbf{M}}$ and invert $\mathbf{L}_{\mathbf{M}}$. Due to the fact that the aforementioned matrices are triangular and small, the determinant and the inverse can be computed in CPU without penalty to the total execution time. At this point, we have the elements necessary to calculate the achieved volumes in one iteration. Note that these elements are computed by multiplying the determinant of $\mathbf{U}_{\mathbf{M}}$ by all the entries in the last row of $\mathbf{L}_{\mathbf{M}}^{-1}\mathbf{P}_{\mathbf{M}}^{\mathbf{T}}\hat{\mathbf{M}}$. We divide these calculation into two phases: in the first phase, we perform the matrix multiplication $\mathbf{S}=\mathbf{L}_{\mathbf{M}}^{-1}\mathbf{P}_{\mathbf{M}}^{\mathbf{T}}\hat{\mathbf{M}}$ in the CPU. The second phase is more computationally expensive and is performed in GPU using the kernel `VolumeCalculation`. The volumes can be obtained by multiplying the determinant of $\mathbf{U}_{\mathbf{M}}$ with the elements of the last row of $\mathbf{L}_{\mathbf{M}}^{-1}\mathbf{P}_{\mathbf{M}}^{\mathbf{T}}\hat{\mathbf{M}}$; that is, by multiplying $\mathbf{U}_{\mathbf{M}}$ by the elements of the last row of $\mathbf{S}\hat{\mathbf{M}}$. The result of this operation is a $p\times m$ matrix, which only needs the elements in the last row. Because of that we can save the calculations to get the first $p-1$ rows by multiplying only the last row of $\mathbf{S}$ by the matrix $\hat{\mathbf{M}}$.

3) Replacement: Once we have computed the volumes for one iteration, the next step is to find the pixel that generated the biggest volume and check if this volume is bigger than `currentVolume`. For this task, we use the kernel `ReductionVol`, which performs a reduction process in which each block works with a section of volume data and extracts the local maximum and the position of this local maximum. Since each block achieves a different value, at the end of the execution we will have as many values as blocks (each value is the local maximum of its section), and it will be necessary to store these values in a structure, together with their positions in global memory. Then, these values will be copied to the main memory of the CPU in order to be reduced again and thus determine the global maximum and its position.

D. Abundance Estimation Using ULS and ISRA

Our GPU implementation of ULS can be summarized in the following two main steps. The first one is to calculate the operation $(\mathbf{M}^T\mathbf{M})$, where $\mathbf{M}=\{\mathbf{e}_i\}_{i=1}^p$ is formed by the p endmembers extracted by NFINDR. The inverse of this operation is calculated in the CPU mainly due to two reasons: 1) its computation is relatively fast and 2) the inverse operation remains the same throughout the whole execution of the code. The result obtained in the previous step is now multiplied by the each pixel $\mathbf{y}$ in the hyperspectral image $\mathbf{Y}$, thus obtaining a set of abundance vectors $\alpha=[\alpha_1,\alpha_2,\dots,\alpha_p]$, each containing the fractional abundances of the p endmembers in $\mathbf{M}$. This is

accomplished in the GPU by means of a specific kernel, which produces p abundance maps. On the other hand, the GPU implementation of ISRA follows an iterative approach which makes use of two kernels. The first one is called `initialize`, and is invoked only once. This kernel uses as many threads as pixels m in the original hyperspectral image, and simply initializes the abundance estimations at each pixel to an ULS estimate. The second kernel is called `update`, and is repeatedly invoked, until convergence. This kernel also uses as many threads as pixels m in the original hyperspectral image $\mathbf{Y}$, and performs partially constrained abundance estimation for every pixel in parallel, using the procedure described in [21].

IV. Experimental Results

We open this section with a description of the hardware setup (i.e., computational resources and power measurement device) as well as the hyperspectral data testbeds that were employed in the experimental study. The analysis of the performance-energy tradeoff of current hardware accelerators when applied to process all stages of the two complete spectral unmixing chains, follows next.

A. Hardware Configurations

The experimental study was performed on three different platforms, equipped with recent graphics technology from NVIDIA and the state-of-the-art multicore processors from Intel or ARM.

1) Carma: An NVIDIA Quadro 1000M (Q1000M) GPU connected to an ARM Cortex A9 multicore processor (4 cores at 1.3 GHz) with 2 GB of DDR3L RAM.
2) Fermi: An NVIDIA GeForce GTX 480 ("Fermi") graphics card connected to a single Intel Xeon i7-3770K ("Ivy Bridge") processor (4 cores at 3.5 GHz) and 16 GB of DDR3 RAM.
3) Kepler: An NVIDIA Tesla K20c ("Kepler") graphics card connected to a single Intel Xeon i7-3930K ("Sandy Bridge E") processor (6 cores at 3.2 GHz) and 24 GB of DDR3 RAM.

These three platforms represent two extremes of the spectrum in hardware acceleration using graphics processors. On the one side, the NVIDIA Q1000M is a 96-core GPU with 2 GB of DDR3 RAM, integrated into a low-power board (Carma development kit) together with a general-purpose processor from ARM. This system has no disk or any other relevant devices attached to it, drawing a mere 12.5 W (on average) when idle, i.e., when doing nothing.

On the other side, Fermi and Kepler correspond the last two generations of high-throughput accelerators from NVIDIA, with both systems integrating a graphics processor (Tesla T10 in the Tesla C1060 and GK110 in the Tesla K20) with a high number of cores (240 and 2496, respectively) and a considerable amount of fast GDDR5 RAM (4 and 5 GB, respectively). To operate, these boards have to be attached via PCI-e to a server with at least one general-purpose processor in charge of controlling the GPU, playing the same role as the ARM A9 processor in the `Carma` system. However, these are regular servers that, in general, contain one or more disks and Ethernet ports (the latter embedded into the mainboard). The net result is a much higher idle power consumption: on average, 97.9 and 102.6 W, respectively, for the platforms where the Fermi and Kepler boards are attached.

Tuned implementations of the numerical linear algebra operations that appear in the algorithms were obtained from recent releases of Intel MKL (version 10.3.9 for the two Intel processors) and NVIDIA CUBLAS (version 4.2.9 for all three GPUs). As many of the operations of the algorithms are cast in terms of the numerical kernels available in these highly tuned libraries, the operating system and compiler that was employed in these cases has a negligible contribution. On the other hand, currently there exist no tuned implementation of analogous kernels for the ARM, so that we had to rely, in this case, in the legacy implementation of these routines available at netlib,[12] compiled with GNU `gcc` (version 4.5.2) and the `-O3` flag.

In order to measure power, we employed a *WattsUp? Pro. Net* wattmeter. This device is plugged to the cable that connects the electrical socket to the power supply unit (PSU), and reports external AC for the full platform, with a sampling rate of 1 Hz, an accuracy of $\pm 1\%$ and a resolution of 0.1 W. We warmed up the platforms by executing each stage of the chains repeatedly during 3 min before the sampling was initiated for that particular stage. Power measures were then continuously recorded while the test (i.e., the stage of the chain) was ran during 3 additional minutes, and power was averaged over this period and multiplied by the execution time of a single execution of the stage to obtain its (total) energy consumption. A complementary metric in order to compare the energy efficiency of the different platforms is the net energy consumption, which was obtained by subtracting the product of idle power and time from the energy consumption. This measure better reflects the energy necessary to perform the work, cancelling the effect of unnecesary components (e.g., the disk) on the power draw. Hereafter, execution time is reported in seconds (s), power in Watts (W), and energy in Joules ($\mathrm{J} = \mathrm{W} \cdot \mathrm{s}$).

B. Hyperspectral Scenes

We leveraged the same two datasets employed for the evaluation in [15], so that we can later compare the performance-energy ratios of the GPU-accelerated systems to those of the multicore architectures that were analyzed in that paper. The first case corresponds to the online[13] Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) `Cuprite` scenario, and is a well-known benchmark for the evaluation of spectral unmixing methods. This specific image corresponds to a 350×350-pixel subset of sector `f970619t01p02_02_sc03.a.rfl`, which features 188 spectral bands with wavelengths between 0.4 and 2.5 μm, and requires an execution time below 2.98 s for real-time processing.

The second image was collected over the World Trade Center (`WTC`) after the attacks on September 11, 2011. It consists of 512×614 pixels and 224 bands with wavelengths between 0.4 and 2.5 μm, which corresponds to the standard data cube size recorded by AVIRIS, as indicated in Table I. Compared with the

[12][Online]. Available: http://www.netlib/org

[13][Online]. Available: http://aviris.jpl.nasa.gov/freedata

previous scenario, processing this image in real time requires a longer time, 5.09 s in this case.

C. Performance-Power-Energy Analysis the Chains

In order to simplify the analysis, we present results only for two of the eight chains that can be formed by different combinations of the methods presented in Fig. 1. The first tested chain (hereinafter, chain #1) consists of the combination VD+PCA+NFINDR+ULS, whereas the second tested chain (hereinafter, chain #2) consists of the combination HYSIME+SPCA+NFINDR+ISRA. It should be noted that these two chains only have NFINDR in common. Although we will discuss results for these two complete chains only, the individual processing module of the chains are completely interchangeable in accordance with the structure of Fig. 1, and the reader can derive conclusions for any of the other possible chains that can be possibly formed, since we report performance results for every processing module in such figure. In this exercise though, it should be noted that VD and HYSIME may estimate a different number of endmembers, which affects the complexity of the remaining modules of the chain. Table II summarizes the results for the application of the two 4-stage chains to the Cuprite and WTC images, on the three GPU systems, using five different metrics: 1) execution time; 2) average power; 3) maximum power; 4) (total) energy; and 5) net energy. For illustrative purposes, Table II also reports the execution times achieved by a serial (single-threaded) implementation of the same processing chains in each case.

Let us analyze the execution time first. A comparison between the two unmixing chains reveals that chain #2 is considerably more costly than chain #1, with an execution time between 4.1 and $4.8\times$ higher except for the analysis of scenario WTC on Carma, where the ratio is $2.6\times$, also in favor of chain #1. These differences are basically due to the algorithms employed in the first stage (VD for chain #1 and HYSIME for chain #2) and the last stage (ULS for chain #1 and ISRA for chain #2), while the remaining two stages show relative variations that are either small or have no impact on the global execution time. We note that the discrepancies between the run time of algorithm ISRA in the two chains are explained by the different number of endmembers determined by previous stages of the chain. Finally, it is important to emphasize that the serial times reported for some modules of the chain (particularly, NFINDR and ULS) are higher than the times measured in the Carma GPU. This is due to the fact that the Carma is a very low power GPU. However, when considering the full chain, the total processing times measured in the Carma GPU are always at least $3\times$ faster than the serial implementation and $10\times$ faster in the best case.

From the point of view of raw performance, it is also clear the advantage of Kepler over Carma, with the former platform being about $11-12\times$ faster in three of the cases and more than $20\times$ faster when chain #1 is applied to WTC, which is well aligned with the large gap in the number of cores between these two architectures (2048 for Kepler for a mere 96 for Carma). On the other hand, while there exists also a relevant distance between the core counts of Kepler and Fermi (only 240 in the latter), the performance advantage of the former is actually much smaller, a ratio at most of $1.53\times$ (chain #2 and WTC). The sources for this are two: 1) for these reduced-size scenarios, the GPU algorithms do not seem to scale far beyond the core number of Fermi and 2) for some stages, the memory bandwidth and the performance of the attached multicore—factors where Kepler is equal or even inferior to Fermi—play an important role.

The analysis from the perspective of power, a crucial figure for power-constrained systems, is quite different. In general, chain #2 exhibits higher average and maximum power rates than chain #1. More interestingly, the clear winner is now the Carma platform, with an average power in the range 22–26.6 W and a highest maximum power of 48.2 W (chain #2 and WTC). Compare these figures with Fermi and Kepler, which feature average powers that vary between 224.7 and 274.4 W, and a highest power rate of 358.3 W for the former and 287.0 W for the latter. While these numbers reveal a remarkable progress in power dissipation for Kepler compared with Fermi, which scales the number of cores by a factor of 8.5 while also increasing the amount of RAM, the outcome is still far from the low-power Carma.

Finally, consider next the effect of the above on energy, which combines performance (i.e., execution time) and power in a single figure of merit that favors architectures that are fast while drawing little power. Note also the difference between (total) energy and net energy, where the second cancels, to certain extent, the contribution of external factors (e.g., disk, ethernet interfaces, etc.) but also internal ones (e.g., leakage), in an attempt to account only for the actual energy that is necessary to perform the task. From the point of view of total energy, now the best platform is Kepler for all four combinations of chains and images, but the distance strongly depends on the case. Thus, for the small Cuprite scenario and the low-intensity chain #1, the differences are small, e.g., around 20% between Kepler and Carma, whereas in the opposite case (chain #2 and scenario WTC), we observe a larger gap, close to 40% between the same two architectures. On the other hand, Carma is the most efficient solution from the perspective of net energy usage for image Cuprite, but it is overcome by Kepler for the larger scenario.

D. Comparative Study Against Conventional Multicore Architectures

In [15], we carried out a study of the performance-energy balance of five different multicore processors using two efficient spectral unmixing methods for identifying the endmembers and estimating their fractional abundances in hyperspectral images: the orthogonal subspace projection (OSP), implemented via Gram–Schmidt [36], and ISRA. The OSP in our multicore implementations was used for identifying the signatures of the endmembers (i.e., as an endmember identification method alternative to N-FINDR), but not their number. As a result, methods for estimating the number of endmembers such as VD or HYSIME are still required in order to estimate the number of endmembers. Unfortunately we have not yet developed multicore versions of VD and HYSIME (only GPU versions), hence the unmixing chain based on the OSP has the disadvantage that the number of endmembers should be provided by the user as an input parameter.

TABLE II
EXECUTION TIME AND POWER-ENERGY PERFORMANCE OF THE COMPLETE SPECTRAL UNMIXING CHAINS APPLIED TO THE `Cuprite` AND `WTC` SCENARIOS (TOP AND BOTTOM, RESPECTIVELY)

`Cuprite`

System	Chain #1					Chain #2				
	VD	PCA	NFINDR	ULS	4 stages	HYSIME	SPCA	NFINDR	ISRA	4 stages
	Time				Total	Time				Total
Carma	1.23	1.35	1.53	0.40	4.51	11.28	1.45	0.91	5.06	18.70
Fermi	0.18	0.11	0.08	0.05	0.42	0.81	0.12	0.06	0.92	1.91
Kepler	0.16	0.08	0.07	0.04	0.35	0.63	0.10	0.05	0.87	1.65
Serial	5.55	5.40	0.88	0.31	12.14	28.27	5.49	0.88	77.63	112.27
	Avg. power				Avg.	Avg. power				Avg.
Carma	28.3	19.5	19.6	22.8	22.2	20.6	19.9	20.0	42.6	26.4
Fermi	248.3	213.9	208.8	218.2	228.2	226.6	220.0	212.9	323.1	272.2
Kepler	251.5	255.9	204.4	211.4	238.5	215.5	199.3	203.1	261.1	238.1
	Max. power				Max.	Max. power				Max.
Carma	32.5	22.5	19.6	23.1	32.5	32.8	28.4	22.5	45.6	45.6
Fermi	322.0	232.3	218.9	236.7	322.0	281.1	243.1	218.1	336.7	336.7
Kepler	273.4	264.2	208.5	214.7	273.4	237.4	218.8	205.0	270.1	270.1
	Energy				Total	Energy				Total
Carma	34.8	26.3	30.0	9.1	100.2	232.4	28.9	18.2	215.6	495.0
Fermi	44.7	23.5	16.7	10.9	95.8	183.5	26.4	12.8	297.3	520.0
Kepler	40.2	20.5	14.3	8.5	83.4	135.8	19.9	10.2	227.2	393.0
	Net Energy				Total	Net Energy				Total
Carma	19.4	9.5	10.9	4.1	43.8	91.4	10.7	6.8	152.3	261.2
Fermi	27.1	12.8	8.9	6.0	54.7	104.2	14.6	6.9	207.1	332.9
Kepler	38.2	19.5	13.4	8.0	79.1	78.1	11.7	5.5	191.8	287.1

`WTC`

System	Chain #1					Chain #2				
	VD	PCA	NFINDR	LSU	4 stages	HYSIME	SPCA	NFINDR	ISRA	4 stages
	Time				Total	Time				Total
Carma	3.55	4.05	12.24	1.48	21.32	33.50	4.57	4.74	13.85	56.66
Fermi	0.44	0.32	0.44	0.18	1.38	2.62	0.37	0.28	2.47	5.74
Kepler	0.34	0.24	0.33	0.14	1.05	2.62	0.30	0.23	1.61	4.76
Serial	19.98	19.97	6.64	1.54	48.13	97.58	20.56	6.64	453.27	578.05
	Avg. power				Avg.	Avg. power				Avg.
Carma	28.9	20.0	18.8	23.7	21.0	20.6	20.8	19.8	45.5	26.6
Fermi	248.8	226.1	211.9	225.8	228.8	219.1	224.6	213.5	347.6	274.4
Kepler	240.1	234.5	205.9	214.5	224.7	198.4	203.9	205.8	280.8	226.9
	Max. power				Max.	Max. power				Max.
Carma	42.1	32.5	29.6	26.1	42.1	41.7	32.1	21.3	48.2	48.2
Fermi	277.4	273.7	228.1	245.4	277.4	315.1	254.3	247.4	358.3	358.3
Kepler	253.3	251.4	232.9	222.9	253.3	226.1	214.7	210.5	287.6	287.6
	Energy				Total	Energy				Total
Carma	102.6	81.0	230.1	35.1	448.7	690.1	95.1	93.9	630.2	1509.2
Fermi	109.5	72.4	93.2	40.6	315.7	574.0	83.1	59.8	858.6	1575.5
Kepler	81.6	56.3	67.9	30.0	235.8	519.8	61.2	47.3	452.1	1080.4
	Net Energy				Total	Net Energy				Total
Carma	58.2	30.4	77.1	16.6	182.2	271.4	37.9	34.6	457.1	800.9
Fermi	66.4	41.0	50.1	23.0	180.5	317.4	46.9	32.4	616.7	1013.3
Kepler	46.7	31.6	34.1	15.7	128.1	305.1	36.6	25.5	394.4	761.6

The execution times achieved by a sequential single-threaded implementation of the same processing chains are also reported in each case.

TABLE III
PERFORMANCE AND NET ENERGY OF THE GPU-EQUIPPED SYSTEMS AND THREE MULTICORE PROCESSORS ON ISRA

Scenario	Cuprite		WTC	
System	Time	Net Energy	Time	Net Energy
Carma	3.56	109.6	9.98	328.3
Fermi	0.66	143.0	1.73	434.7
Kepler	0.55	89.1	1.12	197.4
DSP	4.94	10.08	12.73	25.46
Xeon	0.69	79.41	2.42	280.86
Opteron1	0.34	82.21	1.79	407.85

TABLE IV
EXECUTION TIME AND POWER-ENERGY PERFORMANCE OF THE COMPLETE SPECTRAL UNMIXING CHAINS APPLIED TO THE Cuprite SCENARIO

Scenario	Cuprite					
System	Chain #1' (VD+NFINDR+ULS)			Chain #2' (HYSIME+NFINDR+ISRA)		
	Time	Max. power	Energy	Time	Max. power	Energy
Carma	3.16	32.5	73.9	17.25	45.6	466.1
Fermi	0.31	322.0	72.3	1.79	336.0	493.6
Kepler	0.27	273.1	63.0	1.55	270.2	373.1
AMD	2.30	489.4	787.3	16.97	540.6	8,640.3

Three of the multicore systems used in experiments were low-power processors (Intel Atom, Texas Instruments DSP, and ARM Cortex) while the remaining two corresponded to architectures designed for the desktop and server segments (Intel Xeon and AMD Opteron). That analysis revealed the AMD and the Texas Instruments architectures as clearly superior from the viewpoints, respectively, of performance and energy efficiency. For comparison purposes, we reconsider next the evaluation of ISRA on these two optimal processors as well as the Intel Xeon architecture included in that study:

1) DSP: A single Texas Instrument C6678 digital signal processor (8 cores at 1.0 GHz) with 512 MB of DDR3 RAM.
2) Xeon: Two Intel Xeon E5504 processors (4 cores per processor, at 2.0 GHz) with 32 GB of DDR3 RAM.
3) Opteron1: Two AMD Opteron 6128 processors (8 cores per processor, at 2.0 GHz) with 24 GB of DDR3 RAM.

Tuned implementations of BLAS were employed in all cases; see [15] for details. The datasets and the power measurement methodology are coherent with those employed for evaluation of the GPU-equipped systems mentioned above. To allow a fair comparison, we fixed the number of endmembers for Cuprite and WTC to 19 and 26, respectively, and performed 100 steps of the iteration underlying ISRA for all the implementations.

Table III reports the execution time and net energy obtained from this experiment. Kepler and Opteron1 deliver similar run times, slightly better for the former with WTC and the latter for Cuprite, in part explained by the cost of transferring the image to the GPU, which is better amortized in the case of a larger dataset. The differences in net energy are more relevant: the DSP is far more efficient than any other architecture, followed by the trio Kepler–Xeon–Opteron1 in the case of Cuprite, and only by Kepler for WTC.

E. Comparative Study Against Large Multicore Architectures

Following the trend toward platforms equipped with large numbers of cores, in [37], we investigated the performance-power-energy balance of hyperspectral unmixing on a high-performance platform with 64 cores:

- Opteron2: 4 AMD Opteron 6172 processors (12 cores per processor at 2.1 GHz) and 256 GB of DDR3 RAM.

We next reproduce part of this experiment, in order to expose the performance-power-energy tradeoff of this architecture against the previous GPU implementations. A tuned implementation of BLAS for the 64-core platform and the Cuprite dataset with 19 endmembers were employed in the following evaluation. The power measurement methodology is coherent with those employed for analysis of the GPU-equipped systems.

Table IV reports the execution time, maximum power, and energy resulting from the evaluation of two 3-stage chains, analogous to chains #1 and #2 defined earlier except that we drop the PCA and SPCA methods, using the Cuprite image. These results clearly illustrate the superiority of the GPU-equipped platforms in all metrics: execution time, maximum power, and energy consumption.

V. CONCLUSION AND FUTURE LINES

In this paper, we have explored the balance between performance and energy consumption of different computer architectures when executing different spectral unmixing chains for remotely sensed hyperspectral image analysis. Although our focus is on GPUs, we have also included other platforms such as DSPs and multicore processors in our comparison. In our opinion, this comparison is quite relevant since, to the best of our knowledge, this is the first reference addressing such detailed energy-performance comparison between GPUs and other platforms in the framework of hyperspectral imaging applications. The processing chains selected for evaluation purposes are highly representative of a widely used tool for hyperspectral image analysis. The performance-energy results reported in this contribution will be very useful in order to fully calibrate the possibilities of exploiting these kind of spectral unmixing algorithms in real missions for earth observation, in which energy consumption of hardware is a very important parameter, together with payload and computational performance. The results reported in our comparison indicate that, despite DSPs still offer a better energy-performance tradeoff, GPUs integrated in low-power boards are quickly advancing as an effective onboard devices in hyperspectral imaging applications.

Future work will be focused on analyzing additional low-power GPU architectures and hyperspectral imaging algorithms. In particular, faster GPU implementations exist in the literature for some of the modules used for the unmixing chains considered in this paper, most notably, the NFINDR endmember identification algorithm has been implemented very effectively in recent

references [32], [33] and this could lead to reduced power consumption. Alternative CPU/GPU hybrid implementations have also been recently presented for other endmember identification algorithms [35], and this paradigm should be further explored as it could also lead to more effective implementations in terms of energy consumption. Other architectures, such as field-programmable gate arrays (FPGAs) will be also evaluated from a performance-energy viewpoint in future developments.

References

[1] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for earth remote sensing," *Science*, vol. 228, pp. 1147–1153, 1985.

[2] K. Staenz, A. Mueller, A. Held, and U. Heiden, "Technical committees corner: International spaceborne imaging spectroscopy (ISIS) technical committee," *IEEE Geosci. Remote Sens. Newslett.*, no. 165, pp. 38–42, 2012.

[3] J. 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.

[4] A. Plaza, J. Plaza, A. Paz, and S. Sánchez, "Parallel hyperspectral image and signal processing," *IEEE Signal Process. Mag.*, vol. 28, no. 3, pp. 119–126, May 2011.

[5] S. López *et al.*, "The promise of reconfigurable computing for hyperspectral imaging on-board systems: Review and trends," *Proc. IEEE*, vol. 101, no. 3, pp. 698–722, Mar. 2013.

[6] R. Trautner, "ESA's roadmap for next generation payload data processors," in *Proc. DASIA Conf.*, vol. 1, 2011 [Online]. Available: http://www.esa.int/TEC/OBDP/

[7] A. Plaza and C.-I. Chang, "Special issue on high performance computing for hyperspectral imaging," *Int. J. High Perform. Comput.*, vol. 4, no. 3, pp. 528–544, 2011.

[8] A. Plaza, "Special issue on architectures and techniques for real-time processing of remotely sensed images," *J. Real-Time Image Proc.*, vol. 4, no. 3, pp. 191–193, 2009.

[9] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar./Apr. 2010.

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

[11] J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geosci. Remote Sens. Lett.*, vol. 4, no. 3, pp. 441–445, Jul. 2007.

[12] S. Sánchez, A. Paz, G. Martin, and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units," *Concurrency Comput.: Pract. Exp.*, vol. 23, no. 13, pp. 1538–1557, 2011.

[13] 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," *J. Real-Time Image Process.*, vol. 4, pp. 1–14, 2009.

[14] S. Bernabe, S. Lopez, A. Plaza, and R. Sarmiento, "GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 2, pp. 221–225, Mar. 2013.

[15] M. Castillo *et al.*, "Hyperspectral unmixing on DSPs: Trading off performance for energy," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, 2013, to be published, doi: 10.1109/JSTARS.2013.2266927.

[16] M. Winter, "N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data," in *Proc. SPIE*, 1999, vol. 3753, pp. 266–270.

[17] 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.

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

[19] I. T. Jolliffe, *Principal Component Analysis*. New York, NY, USA: Springer-Verlag, 1986.

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

[21] C. Gonzalez, J. Resano, A. Plaza, and D. Mozos, "FPGA implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 1, pp. 248–261, Feb. 2012.

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

[23] M. Clint, *The Eigensolution of Unsymmetric Matrices by Simultaneous Iteration*. The Queen's University of Belfast, 1970.

[24] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral endmember determination in hyperspectral data," in *Proc. SPIE*, 1999, vol. 3753, pp. 266–277.

[25] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 650–663, Mar. 2004.

[26] D. Heinz and C.-I. Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 3, pp. 529–545, Mar. 2001.

[27] C.-I. Chang and D. Heinz, "Constrained subpixel target detection for remotely sensed imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 38, no. 3, pp. 1144–1159, May 2000.

[28] C. A. Bateson, G. P. Asner, and C. A. Wessman, "Endmember bundles: A new approach to incorporating endmember variability into spectral mixture analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 38, no. 2, pp. 1083–1094, Mar. 2000.

[29] S. Sánchez and A. Plaza, "Fast determination of the number of endmembers for real-time hyperspectral unmixing on gpus," *J. Real-Time Image Process.*, pp. 1–9, 2012 [Online]. Available: http://dx.doi.org/10.1007/s11554-012-0276-3

[30] S. Sánchez, R. Ramalho, L. Sousa, and A. Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs," *J. Real-Time Image Process.*, pp. 1–15, 2012 [Online]. Available: http://dx.doi.org/10.1007/s11554-012-0269-2

[31] C. González *et al.*, "Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing," *Integr. VLSI J.*, vol. 46, no. 2, pp. 89–103, 2013 [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926012000223

[32] H. Qu, B. Huang, J. Zhang, and Y. Zhang, "An improved maximum simplex volume algorithm to unmixing hyperspectral data," in *Proc. SPIE*, 2013, vol. 8895, pp. 889 507–889 507-7 [Online]. Available: http://dx.doi.org/10.1117/12.2034759

[33] Z. Wu, S. Ye, J. Wei, Z. Wei, L. Sun, and J. Liu, "Fast endmember extraction for massive hyperspectral sensor data on GPUs," *Int. J. Distrib. Sensor Netw.*, vol. 2013, no. 217180, pp. 1–7, 2013 [Online]. Available: http://dx.doi.org/10.1155/2013/217180

[34] J. Nascimento and J. Bioucas-Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. and Remote Sens.*, vol. 43, no. 4, pp. 898–910, Apr. 2005.

[35] B. Huang, A. Plaza, and Z. Wu, "Acceleration of vertex component analysis for spectral unmixing with CUDA," in *Proc. SPIE*, 2013, vol. 8895, pp. 889 509–889 509-10 [Online]. Available: http://dx.doi.org/10.1117/12.2031527

[36] J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection," *IEEE Trans. Geosci. Remote Sens.*, vol. 32, no. 4, pp. 779–785, Jul. 1994.

[37] A. Remón, S. Sánchez, S. Bernabé, E. S. Quintana-Ort, 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, 2013 [Online]. Available: http://dx.doi.org/10.1186/1687-6180-2013-68

Sergio Sánchez received the Ph.D. degree in computer engineering from the University of Extremadura, Spain, in 2013, and is currently pursuing the Ph.D. Research Associate with the Department of Chemical and Environmental Engineering, Masdar Institute of Science and Technology, Abu Dhabi, UAE.

His research interests include hyperspectral image analysis and efficient implementations of large-scale scientific problems on commodity graphical processing units (GPUs).

Germán León received the Ph.D. degree in computer science from the Universidad Jaume I (UJI), Castellon, Spain, in 2012, and the B.S. degree in computer science from the Polytechnic University of Valencia, Spain, in 1992.

Since 1999, he is an Associate Professor with the Department of Computer Science and Engineering of the UJI. His research interests include parallel programming, linear algebra, power consumption, as well as advanced architectures, hardware accelerators, and high-level compilation techniques.

Antonio Plaza is an Associate Professor (with accreditation for Full Professor) with the Department of Technology of Computers and Communications, University of Extremadura, Badajoz, Spain, where he is the Head of the Hyperspectral Computing Laboratory (HyperComp).

He was the Coordinator of the Hyperspectral Imaging Network, a European project with total funding of 2.8 MEuro. He authored more than 400 publications, including 119 JCR journal papers (71 in IEEE journals), 20 book chapters, and over 240 peer-reviewed conference proceeding papers (94 in IEEE conferences). He has guest edited seven special issues on JCR journals (three in IEEE journals). He has been a Chair for the IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing in 2011.

Dr. Plaza is a recipient of the recognition of Best Reviewers of the IEEE Geoscience and Remote Sensing Letters in 2009, and a recipient of the recognition of Best Reviewers of the IEEE Transactions on Geoscience and Remote Sensing in 2010, a journal for which he has served as Associate Editor in 2007–2012. He is also an Associate Editor for IEEE Access, and was a member of the Editorial Board of the IEEE Geoscience and Remote Sensing Newsletter in 2011–2012, and the IEEE *Geoscience and Remote Sensing Magazine* in 2013. He was also a member of the steering committee of the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing in 2012. He served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS) in 2011–2012, and is currently serving as President of the Spanish Chapter of IEEE GRSS (since November 2012). He is currently serving as the Editor-in-Chief of the IEEE Transactions on Geoscience and Remote Sensing journal (since January 2013).

Enrique S. Quintana-Ortí received the B.S. and Ph.D. degrees in computer sciences from the Universidad Politecnica de Valencia, Valencia, Spain, in 1992 and 1996, respectively.

Currently, he is a Professor in Computer Architecture in the Universidad Jaume I, Castellán, Spain. He has published more than 200 papers in international conferences and journals, and has contributed to software libraries like SLICOT and libflame. His research interests include parallel programming, linear algebra, power consumption, as well as advanced architectures and hardware accelerators.