Hyperspectral Unmixing on Multicore DSPs: Trading off Performance for Energy

Maribel Castillo, Juan C. Fernández, Francisco D. Igual,
Antonio Plaza, Enrique S. Quintana-Ortí, and Alfredo Remón

***Abstract*—Wider coverage of observation missions will increase onboard power restrictions while, at the same time, pose higher demands from the perspective of processing time, thus asking for the exploration of novel high-performance and low-power processing architectures. In this paper, we analyze the acceleration of spectral unmixing, a key technique to process hyperspectral images, on multicore architectures. To meet onboard processing restrictions, we employ a low-power Digital Signal Processor (DSP), comparing processing time and energy consumption with those of a representative set of commodity architectures. We demonstrate that DSPs offer a fair balance between ease of programming, performance, and energy consumption, resulting in a highly appealing platform to meet the restrictions of current missions if onboard processing is required.**

***Keywords*-Digital signal processors, hyperspectral imaging, energy consumption, 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]. Satellites in operation recently, like `EO-1 Hyperion`[1] (USA), feature a spatial resolution of a few dozens of meters and a revisit time between 3 and 16 days. 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.

Spectral unmixing [2]–[4] is among the most popular techniques to process hyperspectral images, enabling sub-pixel characterization. Consider the linear mixture model in compact matrix form [5]:

$$\mathbf{Y} = \mathbf{EA} + \mathbf{N}, \tag{1}$$

where $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m] \in \mathbb{R}^{n \times m}$ is the hyperspectral image consisting of m pixels (columns) each with n bands (rows), $\mathbf{E} \in \mathbb{R}^{n \times p}$ is the *endmember matrix*, $\mathbf{A} \in \mathbb{R}^{p \times m}$ contains the *endmember abundances* for each pixel of the scene, and $\mathbf{N} \in \mathbb{R}^{n \times m}$ is the noise. Given a number of endmembers p, solving the linear mixture model then involves identifying a collection (matrix) of endmembers $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_p]$, and subsequently estimating the fractional abundances of the p endmembers for each pixel in the hyperspectral data set.

Recent spectral unmixing methods for the linear mixture model are computationally quite costly, being subject to strict restrictions on the response time for applications like, e.g., wild land fire tracking, biological threat detection, and monitoring chemical contamination. To address this problem, a variety of parallel systems have been leveraged in the quest for real-time performance, from small clusters of computers and general-purpose multicore processors to fancier architectures such as field programmable gate arrays (FPGAs) and graphics processing units (GPUs); see, e.g., [6]–[14].

In this paper we investigate an alternative architecture, namely a multicore digital signal processor (DSP) from Texas Instruments (TI), that blends extremely low power consumption, fair computational performance, and relatively easy programming, yielding an appealing choice for airborne and spaceborne remote sensing. In particular, we elaborate a thoughtful analysis of the trade-off between performance and power offered by this architecture, comparing the results with state-of-the-art commodity multicore processors from Intel and AMD, as well as low-power multicore challengers, e.g., from ARM. This type of study is timely and highly relevant to assess the real possibilities of applying today's multicore processors, including DSPs, to efficient hyperspectral image processing in real remote sensing missions.

The rest of the paper is structured as follows. In Section II we briefly review two efficient methods for identifying the endmembers and estimating their fractional abundances in hyperspectral images, respectively Orthogonal Subspace Projection via Gram-Schmidt (OSP-GS) [15] and the Image Space Reconstruction Algorithm (ISRA) [16]. The reasons for the selection of these two methods can be summarized as follows. On the one hand, OSP-GS is a very fast (and regular) endmember extraction algorithm which provides robust results and is relatively easy to parallelize. On the other hand, ISRA provides abundance estimations which are always positive. This is very important in spectral unmixing applications as the derivation of negative abundances has no physical meaning. Also, ISRA does not constraint abundances to sum to one in each pixel as this can lead to model errors if the endmembers are not perfectly selected. As a result, ISRA provides a robust framework for abundance estimation and is also a regular and easy algorithm to parallelize. In Section III we consider the programmability issue for the target DSP, describing the paral-

Maribel Castillo, Juan C. Fernández, Enrique S. Quintana-Ortí, and Alfredo Remón are with Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain (e-mails: {castillo,jfernand,quintana,remon}@icc.uji.es).

Francisco D. Igual is with Depto. de Arquitectura de Computadores y Automática, Univ. Complutense de Madrid, 28.040–Madrid, Spain (e-mail: figual@fdi.ucm.es).

Antonio Plaza is with Hyperspectral Computing Laboratory (HyperComp), Department of Technology of Computers and Communications, University of Extremadura, 10.071–Cáceres, Spain (e-mail: aplaza@unex.es).

[1]http://eol.gsfc.nasa.gov

lelization and fine tuning of the methods on this architecture, while offering a glimpse of the effort needed to program the same methods on general-purpose multicore architectures. In Section IV we perform an experimental comparison of the methods on the candidate architectures from the point of view of both performance and energy consumption. Finally, we close the paper in Section V with some remarks and hints at plausible future research lines.

II. Methods for Spectral Unmixing

A. Endmember identification

OSP [15] was a method originally conceived to find spectrally distinct signatures using orthogonal projections. From the mathematical point of view, our implementation is a variant that employs modified GS transforms [17] to compute an orthonormal p-dimensional basis of the subspace of $\mathbb{R}^{n\times m}$ spanned by $\mathbf{Y}$, combined with pivoting. At each step of the orthogonalization, such pivoting detects the pixel with maximum projection value among those of the image. Unfortunately, this requires that each projector is applied to all m pixels of the scene, not only to p (the number of endmembers to be detected), yielding a significant increase in the arithmetic cost of the algorithm. Given the $3n$ floating-point arithmetic operations (flops) required to apply the projector to one pixel, and the p endmembers that have to be identified, the result is a total cost for the algorithm of $3mnp$ flops.

Algorithm 1 describes the procedure in detail. The first p columns of $\mathbf{Y}$ are overwritten with the set of orthogonal vectors generated by the GS method, forming an orthonormal set that spans the sought-after subspace. While there exist numerically more robust alternatives, e.g. based on the QR factorization with column pivoting [17], we adopt the GS variant because of its reduced cost and straight-forward implementation. On the other hand, in the practical implementation we have applied optimization techniques to further reduce the computational cost of the algorithm. For example, we avoid the recomputation of the norms in line 2, accomodating instead a norm-downdating scheme analogous to that included in routine `dgeqpf` of LAPACK for the QR factorization with column pivoting [18].

Algorithm 1 Pseudocode of the OSP-GS algorithm for identifying p endmembers of a hyperspectral image consisting of m pixels $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m]$ and n bands. Upon completion, the endmembers correspond to the pixels with "coordinates" stored in vector $\mathbf{v}$.

1: **for** $k = 1 : p$
2: $\quad \mathbf{v}(k) = \{k + \hat{k} - 1 \mid \mathbf{Y}(\hat{k}) = \max_{j=k}^{m} \|\mathbf{Y}(1:n, j)\|_2^2\}$
3: $\quad \mathbf{Y}(1:n, \mathbf{v}(k)) \leftrightarrow \mathbf{Y}(1:n, k)$ (swap contents)
4: $\quad \mathbf{w}(k) = \|\mathbf{Y}(1:n, k)\|_2$
5: $\quad \mathbf{Y}(1:n, k) = \mathbf{Y}(1:n, k)/\mathbf{w}(k)$
6: $\quad$ **for** $j = k + 1 : m$
7: $\quad\quad \mathbf{w}(k, j) = \mathbf{Y}(1:n, k)^T\ \mathbf{Y}(1:n, j)$
8: $\quad\quad \mathbf{Y}(1:n, j) = \mathbf{Y}(1:n, j) - \mathbf{Y}(1:n, k)\mathbf{w}(k, j)$
9: $\quad$ **end for**
10: **end for**

Algorithm 2 Pseudocode of ISRA for unmixing a hyperspectral image consisting of m pixels $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m]$ and n bands using a set $\mathbf{E}$ of p endmembers. Upon completion, the abundances are returned in matrix $\hat{\mathbf{A}} \in \mathbb{R}^{p\times m}$.

1: $\mathbf{N} = \mathbf{E}^T\mathbf{Y}$
2: $\mathbf{D} = \mathbf{E}^T\mathbf{E}$
3: $\hat{\mathbf{A}} =$ **rand**$(p \times m)$ (Generate random matrix)
4: $k = 0$
5: **repeat**
6: $\quad \mathbf{A} = \mathbf{D}\,\hat{\mathbf{A}}$, $k = k + 1$
7: $\quad$ **for** $j = 1 : m$
8: $\quad\quad$ **for** $i = 1 : p$
9: $\quad\quad\quad \hat{\mathbf{A}}(i, j) = \hat{\mathbf{A}}(i, j)\ \mathbf{N}(i, j)/\mathbf{A}(i, j)$
10: $\quad\quad$ **end for**
11: $\quad$ **end for**
12: **until** convergence **or** $k \geq$ *max_iter*

B. Abundance estimation

Once a collection of p endmembers $\mathbf{E}$ has been estimated using the OSP-GS algorithm (or other alternative identification method), an unconstrained p-dimensional estimate of the endmember abundances for a given pixel in $\mathbf{y}$ is simply given (in least squares sense) by [19]:

$$\hat{\mathbf{a}}^{\text{UC}} = (\mathbf{E}^T\mathbf{E})^{-1}\mathbf{E}^T\mathbf{y}, \tag{2}$$

where $\hat{\mathbf{a}}^{\text{UC}} = [a_1, a_2, \ldots, a_p] \in \mathbb{R}^p$. However, to avoid the derivation of negative abundances caused, e.g., by spatial or temporal variations [20], it is possible to introduce the abundance non-negativity constraint (ANC), enforcing $a_j \geq 0$ for all j, which results in the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{a}\in\Delta} \left\{ (\mathbf{y} - \mathbf{a}\cdot\mathbf{E})^T (\mathbf{y} - \mathbf{a}\cdot\mathbf{E}) \right\}, \\ &\text{subject to: } \Delta = \{\mathbf{a} | a_j \geq 0 \text{ for all } j\}. \end{aligned} \tag{3}$$

A non-negative constrained least-squares (NCLS) algorithm [21] can then be used to iteratively obtain a solution to (3). A successful approach for this purpose in different applications is ISRA [16], a multiplicative algorithm for solving NCLS problems. The algorithm is based on the iteration:

$$\hat{\mathbf{a}}^{k+1} = \hat{\mathbf{a}}^k \otimes \left(\mathbf{E}^T\ \mathbf{y}\right) \oslash \left(\mathbf{E}^T\mathbf{E}\ \hat{\mathbf{a}}^k\right), k \geq 0, \tag{4}$$

where the operators $\otimes$ and $\oslash$ denote, respectively, the elementwise vector product and vector division. The procedure thus starts with an unconstrained abundance estimation $\hat{\mathbf{a}}^0 = \hat{\mathbf{a}}^{\text{UC}}$, which is progressively refined.

Algorithm 2 details a matrix-oriented variant of ISRA for unmixing an image $\mathbf{Y}$ using a set of $\mathbf{E}$ endmembers. There, *max_iter* is a threshold to avoid stagnation in the convergence of the iteration, while the initial abundance estimations in $\hat{\mathbf{A}}$ are randomly generated (line 3). The ISRA procedure is composed of very simple arithmetic operations, but the innermost loop, for variables m and p, dominates its arithmetic cost. In particular, as 2 arithmetic operations are performed at each iteration of this loop, this yields a total cost for the algorithm of $2mpk$ flops.

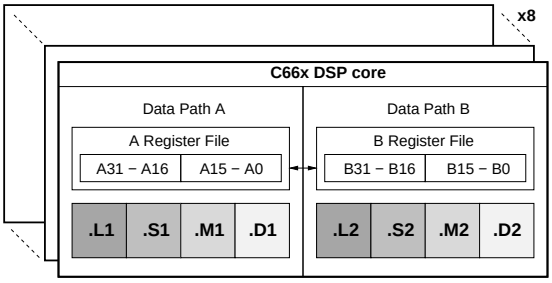

Fig. 1. C66x DSP Core Block Diagram.

III. Spectral Unmixing on TI DSPs

A. Architecture overview

The C6678 from TI is a high performance, low power DSP with floating-point capabilities [22]. It contains eight C66x Very Long Instruction Word (VLIW) cores, running at 1 GHz.

The C66x core, in Figure 1, is the base of the multicore C6678 DSP architecture. It is implemented as a VLIW architecture, and targets three different types of concurrency:

1) Instruction-level parallelism: In the core, eight different functional units are arranged in two independent sides, each with four processing units, namely L, M, S and D. The M units are devoted to multiplication operations. The D unit performs address calculations and load/store instructions. The L and S units are reserved for additions/subtractions, logical, branch and bitwise operations. Thus, this 8-way VLIW machine can issue up to eight instructions per cycle.

2) Data-level parallelism: The C66x instruction set comprises Single Instruction Multiple Data (SIMD) instructions that operate on 128-bit vector registers. More precisely, the M unit performs 4 single-precision (SP) multiplications per cycle while the L and S units carry out 2 SP additions per cycle. Thus, the C66x is ideally able to perform 8 SP multiply-add (MADD) operations in one cycle. With eight C66x cores, a C6678 processor running at 1 GHz this yields 128 SP GFLOPS (1 GFLOPs = 10^9 floating point operations per second.) All floating-point operations are IEEE754-compliant.

3) Thread-level parallelism: This can be exploited by running different threads across the cores of the DSP. In our case, we will use OpenMP[2] to benefit from this type of concurrency.

B. Programming the DSP

The C6678 runs a lightweight real-time native operating system called SYS/BIOS. The compiler is C89/C++98-compliant, and virtually any C89 code can be ported with negligible additional effort. To improve the efficiency of the generated code, the compiler provides optimization techniques in the form of `#pragmas` and intrinsic SIMD instructions to fully exploit the core architecture, and extract all the potential performance without resorting to assembly programming.

The compiler supports OpenMP 3.0 to allow rapid porting of existing multi-threaded codes to multicore DSPs. The OpenMP runtime performs the appropriate cache control operations to maintain the consistency of the shared memory when required, but special caution is due to keep data coherency for shared variables, as no hardware support for cache coherence across cores is provided.

[2]http://openmp.org

C. Implementation details of spectral unmixing methods

Basic codes developed for conventional architectures can be ported to the TI DSP architecture with minor modifications. However, the special features of the C66x reviewed in the previous section ask for a number of optimizations in order to achieve high performance. Our implementations of the spectral unmixing algorithms on the TI DSP followed a four-step refinement procedure that can be applied to port many other numerically-intensive scientific codes to this architecture. In order to illustrate this process, we will mostly refer to the method for aboundance estimation in Algorithm 2. However, analogous techniques were applied to derive a correct and efficient implementation of Algorithm 1.

1) Use of optimized libraries: The *Basic Linear Algebra Subprograms* (BLAS) specifies a set of linear algebra routines that appear frequently in scientific applications. Since its inception, highly tuned implementations of BLAS have been developed by processor manufacturers or by the scientific community. These codes are usually designed to extract all the potential performance of the target architecture. Thus, their use guarantees high performance while simplifying the optimization process.

In our case, we identified several numerical operations in the codes which could be replaced by calls to BLAS. This is the case, e.g., of the matrix-matrix products in lines 1, 2 and 6 of Algorithm 2, which were computed via simple invocations to a highly-tuned implementation of the Level-3 BLAS kernel `sgemm` from TI [23], [24]. Internally, this implementation encapsulates all the necessary optimizations (at instruction, data and thread levels) to ensure high performance.

The BLAS implementation from TI presents some restrictions from the point of view of the matrix operands, as their dimensions are required to be integer multiples of 32. These restrictions are mainly enforced to maintain cache coherence among the cores of the chip, avoiding false sharing among different local caches. This problem can be solved by padding all data structures to the next integer multiple of 32. This implies, for example, that instead of working on an image $\mathbf{Y}$ composed of m pixels with n bands each, our algorithms operate on an $n' \times m'$ matrix, with n' and m' standing for the closest integer multiples of 32 larger or equal than the original values. The additional rows/columns (if any) are simply filled with zeros, and they do not participate in the arithmetic computations as the algorithms still operate on the leading $n \times m$ submatrix of this data structure. For the images analyzed in the next section, the storage overhead introduced by this technique is less than 2.5% and thus clearly affordable.

2) Exploitation of instruction-level parallelism: VLIW architectures like the TI DSP usually require the use of sophisticated compilers that generate tuned code to avoid stalls of the functional units. However, the programmer is welcome to provide additional information in terms of directives (via `#pragma` constructs in the code or reserved keywords) that

help the compiler in the optimization process. Many of these directives are directly related with loop optimizations (e.g., providing safe loop unroll counts), pointer deambiguation, and data alignment information.

After applying these common optimizations, we found out that the floating-point arithmetic divisions and square-root operations present in the codes were a major source of inefficiency, the reason being twofold. First, this kind of operations is intrinsically slow on the C66x, easily consuming dozens of cycles more than the floating-point arithmetic multiplication or addition. Second, the presence of this type of operations inside a loop prevents the automatic application of software pipelining by the compiler, that is key to attaining high performance. This is the case, e.g., of the loop in lines 7–11 of Algorithm 2. To avoid this problem, we replaced the division by an estimate obtained with the `_rcpsp` intrinsic, and the square root by the `_rsqrsp` intrinsic, which are both translated into native instructions of the C66x instruction set, requiring a single cycle to complete. These instructions only offer an approximate solution of the respective operations. To regain higher precision, a reduced number of Newton-Raphson interpolation steps was applied. These steps are cheaper to compute and are based on multiply-add operations, allowing the application of software pipelining while offering enough accuracy for SP data.

3) Exploitation of data-level parallelism: In the next stage, we applied manual SIMD vectorization to certain fragments of the codes. Following with Algorithm 2, the loop in lines 7–11 was unrolled with factors 2 and 4, to exploit the 64-bit and 128-bit arithmetic and load/store units in the C66x cores.

4) Exploitation of thread-level parallelism: Finally, we applied OpenMP pragmas to distribute the workload among the 8 cores of the DSP. In the case of the nested loop in lines 7–11 of Algorithm 2, we simply preceded this loop in the actual implementation by the OpenMP directive `#pragma omp parallel for private(`i`)`. The iteration range was divided into m/t chunks of consecutive pixels, with t being the number of CPU threads, and each chunk was processed by a different thread in parallel.

D. Programability compared with conventional architectures

Ease of programming is a subjective topic. Developing basic implementations for either specific-purpose architectures like the TI DSP, or general-purpose processors like the Intel/AMD x86 or the ARM Cortex A9, does not require significantly different efforts. From that starting point, the optimization labor depends on the specific target architecture and the desired optimization level. However, the techniques to apply are not that different for both types of architectures. Highly optimized libraries, e.g. BLAS, are generally available for x86 architectures (e.g., Intel MKL or AMD ACML), ARM processors (e.g., ATLAS) or TI DSPs, and ease the optimization process. Intel/AMD x86 and ARM Cortex, being superscalar processors, defer the extraction of instruction-level parallelism till runtime, removing this burden from the compiler. However, current compilers for VLIW architectures usually facilitate the optimization process. SIMD instructions (SSE or AVX for Intel x86, Neon for ARM) are fundamental to fully exploit the potential of arithmetic units, as well as for the TI DSP. Finally, the introduction of OpenMP in novel multi-core DSPs dramatically decreases the difficulty of extracting thread-level parallelism, levelling the effort with that required for general-purpose processors. In summary, tuning a basic code for a TI DSP, while not being an easy process if the last drop of performance is sought-after, does not significantly differ from that required for other type of modern multi-core architectures.

IV. Experimental Results

In this section we present the two hyperspectral data sets used in the evaluation; describe the multicore processors and power measurement devices; and finally perform a detailed analysis of performance vs energy consumption.

A. Hyperspectral data sets

We employed two hyperspectral images in the experiments: the Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) `Cuprite` data set, a widely used benchmark for evaluation of spectral unmixing techniques, and an image collected over the World Trace Center (`WTC`) in New York after the terrorist attacks on September 11. The portion in the experiments with `Cuprite` corresponds to an m=350×350-pixel subset of the sector labeled as `f970619t01p02_02_sc03.a.rfl` in the online data[3]. This scene presents n=188 spectral bands between 0.4 and 2.5 μm, and has been widely used to validate the performance of spectral unmixing methods. The `WTC` data set consists of m=512×614 pixels and n=224 bands. These dimensions represent the standard data cube size recorded by AVIRIS. Independent experimentation led us to set the number of endmembers to p=19 for `Cuprite` and p=26 for `WTC`. The (fixed) number of iterations set for ISRA was 100 in both cases. Processing these images in real-time requires execution times below 2.98s for `Cuprite` and 5.09s for `WTC`, resulting from the fact that the AVIRIS instrument requires 8.3 milliseconds to collect a full line made up of 512 pixel vectors.

B. Multicore processors and power measurement setup

Our experiments were performed on a collection of platforms representative of current multicore technology. We next list the main components of these platforms and the implementation of BLAS employed in each case:

- TI C6678 DSP (8 cores) at 1 GHz with 512 Mbytes of RAM, and a proprietary BLAS implementation from TI.
- Two Intel Xeon E5504 (8 cores) at 2.0 GHz with 32 Gbytes of RAM, and Intel MKL 10.3.9.
- Intel Atom D510 (2 cores) at 1.66GHz with 2 Gbytes of RAM, and Intel MKL 10.3.10.
- Two AMD Opteron 6128 (16 cores) at 2.0 GHz with 24 Gbytes of RAM, and Intel MKL 11.1.
- ARM Cortex A9 (2 cores) at 1 GHz (built by TI) with 1 Gbyte of RAM, and ATLAS 3.8.4.

[3]http://aviris.jpl.nasa.gov/freedata

TABLE I
PROCESSOR TDP, AND PLATFORM MAXIMUM OBSERVED POWER (WITH ALL CORES IN USE) AND IDLE POWER.

Processor	TDP	Max. power	System (idle) power
TI DSP	10.0	22.7	18.1
Intel Xeon	80.0	184.8	67.0
Intel Atom	13.0	29.7	21.6
AMD Opteron	115.0	250.6	101.2
ARM Cortex	0.6	5.5	3.6

This selection of architectures offers a wide sample of different performance/power ratios, and includes high-performance/high-power architectures (e.g. Xeon and Opteron), low-performance/ultra low-power architectures (e.g. ARM Cortex and Atom), and mid-performance/low-power architectures (e.g. TI DSP). This range covers the necessities of a wide variety of applications or scenarios in which performance and/or power consumption are the limiting factors.

Power was measured using a *WattsUp? Pro .Net* powermeter directly attached to the cable that connects the electric socket to the PSU (power supply unit) of the target system, thus measuring external AC for the full platform. The system power (i.e., power dissipated while the corresponding platform remained idle; see Table I) was subtracted from all the measurements so as to obtain comparable values, independent of the power dissipated by elements other than the processor(s) and memory. The test (spectral method) was repeated during 3 minutes before power measurements were collected. Then, samples were collected with the test running 3 more minutes; and power was averaged over this period and multiplied by the execution time of one single test to obtain the total energy. In all cases, execution time is reported in seconds (s), energy in Joules (J), and power in Watts (W).

C. Performance and power trade-off

Table II reports the execution time and energy consumption resulting from our complete experimentation, involving the two data sets, both methods, and the five processor types (with 1, 2, ... up to 16 cores, depending on the architecture). For each processor, the bold face in the table identifies the optimal number of cores from the point of view of execution time and energy. Note that, as the energy depends linearly on the execution time, a shorter time could be expected to also yield a lower energy consumption. Interestingly, this is not always the case. A notable example is the application of OSP-GS to the `WTC` scene on the AMD Opteron. The use of 16 cores of this platform renders and execution time of only 0.75s, but an energy consumption of 171.25J. On the other hand, when 8 cores are employed, the time grows to 1.01s (increase of 35%) but the energy is reduced to 149.39J (decrease of 12.76%).

Regarding parallel efficiency, the parallel versions of ISRA scale better across multiple cores than those from OSP-GS. The routines invoked by the latter (mainly from levels 1 and 2 BLAS) involve memory-bound computations so that memory becomes a strong bottleneck as the number of cores is increased. This effect is less evident in ISRA, where the underlying operations (from level 3 BLAS, e.g. `sgemm`) are not limited by memory bandwidth.

Table III offers a more compact and refined view of the information extracted from the experimental study, that allows a direct comparison between the different architectures. Specifically, in this table we report the global execution times and energy consumptions of the two-stage spectral unmixing chain (endmember identification+abundance estimation). For each processor, we offer the optimal (i.e., minimum) execution time and energy consumption in the two columns labelled as "Best time"/"Best energy". The numbers in bold face there identify the best platform from the viewpoint of the corresponding magnitude. For instance, in the `Cuprite` scene, the shortest time among all platforms is 0.54s, obtained when 16 cores of the AMD Opteron were employed to apply both OSP-GS and ISRA (0.20+0.34s). The values inside parenthesis (columns labelled as "Norm.") correspond to normalized values w.r.t. the best result; and the columns with labels "Energy"/"Time" report those magnitudes for their respective "Best time"/"Best energy" case. For the sake of clarity, a comparison of normalized values w.r.t. the best architecture in each case is also reported in Figure 2 for both scenes. Following with the same example, processing the `Cuprite` scene in 0.54s on (the 16 cores of) the AMD Opteron required (46.77+82.21=)128.98J; but using only 8 cores for OSP-GS and 16 for ISRA decreased the energy slightly, to (40.95+82.21=)123.16J, and increased the execution time to (0.27+0.34=)0.61s.

These results show a manifest trade-off between performance (in terms of execution time) and energy. If the goal is high performance, then the clear solution is to use the 16 cores of the AMD Opteron for the two data sets. On the other hand, the low thermal design power (TDP; see Table I) of the TI DSP results in this architecture being patently superior from the energy perspective. The Intel Xeon presents a behaviour that is similar to that of the AMD Opteron, with execution times shorter than those of the TI DSP, but higher consumption of energy for the two images. The Intel Atom and ARM Cortex, while being low-power general-purpose architectures, are far from the energy-efficiency and performance of the TI DSP. A direct comparison between the two winners, AMD Opteron and TI DSP, reveals that the AMD Opteron consumes (128.98/12.93=)9.98× more energy than the TI DSP to process the `Cuprite` image, in exchange for being (5.58/0.54=)10.33× faster; and (578.83/36.89=)15.69× more energy for `WTC` while being (15.27/2.54=)6.01× faster in this case. On the other hand, the TI DSP is (5.58/0.61=)9.15× and (15.27/2.80=)5.45× slower than the AMD Opteron, for `Cuprite` and `WTC` respectively, but (123.16/12.93=)9.53× and (556.97/36.89=)15.10× more efficient in terms of energy consumption.

One interesting question to analyze is the execution time w.r.t. the real-time performance, in particular, which processors are below this threshold (or close to it) and how much we can extend the execution time to reduce energy consumption while still attaining real time. Given the real-time processing baselines of 2.98s for `Cuprite` and 5.09s for `WTC`, only the AMD Opteron and the Intel Xeon meet the bounds for both data sets. The TI DSP is above the required thresholds, by a

TABLE II
EXECUTION TIME AND ENERGY CONSUMPTION REQUIRED BY THE SPECTRAL UNMIXING METHODS.

Data set	Method	Processor/#cores	Time					Energy				
			1	2	4	8	16	1	2	4	8	16
Cuprite	OSP-GS	TI DSP	4.29	2.06	1.10	**0.64**	–	4.33	3.29	2.88	**2.85**	–
		Intel Xeon	0.59	**0.45**	0.48	0.51	–	29.84	**27.81**	36.88	49.73	–
		Intel Atom	4.29	**3.69**	–	–	–	27.03	**26.94**	–	–	–
		AMD Opteron	1.17	0.66	0.40	0.27	**0.20**	94.01	61.73	45.96	**40.95**	46.77
		ARM Cortex	8.04	**7.80**	–	–	–	**9.64**	14.04	–	–	–
	ISRA	TI DSP	17.11	10.17	6.48	**4.94**	–	27.37	17.29	11.66	**10.08**	–
		Intel Xeon	2.01	1.25	1.08	**0.69**		94.88	**76.00**	85.30	79.41	–
		Intel Atom	24.73	**13.27**	–	–	–	153.33	**98.20**	–	–	–
		AMD Opteron	3.50	2.71	1.55	0.79	**0.34**	274.05	243.90	172.67	117.24	**82.21**
		ARM Cortex	24.28	**15.14**	–	–	–	29.14	**27.25**	–	–	–
WTC	OSP-GS	TI DSP	17.91	8.40	4.33	**2.54**	–	19.70	13.69	11.47	**11.43**	–
		Intel Xeon	2.22	**1.82**	1.96	2.09	–	121.03	**113.22**	146.96	203.12	–
		Intel Atom	22.91	**20.00**	–	–	–	144.33	**144.08**	–	–	–
		AMD Opteron	4.59	2.53	1.57	1.01	**0.75**	362.47	240.30	180.16	**149.39**	171.25
		ARM Cortex	34.94	**34.87**	–	–	–	**41.93**	72.97	–	–	–
	ISRA	TI DSP	43.52	26.10	16.43	**12.73**	–	69.63	44.37	29.57	**25.46**	–
		Intel Xeon	7.31	4.50	3.75	**2.42**	–	356.58	**271.17**	307.00	280.86	–
		Intel Atom	82.90	**47.27**	–	–	–	513.98	**349.80**	–	–	–
		AMD Opteron	12.95	9.89	5.61	3.01	**1.79**	1,028.23	905.92	638.98	451.50	**407.58**
		ARM Cortex	91.84	**52.89**	–	–	–	110.21	**95.20**	–	–	–

TABLE III
COMPARISON OF THE EXECUTION TIME AND ENERGY CONSUMPTION REQUIRED BY THE COMPLETE (TWO-STAGE) SPECTRAL UNMIXING CHAIN.

Data set	Processor	Best time	Norm.	Energy	Best energy	Norm.	Time
Cuprite	TI DSP	5.58	(10.33)	12.93	**12.93**	(1.00)	5.58
	Intel Xeon	1.14	(2.11)	107.22	103.81	(8.03)	1.70
	Intel Atom	16.96	(31.41)	125.14	125.14	(9.68)	16.96
	AMD Opteron	**0.54**	(1.00)	128.98	123.16	(9.53)	0.61
	ARM Cortex	22.94	(42.48)	41.29	36.89	(2.85)	23.18
WTC	TI DSP	15.27	(6.01)	36.89	**36.89**	(1.00)	15.27
	Intel Xeon	4.24	(1.67)	394.08	384.39	(10.42)	6.32
	Intel Atom	62.27	(24.52)	493.88	493.88	(13.39)	67.27
	AMD Opteron	**2.54**	(1.00)	578.83	556.97	(15.10)	2.80
	ARM Cortex	87.76	(34.56)	168.17	137.13	(3.72)	87.83

factor of 1.87× for Cuprite and 3× for WTC; and real-time is well beyond question for the Intel Atom and ARM Cortex. With respect to the second question, it is important to note that on the Intel Xeon and AMD Opteron, the combinations that offer the optimal energy consumption in the processing of the Cuprite image (103.81J and 123.16J, respectively) require execution times that are still well below the baseline (1.70s and 0.61s, respectively). In the case of the AMD processor, this also holds for the WTC scene.

On the other hand, on airborne and spaceborne missions, power and/or energy may pose critical limitations on the architecture that can be employed, due, e.g., to the capacity of the PSU to provide a certain wattage or the need to extend the duration of the mission. In this sense, it is important to analyze not only the (nominal) TDP of the CPU (see Table I), but also the maximum power that was observed during the experiments on each platform (see the column labelled as "Max. power" in the same table). These results clearly state the advantage of all the low-power CPUs over the commodity Intel Xeon and AMD Opteron processors. Although temperature measurements were not carried out in our tests, comparative heat dissipation values can be induced from the power dissipation numbers if temperature is a limiting factor.

V. CONCLUDING REMARKS AND FUTURE RESEARCH LINES

We have explored the performance of a multi-core DSP from TI as a low-power architecture to accelerate spectral unmixing methods. Furthermore, we have described a number of techniques to efficiently leverage different forms of hardware concurrency available in this processor, and performed a detailed performance/energy evaluation for two commonly used spectral datasets. The study reveals that these DSPs offer an appealing trade-off between time-to-solution and energy demand to process hyperspectral data compared with alternative state-of-the-art general-purpose and low-power multicore architectures. While, for the test cases considered here, real-time is lost for the TI DSP, power and budget limitations in current and future airborne and spaceborne missions can promote the use of this type of energy-efficient architectures. In addition, the experimental results obtained for different types of architectures can serve as a reference of which architecture is more suitable depending on the time and/or energy restrictions imposed by the target application. Finally, we emphasize that it is possible to use the same DSP architecture to implement other spectral unmixing algorithms, since alternative but similar techniques exist for endmember extraction and abundance estimation. For instance, from the

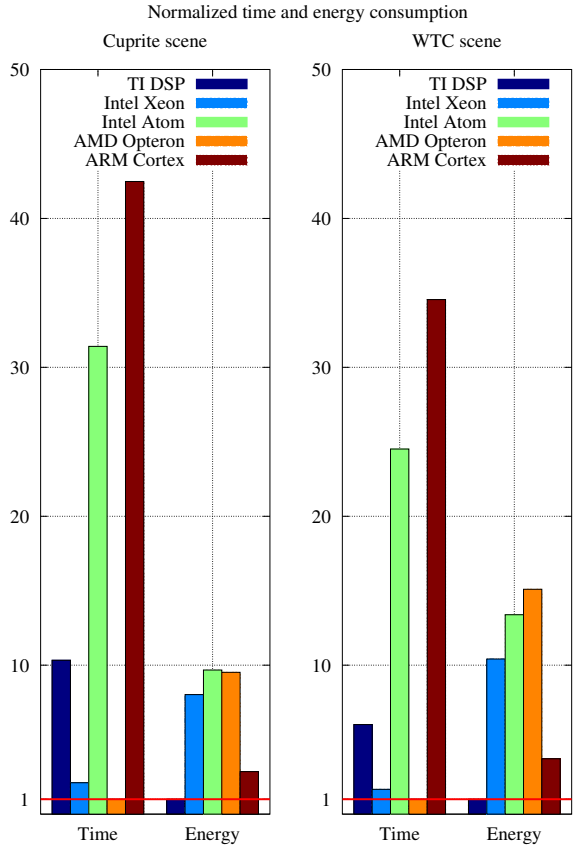

Fig. 2. Comparison of the time and energy (normalized w.r.t. the best architecture in each case) required by the complete spectral unmixing chain.

ISRA implementation it is quite easy to derive other forms of unconstrained and partially constrained abundance estimators. Also, the OSP-GS is similar in implementation to other popular methods for endmember extraction that could be ported to the present architecture with little effort. Despite the fact that the selected algorithms have been shown to be robust in the considered problem, we are currently experimenting with other spectral unmixing algorithms implementations in the considered hardware platform.

ACKNOWLEDGMENTS

The authors from the Universidad Jaume I were supported by project CICYT TIN2011-23283 and FEDER. The author from the Universidad Complutense de Madrid was supported by project CICYT TIN2008-0508. We thank TI for the donation of the DSP processor used in the experimental section. Funding from the Spanish Ministry (CEOS-SPAIN project, reference AYA2011-29334-C02-02) is also gratefully acknowledged.

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] P. E. Johnson, M. O. Smith, S. Taylor-George, and J. B. Adams, "A semiempirical method for analysis of the reflectance spectra for binary mineral mixtures," *J. Geophysical Res.*, vol. 88, pp. 3557–3561, 1983.

[3] J. B. Adams, M. O. Smith, and P. E. Johnson, "Spectral mixture modeling: a new analysis of rock and soil types at the Viking Lander 1 site," *J. Geophysical Res.*, vol. 91, pp. 8098–8112, 1986.

[4] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Processing Magazine*, vol. 19, no. 1, pp. 44–57, 2002.

[5] J. J. Settle and N. A. Drake, "Linear mixing and the estimation of ground cover proportions," *Int. J. Remote Sensing*, vol. 14, pp. 1159–1177, 1993.

[6] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral Imagery," *J. Parallel and Distributed Computing*, vol. 66, no. 3, pp. 345–358, 2006.

[7] A. Plaza and C.-I. Chang, "Clusters versus FPGA for parallel processing of hyperspectral imagery," *Int. J. High Performance Computing Applications*, vol. 22, no. 4, pp. 366–385, 2008.

[8] C. Gonzalez, J. Resano, D. Mozos, A. Plaza, and D. Valencia, "FPGA implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis," *EURASIP J. Advances in Signal Processing*, vol. 969806, pp. 1–13, 2010.

[9] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "FPGA implementation of the N-FINDR algorithm for remotely sensed hyperspectral image analysis," *IEEE T. Geoscience and Remote Sensing*, vol. 50, no. 2, pp. 374–388, 2012.

[10] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado, "GPU for parallel on-board hyperspectral image processing," *Int. J. High Performance Computing Applications*, vol. 22, no. 4, pp. 424–437, 2008.

[11] M. Hsueh and C.-I. Chang, "Field programmable gate arrays (FPGA) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery," *Int. J. High Performance Computing Applications*, vol. 22, pp. 408–423, 2008.

[12] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 508–527, 2011.

[13] A. Plaza, Q. Du, Y.-L-Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 528–544, 2011.

[14] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, "Foreword to the special issue on high performance computing in earth observation and remote sensing," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE J.*, vol. 4, no. 3, pp. 503 –507, sept. 2011.

[15] J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection," *IEEE T. Geoscience and Remote Sensing*, vol. 32, no. 4, pp. 779–785.

[16] M. E. Daube-Witherspoon and G. Muehllehner, "An iterative image space reconstruction algorithm suitable for volume ECT," *IEEE T. Medical Imaging*, vol. 5, pp. 61–66, 1986.

[17] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.

[18] E. Anderson et al, *LAPACK Users' guide*, 3rd ed. SIAM, 1999.

[19] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. Kluwer Academic/Plenum Publishers: NY, 2003.

[20] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE T. Geoscience and Remote Sensing*, vol. 42, no. 3, pp. 650–663, 2004.

[21] C.-I. Chang and D. Heinz, "Constrained subpixel target detection for remotely sensed imagery," *IEEE T. Geoscience and Remote Sensing*, vol. 38, pp. 1144–1159, 2000.

[22] "TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor," February 2012, Texas Instruments Literature #SPRS691C.

[23] M. Ali, E. Stotzer, F. D. Igual, and R. A. van de Geijn, "Level-3 BLAS on the TI C6678 multi-core DSP," in *IEEE 24th Int. Symp. Computer Architecture and High Performance Computing*, 2012, pp. 179 –186.

[24] F. D. Igual, M. Ali, A. Friedmann, E. Stotzer, T. Wentz, and R. A. van de Geijn, "Unleashing the high-performance and low-power of multi-core DSPs for general-purpose HPC," in *Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 26:1–26:11.