Real-Time Identification of Hyperspectral Subspaces

Emanuele Torti, *Member*, *IEEE*, Marco Acquistapace, Giovanni Danese, *Member*, *IEEE*, Francesco Leporati, *Member*, *IEEE*, and Antonio Plaza, *Senior Member*, *IEEE*

***Abstract*—The identification of signal subspace is a crucial operation in hyperspectral imagery, enabling a correct dimensionality reduction that often yields gains in algorithm performance and efficiency. This paper presents new parallel implementations of a widely used hyperspectral subspace identification with minimum error (HySime) algorithm on different types of high-performance computing architectures, including general purpose multicore CPUs, graphics processing units (GPUs), and digital signal processors (DSPs). We first developed an optimized serial version of the HySime algorithm using the C programming language, and then we developed three parallel versions: one for a multi-core Intel CPU using the OpenMP API and the ATLAS algebra library, another one using NVIDIA's compute unified device architecture (CUDA) and its basic linear algebra subroutines library (CuBLAS), and another one using a Texas' multicore DSP. Experimental results, based on the processing of simulated and real hyperspectral images of various sizes, show the effectiveness of our GPU and multicore CPU implementations, which satisfy the real-time constraints given by the data acquisition rate. The DSP implementation offers a good tradeoff between low power consumption and computational performance, but it is still penalized by the absence of double precision floating point accuracy and/or suitable mathematical libraries.**

***Index Terms*—Digital signal processors (DSPs), graphics processing units (GPUs), hyperspectral imaging, hyperspectral signal identification with minimum error (HySime).**

I. Introduction

HYPERSPECTRAL sensors sample the solar radiation, reflected by the Earth's surface, in hundreds of narrow contiguous spectral bands [1]. These devices are characterized by high spectral resolutions, which yield large amounts of data. For instance, the NASA Jet Propulsion Laboratory's Airborne Visible Infrared Imaging Spectrometer (AVIRIS) [2] produces "data cubes" with a typical size of 614 lines with 512 samples and 224 spectral bands, and spectral resolution of 10 nm. This means an approximate amount of 140 MB per data cube. Such a volume puts stringent requirements on data transmission, storage, and processing.

We can represent the values of the spectral radiance, acquired by a hyperspectral sensor from a given pixel as an *L-dimensional* vector, where L is the number of bands. Under the linear mixing hypothesis, the spectral vectors are a linear combination of the endmember signatures, i.e., the spectral signatures of pure constituents [3]. The number of endmembers present in a given scene is, very often, much smaller than the number of bands L. This means that the hyperspectral vectors lie in a low-dimensional subspace [4]. When this subspace is identified, we can represent these vectors in it, with obvious benefits in computational time, complexity, and data storage [5]. Several approaches have been used to perform this operation, among which we can outline methods based on the existing correlation between adjacent bands [6], projection techniques which seek for the best subspace minimizing an objective function [7], [8], or topological methods which infer the manifold where the data set lives [9].

Subspace identification is a critical issue in hyperspectral image processing for various applications such as unmixing, classification, target detection, and recognition [10]. Moreover, subspace identification enables faster elaborations and optimized data storage.

A widely used algorithm for this purpose is the hyperspectral subspace identification with minimum error (HySime) [10], which is based on a minimum square error approach.

As mentioned before, the amount of hyperspectral data collected on a daily basis requires high-performance computing, particularly for scenarios in which real-time response is crucial (for instance, biological threat detection, monitoring of chemical contamination, wildfire tracking, etc.). In recent years, many parallel computing techniques have been used to accelerate hyperspectral imaging applications. In particular, implementations have been already developed on supercomputers [11], clusters [12], [13], and specialized hardware devices, such as graphics processing units (GPUs) [14]–[16]. In this context, the HySime algorithm, together with other approaches for subspace identification, has been implemented in parallel [16]. However, the parallel implementation of HySime has not been analyzed in the context of different high-performance computing platforms, and a study of the differences between the various architectures, applied to a complete implementation of a hyperspectral imaging algorithm, has not yet been carried out. In addition to GPUs, several other hardware accelerators have also been considered such as field-programmable gate arrays (FPGAs) [17]–[19]. However, GPUs are being now widely used in hyperspectral imaging applications due to their high-computational power [15]. Recent works showed that modern multicore DSPs can also offer a good tradeoff between performance and power consumption in intensive matrix calculations [20]. Solutions based on CPUs, GPUs, and FPGA show good performance for what concerns execution times, permitting to achieve real-time elaborations. On the other hand, DSP performances are low but, as said before, they also offer lower power consumption.

Manuscript received June 27, 2013; revised October 24, 2013; accepted January 22, 2014. Date of publication March 30, 2014; date of current version August 01, 2014.

E. Torti, G. Danese, and F. Leporati are with the Dipartimento di Ingegneria Industriale e dell'Informazione, University of Pavia, 27100 Pavia, Italy (e-mail: emanuele.torti01@ateneopv.it; gianni.danese@unipv.it; francesco.leporati@unipv.it).

M. Acquistapace is with Positech Consulting s.r.l., 20123 Milano, Italy (e-mail: marco.acquistapace01@ateneopv.it).

A. Plaza is with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, 10071 Caceres, Spain (e-mail: aplaza@unex.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.2014.2304832

In this paper, we present several new parallel implementations of HySime based on various architectures and software tools: a first version based on a general purpose multicore Intel CPU, using the OpenMP API and the algebra library ATLAS, another one using NVIDIA's compute unified device architecture (CUDA) and its basic linear algebra subroutines library (CuBLAS), and finally an implementation using Texas' multicore software development kit (MCSDK). The reminder of the paper is organized as follows: Sections II and III describe the HySime algorithm for signal subspace identification and our parallel implementations, respectively. Section IV presents the experimental results carried out by using hyperspectral images of various sizes and the comparison between different architectures. Section V concludes the paper with some remarks and hints at possible future research lines.

II. HySime Algorithm

The HySime algorithm consists of two parts: noise and signal subspace estimations. First of all, the method evaluates the signal and the noise correlation matrices. It then selects the set of eigenvectors that better represents the signal subspace in terms of minimum square error. This leads to a minimization of a two-terms objective function, in which one term is represented by the power of the signal projection error, while the other term corresponds to the power of the noise projection. It should be noted that the former is a decreasing function of the subspace dimension, while the latter is an increasing one.

A. Noise Estimation

With the linear mixing hypothesis, it is assumed that the observed spectral vectors in a hyperspectral image, denoted by $\mathrm{y} \in \mathbb{R}^L$, are given by $\mathbf{y} = \mathbf{x} + \mathbf{n}$; where $\mathbf{x}$ and $\mathbf{n}$ are $L - D$ vectors representing the signal and the additive noise. We denote with $\mathbf{Y}$ an $L \times N$ matrix, containing the N observed spectral vectors. We also define the matrix $\mathbf{Z} = \mathbf{Y}^{\mathrm{T}}$ and the $N \times 1$ vectors $\mathbf{z}_{\mathrm{i}} = [\mathrm{Z}]_{:,i}$, where $[\mathrm{Z}]_{:,i}$ indicates the ith row of $\mathbf{Z}$ (i.e., the acquired hyperspectral data regarding the ith band), and the $N \times (L-1)$ matrix defined as $\mathbf{Z}_{\partial_i} = [\mathbf{z}_1, \ldots, \mathbf{z}_{\mathrm{i}-1}, \mathbf{z}_{\mathrm{i}+1}, \ldots, \mathbf{z}_L]$.

The assumption is that the vectors $\mathbf{z}_{\mathrm{i}}$ are a linear combination of the remaining $L-1$ bands. This leads to the following expression:

$$\mathbf{z}_{\mathrm{i}} = \mathbf{Z}_{\partial_i}\boldsymbol{\beta}_i + \boldsymbol{\xi}_i \tag{1}$$

where $\boldsymbol{\beta}_i$ is the regression vector of size $(L-1) \times 1$ and $\boldsymbol{\xi}_i$ is the modeling error vector of size $(N-1)$. For every $i \in \{1, \ldots, L\}$, the least squares estimator of the regression vector $\boldsymbol{\beta}_i$ is given by

$$\widehat{\boldsymbol{\beta}}_i = (\mathbf{Z}_{\partial_i}^{\mathrm{T}}\mathbf{Z}_{\partial_i})^{-1}\mathbf{Z}_{\partial_i}^{\mathrm{T}}\mathbf{z}_i. \tag{2}$$

The noise estimation is given by

$$\widehat{\boldsymbol{\xi}}_i = \mathbf{z}_{\mathrm{i}} - \mathbf{Z}_{\partial_i}\widehat{\boldsymbol{\beta}}_i. \tag{3}$$

Finally, the noise correlation matrix is given by: $\widehat{\mathbf{R}}_{\mathbf{n}} = [\widehat{\boldsymbol{\xi}}_1, \ldots, \widehat{\boldsymbol{\xi}}_N]^{\mathrm{T}}[\widehat{\boldsymbol{\xi}}_1, \ldots, \widehat{\boldsymbol{\xi}}_N]/N$.

TABLE I
Noise Estimation

INPUT $\mathbf{R} \equiv [\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N]$
1: $Z := \mathrm{R}^{\mathrm{T}}$, $\hat{R} = (Z^T Z)$;
2: $R' := \hat{R}^{-1}$;
3: **for** $i := 1$ **to** L **do**
4: $\hat{\beta}_i := \left([R']_{\partial_i,\partial_i} - \frac{[R']_{\partial_i,i}[R']_{i,\partial_i}}{[R']_{i,i}}\right)[\hat{R}]_{\partial_i,i}$
{Note that $\partial_i = 1, \ldots, i\text{-}1, i\text{+}1, \ldots, L$}
5: $\hat{\xi}_i := z_i - Z_{\partial_i}\hat{\beta}_i$;
6: **end for**
OUTPUT $\hat{\xi}$;

Table I shows the pseudocode of the noise estimation algorithm. The symbol $[\widehat{\mathbf{R}}]_{\partial_i,\partial_i}$ indicates the matrix obtained by $\widehat{\mathbf{R}}$ removing the ith row and the ith column. $[\widehat{\mathbf{R}}]_{i,\partial_i}$ denotes the ith row of $[\widehat{\mathbf{R}}]_{:,\partial_i}$, and $[\widehat{\mathbf{R}}]_{\partial_i,i}$ indicates $[\widehat{\mathbf{R}}]_{\partial_i,i}^{\mathrm{T}}$. Steps 1 and 2 calculate the matrix $\widehat{\mathbf{R}} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z}$ and its inverse, respectively. Steps 4 and 5 estimate the regression vector $\widehat{\boldsymbol{\beta}}_i$ and the noise vector $\widehat{\boldsymbol{\xi}}_i$, for every $i = 1, \ldots, L$.

B. Signal Subspace Estimation

The next step of the HySime algorithm is based on the signal estimation procedure. It identifies a set of orthogonal directions, of which an unknown subset spans the signal subspace. This subspace is determined by seeking the minimum mean square error between the original signal $\mathbf{x}$, and a noisy projection obtained by the vector $\mathbf{y} = \mathbf{x} + \mathbf{n}$. Another assumption is that the noise is zero-mean Gaussian distributed with the covariance matrix $\widehat{\mathbf{R}}_{\mathbf{n}}$, i.e., $\mathbf{n} \sim \mathcal{N}(0, \widehat{\mathbf{R}}_{\mathbf{n}})$. As already mentioned, the signal subspace is obtained by minimizing a two-terms objective function, in which one term is represented by the power of the signal projection error, and the other is the power of the noise projection. The minimization is given by [10]

$$(\hat{k}, \hat{\pi}) = \underset{k,\pi}{\operatorname{argmin}}\left\{c + \sum\nolimits_{j=1}^{k} \underbrace{-p_{i_j} + 2\sigma_{i_j}^2}_{\delta_{i_j}}\right\} \tag{4}$$

where $\pi = \{i_1, \ldots, i_L\}$ is the permutation of indices $i = 1, \ldots, L$ and k is the subspace dimension. c is a suitable constant, while p_{i_j} and $\sigma_{i_j}^2$ are quadratic forms given by

$$p_{i_j} = \mathbf{e}_{i_j}^T\widehat{\mathbf{R}}_y\mathbf{e}_{i_j}, \quad \sigma_{i_j}^2 = \mathbf{e}_{i_j}^{\mathrm{T}}\widehat{\mathbf{R}}_{\mathrm{n}}\mathbf{e}_{i_j} \tag{5}$$

where $\mathbf{e}_i$ represents the eigenvalues of the estimated signal correlation matrix $\widehat{\mathbf{R}}_{\mathbf{x}}$.

Table II shows the pseudocode of the signal subspace estimation algorithm. The inputs are the observed spectral vectors, the correlation matrix $\widehat{\mathbf{R}}_{\mathbf{y}}$, and the noise estimation $\widehat{\boldsymbol{\xi}}$. Step 1 estimates the noise correlation matrix $\widehat{\mathbf{R}}_{\mathbf{n}}$. Step 2 estimates the signal correlation matrix $\widehat{\mathbf{R}}_{\mathbf{x}}$. Steps 3 and 4 calculate the eigenvalues of the signal correlation matrix and the δ_i terms based on the quadratic forms (5). Steps 5 and 6 implement the minimization (4). Finally, the signal subspace is obtained.

TABLE II
SIGNAL SUBSPACE ESTIMATION

INPUT $Y \equiv [y_1, y_2, \ldots, y_n], \hat{R}_y = YY^T/N, \hat{\xi}$
1: $\hat{R}_n := \frac{1}{N}\sum_i (\hat{\xi}_i \hat{\xi}_i^T)$;
2: $\hat{R}_x := \frac{1}{N}\sum_i [(y_i - \hat{\xi}_i)(y_i - \hat{\xi}_i^T)]$; {estimate of $\hat{R}_x$}
3: $E := [\mathrm{e}_1, \ldots, \mathrm{e}_L]$; {$\mathrm{e}_i$ are the eigenvalues of $\hat{R}_x$}
4: $\delta := [\delta_1, \ldots, \delta_L]$; {$\delta_i$ given by (4)}
6: $(\hat{\delta}, \hat{\pi}) := sort(\delta)$; {sorts δ_i by ascending order and saves the permutation $\hat{\pi}$}
7: $\hat{k} :=$ number of $\hat{\delta}_i$ terms < 0;
OUTPUT $\hat{X} = \langle [e_{\hat{\imath}_1}, \ldots, e_{\hat{\imath}_{\hat{k}}}] \rangle$; {signal subspace}

TABLE III
PROFILING THE OPERATIONS INVOLVED IN THE PROCESSING OF A HYPERSPECTRAL IMAGE USING THE HySime ALGORITHM

Instructions	Total (s)	Total %
Matrix multiplications, e.g.,: $\hat{\mathrm{R}} = (\mathrm{Z}^{\mathrm{T}}\mathrm{Z})$	2.94	26.4
Noise estimation cycle (step 3 to 6)	7.78	69.9

III. HySime IMPLEMENTATION ON DIFFERENT ARCHITECTURES

A MATLAB implementation of the HySime algorithm can be found online (www.lx.it.pt/~bioucas/code/demo_HySime.zip).

We performed a preliminary algorithm profiling in order to identify the most intensive code parts on an Intel i7 3770 CPU working at 3.40 GHz, with four physical cores corresponding to eight threads; we found out that the bottleneck is represented by the matrix–matrix multiplications (usually performed to calculate covariance matrices) and the additive noise calculation cycle. These results (Table III) were obtained using a simulated hyperspectral image with 200 000 pixels (341 MB). First of all, we developed an optimized C implementation in order to achieve a fair benchmark with respect to the parallel versions, and as a basis for the subsequent implementations. This sequential version does not use any optimized library for linear algebra multiplications. We then developed a parallel version using the Open Multiprocessing (OpenMP) API (http://openmp.org/wp/openmp-compilers/) and the optimized Automatically Tuned Linear Algebra Software (ATLAS) library (www.netlib.org/atals/), and we tested the algorithm on an Intel multicore i7 3370 CPU working at 3.40 GHz. After that, we performed our GPU and DSP implementations. We used the vc100 compiler under Windows and gcc 4.6.3. under Linux, enabling optimizations using several compilation flags such as $-$O3. In the following, we describe some general aspects of the various architectures used, the NVIDIA Fermi GPU architecture and the Texas C66x DSP Core architecture, and our implementations of HySime on them.

A. HySime Implementation on NVIDIA GPUs

The particular structure of GPUs makes them suitable for accelerating highly intensive linear algebra calculations.

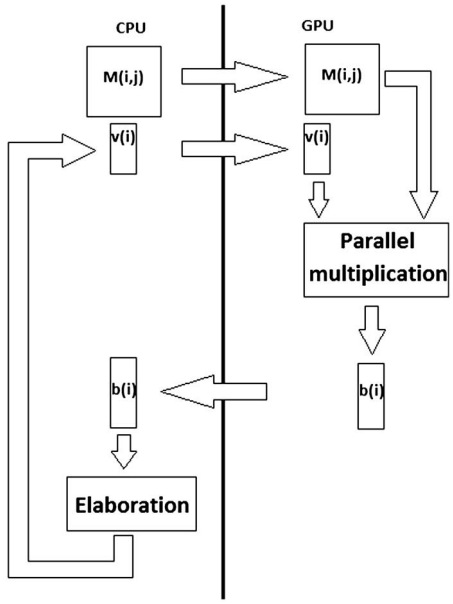

Fig. 1. GPU parallelization of the noise estimation loop in the HySime algorithm.

According to the CUDA programming paradigm, when executing a function (or *kernel*) on the device, one has to allocate enough memory on it, transfer data from the host (CPU) to the GPU and finally transfer data back to the CPU, freeing the device memory. The actual kernel can be either manually defined or an optimized routine, like those offered by the CUBLAS library. Recent works [14] showed that this library offers very good results in terms of execution time, often better than using manual thread dimensioning and assignment. Generally speaking, hyperspectral data cubes fit inside the GPU global memory, so an entire image can be sent to the device memory. Many other matrices involved in our calculations feature the same dimensions or less, so we usually choose to fully send them to the GPU memory, when each calculation is required. The only exception, in our application, is represented by the additive noise cycle, which contains a matrix–vector multiplication. Transferring the matrix to the GPU each time, an iteration is calculated which causes heavy drawbacks in execution times, because of the memory latencies involved. We implemented, then, a custom function that allows the matrix to be maintained in an appropriate section of the device memory for the whole duration of the cycle, since the matrix itself does not change during the iterations. The situation is depicted in Fig. 1. At this point, we emphasize that we cannot use particular GPU optimization techniques such as overlapping memory transfer since the algorithm is strictly sequential.

The computational complexity of HySime shows linear dependence on the number of pixels N, so increasing the image size directly impacts algorithm performance [10]. In particular, the identified intensive sections become increasingly significant as the volume of data to be processed grows up. Moreover to get better execution times even in those code sections not suitable for a GPU execution, we made wide use of OpenMP directives in our implementation, speeding up CPU sections as well. As of today, OpenMP supports only symmetric multiprocessing (SMP) systems, so we could not try out its directives in GPU sections. One

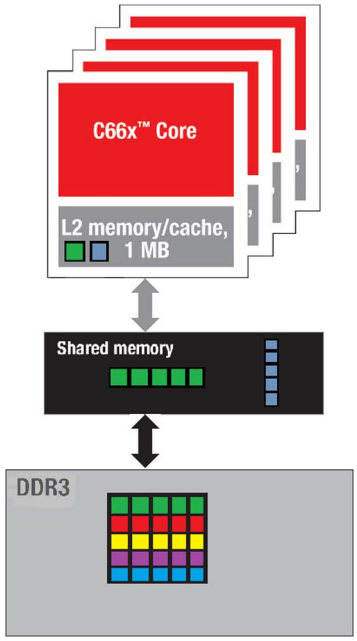

Fig. 2. Data storage and partition in the multicore DSP. The matrix is stored in DDR3 memory and is divided into blocks. Each row (green in the example image) is transferred into shared memory, where there is the vector to be multiplied (blue). Now, the row and the vector are divided into blocks and transferred to the core memory.

last optimization involved two different parallelizing philosophies for the additive noise cycle. Our results showed in fact that, in the case of bigger images, an optimized matrix–vector multiplication CuBLAS function worked better than parallelizing the whole cycle with the use of OpenMP directives. Since we could not use both solutions at the same time for the reason explained above, we chose to introduce an appropriate threshold based on image size.

B. HySime Implementation on TI' Multicore DSP

In our implementation, we used Texas Instruments' Multicore Software Development Kit (MCSDK), which supports the SYS/BIOS operating system and provides integrated common software layers. These layers include configuration and control drivers for peripherals and accelerators.

The main challenges we have faced in our implementation concern DSP configuration and initialization, plus managing I/O operations with the device, particularly with big hyperspectral images. The DSP configuration is stored in a configuration file, written in JavaScript and parsed through TI's XDCtools software. In our application, we had to set a number of parameters regarding OpenMP and memory management.

For example, we must set the appropriate size for both local and shared memory heap, to assign them to the MCSM memory; we also fixed the correct number of processors for OpenMP and manually specified that data of considerable dimensions (like vectors or matrices) had to be stored in DDR3 memory, i.e., the only one with enough space. We also had to enable cache write-back for OpenMP and to set the appropriate OpenMP cache size. Concerning I/O operations, the only practical way to manage big hyperspectral images involves directly transferring the whole matrix into the DSP's DDR3 memory. This meant that the loading operation had to be properly controlled via a special .dat file, with a fixed overhead specifying information like data representation (float or integers), base address, page number, and data size. However, we still cannot test the biggest image sizes due to space constraints even on the DDR3 memory. One major drawback of our DSP implementation regarded the absence of linear algebra routines fully supported by TI. At the date of this work, an optimized porting of the *libflame* linear algebra library was still in experimental state. This fact forced us to write multiplication routines by hand (with the help of OpenMP directives), with a considerable loss in processing performances.

TABLE IV
NVIDIA GPU FEATURES

	GTX 460 SE	GTX 460	GTX 550Ti	GT 540M
CUDA core	288	336	192	96
Clock frequency (MHz)	1300	1350	1800	1344
Available memory (MB)	993 DDR5	993 DDR5	1024 DDR5	1024 DDR3
Memory clock (MHz)	1700	1800	533	900
Memory interface (bit)	256	256	192	128
Memory bandwidth (GB/s)	108.8	115.2	98.4	28.8

At this point, it is also important to emphasize that we parallelized each code block using appropriate *#pragma* directives such as *#pragma omp parallel for* in the noise estimation cycle, adding information about variable scope using *private* and *shared* annotations. As mentioned before, the matrix–vector multiplication is the bottleneck; we chose to store the matrix in the external DDR3. Fig. 2 shows how the matrix and the vector are also divided into sub-blocks and how the data have been partitioned and allocated among the DSP computational cores.

Another performance constraint regarded the double precision requirement: the used DSP is a 32-bit device, so even if it offers better floating point performance than its predecessors, it still has a disadvantage with respect to a 64-bit Intel i7 CPU or a GPU. We shall discuss how this issue affects performance in Section IV.

IV. EXPERIMENTAL RESULTS

A. GPU Architecture

The NVIDIA architecture used in our implementation is code-named "Fermi." It is characterized by 16 streaming multiprocessors (SMs), six 64 bit memory partitions, a two-level cache hierarchy, a PCI express bus connecting the CPU and the GPU, and a global scheduler (named GigaThread) that distributes blocks (i.e., sets of threads) to the SMs [21]. The most important component on the GPU is the SM. Each one features, depending on the release (or "compute capability"), 32 or 48 CUDA cores. Each CUDA core has both an integer ALU and a floating point unit (FPU) and both have a pipelined structure. The Fermi architecture supports the floating point standard IEEE 754-2008 for both single and double precision, providing also a fused multiply–add (FMA) instruction that allows more precise results [21]. This is particularly important for our application,

TABLE V
PROCESSING TIMES OBTAINED THROUGH DIFFERENT IMPLEMENTATIONS OF HYSIME ALGORITHM ON DIFFERENT HYPERSPECTRAL IMAGES

Image size (MB)	MATLAB (s)	OpenMP (s)	ATLAS (s)	GTX 460 SE (s)	GTX 460 (s)	GTX 550Ti (s)	GT 540M (s)
3.4	0.191	0.163	0.147	0.566	0.464	0.313	0.510
17	0.657	0.467	0.293	0.928	0.722	0.609	1.045
34	1.262	0.904	0.550	1.373	1.107	0.890	1.716
85	2.996	2.240	1.169	2.694	2.022	1.841	3.682
128	4.321	3.638	1.795	3.949	2.987	2.761	5.538
170	5.908	5.890	3.730	4.892	3.573	3.416	6.911
341	11.137	19.855	7.383	9.268	6.549	6.443	13.261

TABLE VI
DETAILED PROCESSING TIMES OBTAINED THROUGH DIFFERENT IMPLEMENTATIONS OF HYSIME ALGORITHM (128 MB IMAGE)

Operation	MATLAB (s)	OpenMP (s)	ATLAS (s)	GTX 460 SE (s)	GTX 460 (s)	GTX 550Ti (s)	GT 540M (s)
estAdditiveNoise	2.107	1.643	0.313	1.240	1.231	1.231	1.298
Ry=y*y'/N	0.170	0.400	0.137	0.148	0.141	0.138	0.150
Rx=x*x'/N	0.220	0.406	0.136	0.147	0.142	0.138	0.153
W(i,:)=r(i,:)-beta*r	1.610	0.430	0.460	0.773	0.770	0.750	0.790
Rw=w*w'/N	0.170	0.427	0.136	0.140	0.139	0.135	0.150

since double precision calculation is required (this was experimentally tested). The architecture also features 16 load/store units, which allow each source and destination address to be calculated for 16 threads per block, and four special function units which execute transcendental instructions. Each SM has a set of 32 bit registers and an on-chip 64-kB configurable memory, enabling the programmer to choose between 16 kB of shared memory and 48 kB of cache and vice versa. Memory management is a key aspect in many applications, being often the real challenge that must be faced when programming these devices. The GPU features an off-chip memory (divided in global and local) common to all SMs and an on-chip memory. Each SM has one set of 32 bit registers per core and a shared memory between all the cores. Special read-only constant and texture memories are also available, shared by all cores. The global memory is the biggest one on the GPU and can be read/written by the host and by all threads of a grid, but is characterized by long access latencies (hundreds of clock cycles). The registers are very high-speed on-chip memories, but each SM has only 32 768 registers split among all its blocks and allocated to individual threads. Finally, the GPU has a fast on-chip low latency memory. A suitable portion is used as local cache for each SM, while the remaining is shared among all the SMs.

B. DSP Architecture

The TI C66x DSP architecture family is widely used in embedded applications, thanks to its low power characteristic and the possibility of using standard memory allocation and management techniques along with plain $\mathrm{C/C{+}{+}}$ code. In our implementation, we used a TMS320C6678 DSP based on the KeyStone multicore architecture. This device features eight C66x cores, with single and double precision floating point support. It consumes 10 W at 1 GHz working frequency. A set of peripherals is provided: PCI interface with the CPU, direct memory access between DDR3 memory and on-chip memory, and serial rapid I/O to communicate with DSP devices.

The DSP features a VLIW architecture and allows multiple levels of parallelism.

The data level parallelism is allowed by the SIMD support offered by the instruction set. The device can carry out two multiply–add operations in one cycle, and it is IEEE 754-2008 compliant. Finally, the system can run different threads across different cores, allowing thread-level parallelism. The memory hierarchy is made up of two levels: $32 + 32$ kB of L1 cache (program and data), plus 512 kB of L2 cache. There is also an additional level (4096 kB), the previous mentioned MSMC memory, which is shared between the cores. The DSP runs a lightweight, real-time operating system called SYS/BIOS, and the compiler provided as part of the development package supports $\mathrm{C/C{+}{+}}$ and OpenMP 3.0. It must be noted that, while OpenMP allows explicit control of data coherency, there is not any hardware cache coherency support.

For our tests, we used a Texas Instruments TDMSEVM6678L board equipped with an eight core DSP.

C. Performance Evaluation

At first, we generated a set of hyperspectral images of increasing sizes using a MATLAB script written by Nascimento and Bioucas-Dias [10] for HySime testing. All the images were generated using endmembers signature from USGS Library (1995 version, available online at http://speclab.cr.usgs.gov) and the size of the images from 3.4 to 341 MB. We compared the results obtained by the MATLAB implementation with the results of our parallel codes, using synthetic hyperspectral data, for guaranteeing correctness of our implementations. We also tested our implementation using the well-known AVIRIS Cuprite image collected over the mine district in Nevada (188 bands $\times$ 255 rows $\times$ 191 columns, 34.24 MB).

We tested our CPU, GPU, and CPU/GPU-based implementations (except for the card NVIDIA GT540M) with a PC equipped with an Intel i7 3770 CPU working at 3.40 GHz, with 8 GB RAM. For the tests with NVIDIA GT 540M, we used a laptop

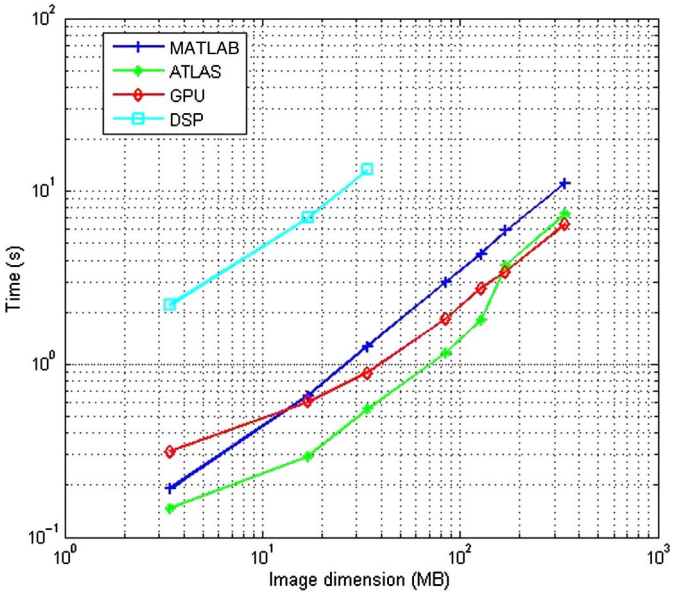

Fig. 3. Different implementations of the HySime algorithm on MATLAB, ATLAS, CPU/GPU, and DSP.

equipped with an Intel i7 2630QM CPU working at 2.0 GHz, with 8 GB RAM. The different GPUs features are summarized in Table IV.

In both cases, the ATLAS library, the CUDA development environment 4.1, and the CUBLAS library were installed.

The tests were performed on different operating systems: a 64-bit Windows 7 and a 64-bit Ubuntu 12.10 running Linux kernel 3.5. In all the cases, Linux was faster than Windows accelerating the computations from 10% to 30%; this is due to more efficient memory transfers and threads management capability.

Table V shows the execution times of our implementations related to simulated images size, whereas Table VI shows detailed execution times for each operation performed in a parallel way while elaborating an image of 128 MB.

We report only Linux execution times. Moreover for obtaining even better results, we include also some OpenMP directives for parallelizing some code blocks that cannot be executed on GPUs.

The reported times are obtained as the mean of multiple executions, with a relative standard deviation less than 3%.

OpenMP implementation is faster than MATLAB except for the last test, for which we used a big image. In this case, MATLAB, which uses extremely optimized routines, performs better.

Our ATLAS implementation is faster than MATLAB in all the performed tests, because this library has been fully optimized for our architecture at compilation time. Notice that it is even faster than our GPU implementation for tests with smaller images. For the other tests, three GPUs (GTX 460 SE, GTX 460, and GTX 550Ti) perform better than other implementations. This is due to the fact that with smaller images memory transfers needed by GPUs are the system bottleneck, while with bigger images the transfer latencies are masked by the speed up achieved by GPU computations. A comparison of the MATLAB, ATLAS, DSP, and best GPU implementations is shown in Fig. 3.

For the DSP, we tested our implementation only with three small simulated images, because there are limitations on memory transfers allowed in the version of the development kit used. On the other hand, Cuprite has been elaborated on all the previous described devices and the different execution times are summarized in Table VII.

TABLE VII
PROCESSING TIMES FOR CUPRITE IMAGE

Implementation	Total time (s)
MATLAB	1.310
OpenMP	0.911
ATLAS	0.549
GTX460SE	1.380
GTX460	1.310
GTX550Ti	0.890
GT 540M	1.719
DSP TMS320C6678	13.321

TABLE VIII
PROCESSING TIMES OBTAINED THROUGH DSP IMPLEMENTATIONS OF HYSIME ALGORITHM ON DIFFERENT HYPERSPECTRAL IMAGES

Image size (MB)	Total time (s)
3.4	2.190
17	6.983
34	13.244

TABLE IX
TDP OF THE DIFFERENT USED TECHNOLOGY

Device	TDP (W)
DSP TMS320C6678	10.5
Intel i7 2630QM	45
Intel i7 3770	77
NVIDIA GT540M	35
NVIDIA GTX550Ti	116
NVIDIA GTX460	140
NVIDIA GTX460SE	160

On the other hand, execution times measured in the DSP are shown in Table VIII. These results are slightly worse when compared to the other presented implementations; this is probably due to the fact that code routines have been manually written, since there are no stable linear algebra routines at this time. Moreover, this is a 32-bit DSP and the computations involved in HySime require double precision floating point accuracy, which is obtained using software routines.

Concerning power consumption, Table IX shows the thermal design power (TDP), which can be found on devices data sheets, of each technology used for software implementations. These data show that DSP uses the less power consuming technology, so it is a promising solution for embedded computation, but only if a 64-bit device architecture and linear algebra routines will be available. In practice, it consumes one order of magnitude less than desktop CPU and GPU.

The TDP of the CPUs is not so different and is significantly lower than the one for GPUs. However, GPU solutions perform better than CPU solution particularly for the processing of big images.

Finally, it is worth noting that the processing time achieved by OpenMP, ATLAS, and GPUs solutions meets the real-time processing constraint. While the AVIRIS scanning rate is 12 Hz, more recent instruments such as Hyperion [22] or Environmental Mapping and Analysis Program [23] are able to perform 220 and 230 Hz scanning rate, respectively, so a typical data cube of 614×512 pixels and 224 bands, which means about 140 MB of data, can be collected in 5 s. These time constraints are totally satisfied by our OpenMP, ATLAS, and CPU/GPU implementations, except for NVIDIA GT540M and DSP, as explained before in Table V.

We also took into account three different performance indices for comparing the different implementations. The first index is inversely proportional to the product between execution time and power consumption. The second and the third indices are similar to the first one, except for the weight given, respectively, to the execution time and to the power consumption. In the second index the time is squared, whereas in the third index the power consumption is squared. Using the first two indices the best solutions are the multicore implementations, while using the third index the better solutions are offered by both the DSP and GPU.

To conclude, we emphasize that the works in [16], [24] also provided several parallel implementations of different hyperspectral unmixing algorithms. Specifically, in [24], different algorithms are parallelized and parallel implementations are tested on NVIDIA Tesla C1060 GPU, an NVIDIA GTX 580 GPU, two Intel CPU, an i7, and a Xeon, while in our work we implemented different GPU and CPU parallel implementations, testing them on four different GPU, two different Intel i7 CPU, and a DSP. If we compare our results with the ones presented in [16], which implemented HySime and another hyperspectral subspace estimation algorithm on GPUs, we can conclude that our results are similar to those reported in that paper, but obtained on a 300 cores less GPU device. This indicates the effectiveness of the GPU implementation conducted in this work.

V. Conclusion

In this paper, we presented different real-time implementations of a popular technique for subspace identification in hyperspectral data called HySime. Our implementations have been tested on multiple architectures, mainly NVIDIA Fermi architecture-based GPUs, Intel i7 multicore CPUs, and a TI C6678 DSP. Our results show that the GPU and multicore CPU versions are both faster than the original MATLAB code. These results also suggest that employing these technologies on airborne platform would be more effective, in terms of costs and performance, than the current quadcore boards, especially with the use of future GPU generations which will further reduce energy consumption rates.

The multicore CPU version offered good performance, combined with high flexibility and a simpler development process, while the GPU version provided better performances especially with higher image dimensions, representing the better all-around solution.

Unfortunately due to the strictly sequential nature of HySime, it is not possible to introduce further optimizations, such as memory transfer overlapping.

The DSP implementation, instead, still suffers from factors like the absence of a fully supported linear algebra optimized library and more difficult I/O operations with the device in case of big images. However, the power consumption comparisons still indicate that, when this kind of libraries will get fully supported by TI, the DSP performances would become more interesting considering a tradeoff between consumption and performance. The use of more DSP cores together with MPI would be a feasible future research development to improve performances.

We are currently developing an HySime implementation for field programmable gate arrays (FPGAs). This approach is more challenging than the others presented but preliminary results show better performance than the corresponding CPU/GPU instruction, with very low power consumption (and a lower sensitivity to space radiations when compared to GPUs). This encourages us in the exploitation of FPGA technology for embedded real-time implementation of HySime algorithm. Moreover, recent works based on FPGA implementations of hyperspectral image processing algorithms [25] showed good performance and reliability and are worth being explored for comparisons.

References

[1] T. M. Lillesand, R. W. Kiefer, and J. W. Chipman, in *Remote Sensing and Image Interpretation*, 5th ed. Hoboken, NJ, USA: Wiley, 2004.

[2] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis, M. R. Olah, and O. Williams, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer," *Remote Sens. Environ.*, vol. 65, no. 3, pp. 227–248, Sep. 1998.

[3] N. Keshava and J. Mustard, "Spectral unmixing," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 44–57, Jan. 2002.

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

[5] D. Landgrebe, "Hyperspectral image data analysis," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 17–28, Jan. 2002.

[6] C. Chang and S. Wang, "Constrained band selection for hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 44, no. 6, pp. 1575–1585, Jun. 2006.

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

[8] L. L. Scharf, *Statistical Signal Processing, Detection Estimation and Time Series Analysis.* Reading, MA, USA: Addison-Wesley, 1991.

[9] J. Bruske and G. Sommer, "Intrinsic dimensionality estimation with optimally topology preserving maps," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 5, pp. 572–575, May 1998.

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

[11] A. Plaza, D. Valencia, J. Plaza, and C.-I. Chang, "Parallel implementation of endmember extraction algorithms for hyperspectral data," *IEEE Geosci. Remote Sens. Lett.*, vol. 3, no. 3, pp. 334–338, Jul. 2006.

[12] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *J. Parallel Distrib. Comput.*, vol. 66, no. 3, pp. 345–358, Mar. 2006.

[13] W. Luo, "Parallel VCA algorithm for hyperspectral remote sensing image in SMP cluster environment," in *Proc. CISP,* 2010, pp. 2216–2220.

[14] A. Barberis, G. Danese, F. Leporati, A. Plaza, and E. Torti, "Real-time implementation of the vertex component analysis algorithm on GPUs," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 2, pp. 251–255, Mar. 2013.

[15] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 528–544, Sep. 2011.

[16] S. Sanchez and A. Plaza, "Fast determination of the number of endmembers for real-time hyperspectral unmixing on GPUs," *J. Real-Time Image Process.*, Special issue, published online: Oct. 2012.

[17] M. Hsueh and C.-I. Chang, "Field programmable gate arrays (FPGA) for pixel purity index using blocks of skewers for endmembers extraction in hyperspectral imagery," *Int. J. High Perform. Comput. Appl.*, vol. 22, no. 4, pp. 408–423, Nov. 2008.
[18] A. Gonzalez, J. Resano, D. Mozos, A. Plaza, and D. Valencia, "FPGA implementation of the PPI algorithm for remotely sensed hyperpsectral image analysis," *EURASIP J. Adv. Signal Process.*, vol. 2010, no. 68, pp. 1–13, Feb. 2010.
[19] S. Lopez, T. Vladimirova, C. Gonzales, J. Resano, D. Mozos, and A. Plaza, "The Promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends," *Proc. IEEE*, vol. 101, no. 3, pp. 698–722, Mar. 2013.
[20] M. Castillo, J. Fernandez, E. Quintana-Orti, A. Remon, and F. Igual, "Hyperspectral Unmixing on Multicore DSPs: Trading off Performance for Energy," *J. Sel. Topics Appl. Earth Observ. Remote Sens.,* 2014, to be published.
[21] NVIDIA Corporation, *NVIDIA's Next Generation CUDA Compute Architecture: FERMI,* 2009, whitepaper.
[22] J. S. Pearlman, P. S. Barry, C. C. Seagal *et al.*, "Hyperion, a space-based imaging spectrometer," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 8, pp. 2435–2445, Aug. 2008.
[23] T. Stuffler, K. Foster, S. Hofer, M. Leipold, B. Sang, H. Kaufmann, B. Penné, A. Mueller, and C. Chlebek, "Hyperspectral imaging – An advanced instrument concept for the EnMAP mission," *Acta Astronaut.*, vol. 65, no. 7–8, pp. 1107–1112, Oct./Nov. 2009.
[24] S. Barnebe, S. Sanchez, A. Plaza, S. Lopez, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, Jun. 2013.
[25] S. Lopez, P. Horstrand, G. M. Callico, J. F. Lopez, and R. Sarmiento, "A novel architecture for hyperspectral endmember extraction by means of the modified vertex component analysis (MVCA) algorithm," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 6, pp. 1837–1848, Dec. 2012.

Emanuele Torti (M'13) was born in Voghera, Italy, in 1987. He received the B.S. degree in electronic engineering and M.S. degree in computer science engineering (*cum laude*) from the University of Pavia, Pavia, Italy, in 2009 and 2011, respectively, where he is currently pursuing the Ph.D. degree in computer science engineering.

His research is focused on high-performance architectures for real-time image processing and signal elaboration.

Marco Acquistapace was born in Pavia, Italy, in 1986. He received the B.S. and M.S. degrees in computer science engineering from the University of Pavia, Pavia, Italy, in 2010 and 2013, respectively.

Currently, he is working as a Consultant in the automotive field, in particular for Magneti Marelli S.p.A. in the Research & Development Department. His work consists in developing software for automotive clusters, focusing especially in interfacing the clusters' GPUs and graphical software modules with underlying software layers. His scientific interests involve developing innovative embedded software applications, studying the performance bottlenecks in current platforms and architectures, remote sensing, and image processing.

Giovanni Danese (M'99) received the Ph.D. degree in electronics and computer engineering from the University of Pavia, Pavia, Italy.

He is a Full Professor with the Computer Programming and Computer Architecture in the Engineering Faculty at the University of Pavia. His current research interests include parallel computing, special-purpose computers, and signal and image processing.

Francesco Leporati (M'98) received the Ph.D. degree in electronics and computer engineering from the University of Pavia, Pavia, Italy.

He is an Associate Professor with the Industrial Data Processing and Computer Architecture in the Engineering Faculty at the University of Pavia. His current research interests include automotive applications, FPGA and application-specific processors, embedded real-time systems, and computational physics.

Prof. Leporati is a member of the Euromicro Society and Associate Editor of Microprocessors and Microsystems.

Antonio Plaza (SM'07) is an Associate Professor (with accreditation for Full Professor) with the Department of Technology of Computers and Communications, University of Extremadura, Caceres, Spain, where he is the Head of the Hyperspectral Computing Laboratory (HyperComp).

Prof. Plaza is an Associate Editor for IEEE Access and the IEEE Geoscience and Remote Sensing Magazine, and was a Member of the Editorial Board of the IEEE Geoscience and Remote Sensing Newsletter (2011–2012) and a Member of the Steering Committee of the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (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). Currently, he is serving as the Editor-in-Chief of the IEEE Transactions on Geoscience and Remote Sensing Journal (since January 2013).