Efficient Implementation of Hyperspectral Anomaly Detection Techniques on GPUs and Multicore Processors

José M. Molero, Ester M. Garzón, Inmaculada García,
Enrique S. Quintana-Ortí, and Antonio Plaza, *Senior Member, IEEE*

***Abstract*—Anomaly detection is an important task for hyperspectral data exploitation. Although many algorithms have been developed for this purpose in recent years, due to the large dimensionality of hyperspectral image data, fast anomaly detection remains a challenging task. In this work, we exploit the computational power of commodity graphics processing units (GPUs) and multicore processors to obtain implementations of a well-known anomaly detection algorithm developed by Reed and Xiaoli (RX algorithm), and a local (LRX) variant, which basically consists in applying the same concept to a local sliding window centered around each image pixel. LRX has been shown to be more accurate to detect small anomalies but computationally more expensive than RX. Our interest is focused on improving the computational aspects, not only through efficient parallel implementations, but also by analyzing the mathematical issues of the method and adopting computationally inexpensive solvers. Futhermore, we also assess the energy consumption of the newly developed parallel implementations, which is very important in practice. Our optimizations (based on software and hardware techniques) result in a significant reduction of execution time and energy consumption, which are keys to increase the practical interest of the considered algorithms. Indeed, for RX, the runtime obtained is less than the data acquisition time when real hyperspectral images are used. Our experimental results also indicate that the proposed optimizations and the parallelization techniques can significantly improve the general performance of the RX and LRX algorithms while retaining their anomaly detection accuracy.**

Manuscript received February 04, 2014; revised April 27, 2014; accepted May 26, 2014. Date of publication July 08, 2014; date of current version August 01, 2014. This work was supported in part by the Spanish Ministry of Science and Innovation under Grant TIN2008-01117, Grant TIN2011-23283, Grant TIN2012-37483, and Grant AYA2011-29334-C02-02, Junta de Andalucia (P10-TIC-6002) and Junta de Extremadura (PRI09A110 and GR10035) and in part by the European Regional Development Fund (ERDF). This work was developed in the framework of the network High Performance Computing on Heterogeneous Parallel Architectures (CAPAP-H4), supported by the Spanish Ministry of Science and Innovation under Grant TIN2011-15734-E. The work of E. S. Quintana-Ortí was supported by the Project CICYT TIN2011-23283 and FEDER.

J. M. Molero and E. M. Garzón are with the Department of Informatics and Agrifood Campus of International Excellence (CEIA3), University of Almería, 04120 Almería, Spain (e-mail: jmp384@ual.es; gmartin@ual.es).

I. García is with the Department of Computer Architecture, University of Málaga, 29071 Málaga, Spain (e-mail: igarciaf@uma.es).

E. S. Quintana-Ortí is with the Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12071 Castellón, Spain (e-mail: quintana@uji.es).

A. Plaza is with the Hyperspectral Computing Laboratory (HyperComp), Department of Technology of Computers and Communications, University of Extremadura, 10071 Cáceres, 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.2014.2328614

***Index Terms*—Anomaly detection, energy consumption, graphics processing units (GPUs), hyperspectral imaging, multicore processors.**

I. Introduction

A hyperspectral image can be considered as a stack of images where each image (or spectral band) represents a wavelength of the electromagnetic spectrum—normally covering the visible and near infra-red regions—and each pixel has an associated spectral signature or *fingerprint* that uniquely characterizes the underlying objects [1]–[3]. These images are composed of a large number of narrow spectral *bands* (b). In spatial terms, the images comprise a number of columns or *samples* (s) and a number of rows or *lines* (l) [4], [5].

Anomaly detection [6] is an important task for hyperspectral data exploitation. An anomaly detector identifies spectral signatures, which are spectrally distinct from their surroundings without prior knowledge [7], [8]. Anomalies can be considered as a set of isolated pixels with anomalous signatures (when compared to the image background), which represent a very small piece of the full image, and they only occur in the image with low probabilities [9]–[11]. A classic example of anomaly detector is the RX algorithm, developed by Reed and Xiaoli [12], which has been widely used in several different hyperspectral imaging applications. This method has shown great success and is usually adopted as a benchmark for anomaly detection purposes [13].

A variant of the RX algorithm is the local RX (LRX), which consists in applying the same concept to a sliding window centered around each image pixel. These type of variations can be used to detect small anomalies [7]. LRX has shown great success to detect small size and subpixel anomalies, improving the results of the RX algorithm as demonstrated in previous works [14], [15]. However, this kind of local methods is computationally more expensive than the original RX because they involve the computation of the RX filter for every local window as opposed to the standard RX, which calculates the filter for the whole image [15]–[17]. In summary, a local method offers better accuracy and detection results for small anomalies at the expense of increased computational cost.

Compared with RX, LRX exhibits a very large computational burden, which generally limits its practical application [9]. However, the use of high-performance computing (HPC) platforms and the design of appropriate optimizations can turn LRX

1939-1404 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

into a reliable and accurate alternative with a reasonable response time. However, this is not a simple task and requires a deep knowledge of both the algorithm and the architecture [18]–[20]. So far, very few works in the literature have reported a real-time processing performance for the RX algorithm or its local versions. Here, by real-time performance, we refer to the fact that the processing can be performed without delay as the data are collected (but not necessarily immediately after the data are collected). In the following, we will avoid the term real-time processing, and refer to processing concurrently or in parallel with data collection [21], [22].

Current software and hardware advances in HPC offer an unprecedented opportunity to significantly reduce the runtimes for hyperspectral applications [19], [23]–[25]. In the particular case of anomaly detection, the development and implementation of optimized and parallel algorithms allow nowadays to explore variations of these algorithms and new theoretical methods, which present a high computational cost [9], [24], obtaining the results in a reasonable processing time using recent multicore processors [14], [15] and modern graphics processing units (GPUs) architectures [26]–[29].

In the previous work, several optimizations and parallel implementations were developed for the RX and LRX algorithms on different HPC platforms. Concretely, the RX algorithm has been implemented on a heterogeneous cluster in [30]; a local version of RX, expressed in terms of the pseudoinverse, was proposed and parallelized on a multicore platform in [14]. Moreover, novel implementations of both algorithms have been investigated on a multicore platform in [15], where the mathematical formulation of LRX is revisited and its computational cost is reduced. The novelties in [15] consisted of: 1) including a recurrence relation for the correlation matrix; and 2) the use of the QR method to decompose the correlation matrix instead of computing the inverse or pseudoinverse. The experimental analysis carried out in [15], using both synthetic and real hyperspectral datasets, revealed the great success of LRX for detecting small-size anomalies, as well as discussed the effectiveness (detection accuracy) and efficiency on multicore platforms of RX and LRX methods. Focusing on the acceleration of the RX algorithm on a GPU platform, [31] describes a GPU implementation of the standard RX algorithm based on the inverse of the correlation matrix, which is evaluated and compared to other detection methods. However, GPU implementations of LRX have never been discussed in the past. Further, a detailed analysis of the energy consumption of both RX and LRX, which is relevant in terms of their practical exploitation, has not been conducted in the previous developments.

Reducing the runtime and exploiting the HPC architectures is quite important in terms of the energy needed to process hyperspectral images. With this goal, we have developed new parallel implementations tailored to multicore and GPU architectures. This paper thus introduces new optimized versions of RX and LRX based on the Cholesky decomposition of the correlation matrices, which result in computationally cheaper approaches than the solutions based on the QR method. Specifically, parallel implementations have been developed to exploit several forms of acceleration, with the ultimate goal of achieving similar times for the processing and data acquisition stages for RX, and explore the practical interest of its local version LRX. In this way, the in-depth study of these algorithms, the use of efficient and adequate algebraic operations, and the application of HPC techniques can result in efficient of algorithms for anomaly detection. Therefore, our aim in this work is threefold: 1) we provide new mathematical expressions for computing RX and LRX based on the Cholesky decomposition; 2) these new versions are implemented on two HPC platforms: multicore systems and GPUs; and 3) the newly developed multicore and GPU implementations are compared in terms of performance (run-time), and efficiency (energy consumption) using real hyperspectral datasets.

This paper is organized as follows. Section II briefly describes the RX algorithm and its local variant (LRX). Section III introduces the sequential and parallel optimizations studied and developed for the RX and LRX algorithms. Section IV describes the real hyperspectral datasets used in our experimental evaluation and the results obtained in terms of parallel performance and energy efficiency. Finally, Section V summarizes the most relevant conclusions and discusses future work.

II. Anomaly Detection Based on the RX Algorithm

A. RX and Local RX Algorithms

The RX algorithm has been widely used in hyperspectral signal and image processing [12]. A variant of this algorithm [7], [13] replaces the covariance matrix by the sample correlation matrix and removes the mean vector from each b-dimensional hyperspectral pixel vector $\mathbf{x} = [x^{(0)}, x^{(1)}, \dots, x^{(b)}]$. This modification represents an adaptation of the original RX to online analysis scenarios, without penalization on its ability to detect anomalies.

The RX algorithm can be considered as a global anomaly detector because the correlation matrix ($\Re$) is computed using all the pixels of the image. However, local versions of RX consider that each pixel of the image has its own correlation matrix $\Re$, which is computed by just considering a small region of pixels around the pixel under test. This implementation, named LRX, uses the concept of a sliding local window [9], [14]. Specifically for each pixel $\mathbf{x}$, the LRX filter (and the corresponding correlation matrix) is computed using a local square window of size $\kappa \times \kappa$ pixels, centered at pixel $\mathbf{x}$. Consequently, the RX and LRX filters can be written as

$$\delta(\mathbf{x}) = \mathbf{x}^T \Re^{-1} \mathbf{x}$$

where the correlation matrix $\Re$ is defined as follows:

$$\Re = \begin{cases} \mathbf{R} = \frac{H^T \cdot H}{n} & \text{for the RX algorithm, and} \\ \mathbf{R}_\kappa(\mathbf{x}) = \frac{H_\kappa^T \cdot H_\kappa}{n_\kappa} & \text{for the LRX algorithm} \end{cases}$$

where H is the hyperspectral image, i.e., $H^T = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$, with $n = l \times s$ pixel vectors, and the subscript κ indicates the pixels included in the local sliding window for the LRX algorithm. Notice that the anomaly detection results can be simply visualized as a grayscale image where each pixel value represents its probability of being an anomalous pixel [3].

B. Optimizations of the RX and Local RX Algorithms

In this section, we describe the optimizations applied to the anomaly detection algorithms. In order to assess the computational benefits of there optimizations, we introduce the term *flops*, which counts the number of floating-point arithmetic operations implied in a computational calculation [32]. Bearing in mind the previous descriptions, two stages can be identified in the computation of both RX and LRX:

a) Stage 1. Evaluation of the correlation matrix $\Re$*:* This computation can be simplified/accelerated taking into account: 1) the symmetry of the correlation matrix, since it is not necessary to compute the entire matrix but only its lower or upper triangle, dividing the cost of this step by 2; 2) the correlation can be expressed in terms of a fast matrix–matrix product; and 3) in the local version, the recurrence relation between the correlation matrices for two contiguous pixels can be exploited to reduce the computational cost by a factor of κ. All summed up, the complexity of this stage without optimizations is $(b^2 \cdot s \cdot l)$ flops for RX and $(b^2 \cdot \kappa^2 \cdot s^2 \cdot l^2)$ flops for LRX, which is reduced with the optimizations previously described to $(\frac{b^2 \cdot s \cdot l}{2})$ flops for RX and $(\frac{b^2 \cdot \kappa \cdot s^2 \cdot l^2}{2})$ flops for LRX [15], [29].

b) Stage 2. Computation of the filter $\delta(\mathbf{x}) = \mathbf{x}^T \Re^{-1} \mathbf{x}$*:* There are different approaches to compute $\delta(\mathbf{x})$. The classic one first computes the inverse (or pseudoinverse) matrix $\Re^{-1}$ followed by the evaluation of the product $\mathbf{x}\Re^{-1}\mathbf{x}$ [9], [30]. However, in this work, we propose an alternative approach to reduce the high cost of explicitly computing the inverse matrices based on the Cholesky factorization and the solution of subsequent linear systems [15]. This implementation of RX and LRX, based on solving the linear systems, is superior not only in terms of performance [33] but also has numerical advantages because the instabilities caused by ill-conditioned correlation matrices can be better tackled by solving the linear equations [34]. At this point, it is important to emphasize that solving the linear system is cheaper than calculating the inverse of the matrix. In our case, this advantage is more relevant because, as the correlation matrix $\Re$ is symmetric and positive-definite, an efficient solver based on the Cholesky factorization can be applied [33], [35]. Given the Cholesky factorization $\Re = U^T U$, his approach computes $\delta(\mathbf{x})$ as follows:

$$\begin{aligned}\delta(\mathbf{x}) &= \mathbf{x}^T \Re^{-1} \mathbf{x} = \mathbf{x}^T (U^T U)^{-1} \mathbf{x} = \mathbf{x}^T U^{-1} U^{-T} \mathbf{x} \\ &= (U^{-T}\mathbf{x})^T (U^{-T}\mathbf{x}) = \mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}).\end{aligned}$$

Therefore, this method based on the Cholesky factorization consists of the following steps: 1) decompose the correlation matrix $\Re$ into a product of two triangular matrices $\Re = U^T U$; 2) solve the triangular system $U^T \mathbf{z}(\mathbf{x}) = \mathbf{x}$ for the intermediate vector $\mathbf{z}(\mathbf{x})$; and 3) $\delta(\mathbf{x})$, which represents the probability of a pixel to be anomalous, is computed by the evaluation of the norm of the vector $\mathbf{z}(\mathbf{x})$ [21], [22]. Therefore, the symmetry of $\Re$ allows us to compute $\delta(\mathbf{x})$ by solving only one triangular system of equations to determine $\mathbf{z}(\mathbf{x})$ due to this particular property of Cholesky factorization. Considering that the correlation matrix is square, symmetric and positive definite, the Cholesky decomposition is the most efficient way (in fact twice more efficient than the LU decomposition and four times more than the QR decomposition discussed in previous works) for solving this type of linear systems [14]. Bearing in mind these considerations the complexity of this stage is $(\frac{b^3}{3})$ flops for RX and $(\frac{s \cdot l \cdot b^3}{3})$ flops for LRX, while the complexity without this optimization, only for the inverse matrix is (b^3) flops for RX and $(s \cdot l \cdot b^3)$ flops for LRX (more details about the complexity can be found in [33, Sect. 2.9] and [35, Sec. 4.2.5]). Moreover, it is worth pointing out that these three phases can be merged into the same iterative process used to compute $\delta(\mathbf{x})$.

III. Parallelization of RX and Local RX Algorithms

This section describes in detail the parallel implementations of RX and LRX, which incorporate all the afore-mentioned optimizations. These parallel algorithms have been developed for two different target architectures: a multicore processor and a manycore GPU. These architectures are quite different but appropriate for processing hyperspectral images, as they both achieve similar performances, reduce the total runtime of the algorithms, and save energy with respect to the sequential executions [28], [36], [37]. One additional advantage of these platforms is that they can be used to tackle large hyperspectral images and experiment with new algorithms of high computational cost [15], [24], [25], [28], [38].

A. Multicore Implementations

Modern multicore processors provide an inexpensive and widely available technology for HPC, which represents a source of computational power to optimize the performance and the energy consumption of anomaly detection algorithms for hyperspectral data [32], [39]. Currently, the number of cores for the most popular processors ranges from 2 to 8 and their design includes energy-saving mechanisms. Therefore, our interest is focused on exploiting this kind of architectures in anomaly detection.

There exist several programming tools such as OpenMP and POSIX threads to exploit concurrency on a multicore processor. Besides, there exists a wide variety of linear algebra libraries optimized for these architectures (Intel MKL,[1] LAPACK,[2] BLAS,[3] etc.) which can be leveraged to exploit the inherent parallelism of the algebraic computations [40]–[42].

In this work, the multicore implementations of RX and LRX leverage OpenMP as the parallelization programming interface. The strategies for distributing the workload of RX and LRX are different. The workload distribution for RX is compute-oriented while LRX load distribution is data-oriented, with the image being distributed among the cores by groups of lines.

Our approaches to optimize the computation of matrix $\Re$ have been presented in detail in the previous work [14], [15]. We next describe the improvements and new considerations of the multicore implementation introduced in this work. The above-described stageshave been parallelized and accelerated using the Intel MKL library [41]. For the calculation of the correlation

[1]Available: http://software.intel.com/en-us/intel-mkl.

[2]Available: http://www.netlib.org/lapack.

[3]Available: http://www.netlib.org/blas.

matrices (Stage 1), the `gemm` function was used for RX, and a specific routine, which takes advantage of the relation between the matrices associated to neighbor pixels was developed for the LRX algorithm. For the factorization of the correlation matrix (Stage 2), routine `potrf` has been used for the Cholesky factorization; and routine `trtrs` for the triangular system with multiple right-hand sides in the case of RX algorithm, and routine `trsv` for the triangular system with single right-hand side in LRX, obtaining in both cases the intermediate vector $z(\mathbf{x})$ for each pixel. For the inner product for $z(\mathbf{x})$, the `dot` function was used. This implementation is detailed in Algorithm 1.

Algorithm 1. Pseudocode of the multicore implementation of RX algorithm. Parallelization has been carried out using the multithreaded library Intel MKL.

```
$H = Load\ Hyperspectral\ Image$
$R = H^T \cdot H$ {matrix-matrix product using gemm function}
$U = Cholesky\ factorization\ (R)$ {Cholesky fact. R matrix using potrf function}
$z = Solve\ linear\ systems\ (U, H)$ {solve triangular system with multiple right-hand sides using trtrs function}
for $i = 1$ to $l$ do {paralellized by OpenMP threads}
    for $j = 1$ to $s$ do
        $\delta[i, j] = z \cdot z$ {compute RX filter using dot function}
    end for
end for
```

B. GPU Implementations

GPU constitute an alternative HPC platform considered in our work. Nowadays, NVIDIA GPUs are widely used in scientific computation [39], [43]. NVIDIA has developed CUDA[4] as a standard GPU application programming interface to ease programming of this kind of architectures. A GPU is composed of hundreds of simple cores organized as a set of stream multiprocessors. According to the CUDA model, each GPU routine, called "kernel," is executed by a batch of threads organized as a grid of thread blocks whose configuration is defined by the programmer. The computation corresponding to each CUDA block is executed in one stream multiprocessor sharing the local memory [43]–[46].

In general, GPU architectures require large-scale problems to deliver high performance. There are optimized linear algebra CUDA libraries (CUBLAS,[5] CULA,[6] MAGMA,[7] etc.). CULA and MAGMA provide a GPU implementation of the blocked Cholesky factorization, but these implementations have been designed to accelerate a single solve and are especially tuned to deal with very large dense matrices [47].

The aforementioned libraries achieve high performance only for very large matrices because all threads collaborate to compute the operation. However, this strategy is not optimal because in LRX it is necessary to solve hundreds of systems of small to moderate size. Thus, our anomaly detection algorithms require a different parallelization strategy.

This kind of computation is often referred to as batched processing [48]. In order to take advantage of a batched approach, two types of concurrency can be exploited on the GPU: 1) the parallelism of the standard operations, which is constrained by the data dependencies and the size of the local problem and 2) the intrinsic concurrency of the CUDA blocks. Very few references address the parallel implementation of batched implementations on GPUs. In this work, we have developed a CUDA Batched Cholesky Solver (hereafter, referred to `CuBCholS`). This new batched GPU implementation is especially appropriate to handle hundreds of small to moderate dense linear systems. We expect that this implementation and/or strategy can also be applied in analogous and related operations arising in hyperspectral image processing.

1) Parallel RX on the GPU Architecture: The various stages in the RX algorithm have been parallelized using different strategies. In the first stage, the correlation matrix is computed using the entire GPU architecture. In this case, the calculation of the correlation matrix is cast as a matrix–matrix product and is computed via function `gemm` of the CUBLAS library (available in the NVIDIA CUDA SDK [49]). For real hyperspectral images, the matrix–matrix product is usually large enough to be efficiently performed by the GPU, since the two matrices being multiplied are of dimension $(l \cdot s) \times b$.

For the second stage, we have developed specialized kernels for the Cholesky factorization, the computation of the linear system solver, and the inner product. In this case, the small size of the system drove us to implement our own algebraic routines instead of leveraging routines from existing general libraries. The Cholesky factorization of the matrix is carried out using only one CUDA block to avoid communication penalties, and the results are stored in the GPU memory. For the solution of the linear system, each pixel of the image is assigned to one CUDA block and each simple band of the image is mapped to one CUDA thread. That is, the threads of a block cooperate in the computation of a single independent linear system and all the blocks work independently in parallel, avoiding communication between different blocks. Furthermore, the inner product involving $\mathbf{z}$ is merged with the same function of the system solver.

2) Parallel LRX on the GPU Architecture: The previous description of LRX reveals that, for real hyperspectral scenes, this algorithm exhibits a high computational cost. Fig. 1 shows the workflow of our proposed GPU processing of the LRX algorithm including the routine `CuBCholS`.

In principle, the computation of the correlation matrices (Stage 1) can be regarded as a set of independent procedures or tasks assigned to a CUDA block, where each block computes the matrix associated to the pixel being processed. However, our implementation takes advantage of the recurrence relation between the correlation matrices associated to neighbor pixels to

[4]Available: http://www.nvidia.com/object/cuda_home_new.html.

[5]Available: https://developer.nvidia.com/cuBLAS.

[6]Available: http://www.culatools.com.

[7]Available: https://developer.nvidia.com/magma.

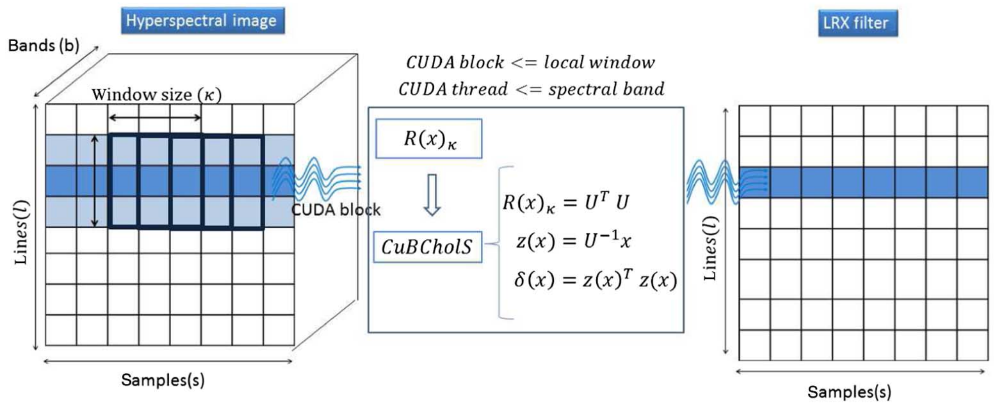

Fig. 1. Workflow of our proposed GPU processing of the LRX algorithm. (Left) representation of a hyperspectral image. (Center) stages for the computation of LRX algorithm and the use of CuBcholS. (Right) grayscale image resulting after LRX process.

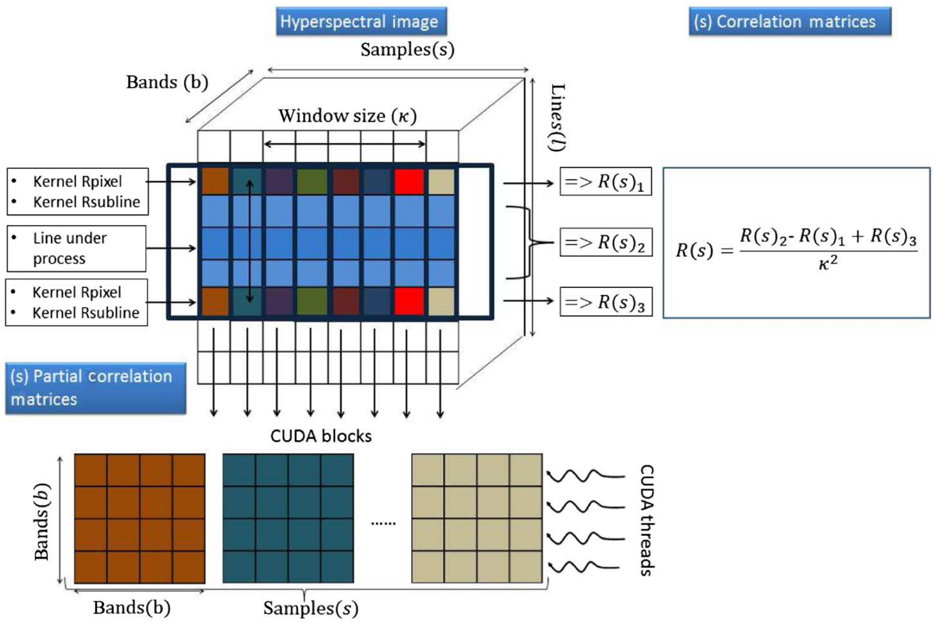

Fig. 2. Diagram of the recurrence correlation matrix implementation for LRX on GPU.

reduce the computational cost [15], [29]. As a result, it is important to note that each pixel y defines a correlation matrix as $R_{1\times 1}(y) = y \cdot y^T$, and the correlation matrix related to each $\kappa \times \kappa$ window centered at pixel x can be computed as $R_{\kappa\times\kappa}(x) = \sum_{y\in W} R_{1\times 1}(y)$, where W denotes the set of pixels within the window. Therefore, windows which share pixels also have in common partial computations of their correlation matrices. Thus, the computation of the correlation matrices $R_{\kappa\times\kappa}(x)$ on GPUs that avoids redundant operations is based on the following two kernels: 1) `Rpixel` computes the correlation matrices $R_{1\times 1}(y) = y_i y_i^T$ for all pixels into one line of the image, where each CUDA block computes one matrix $R_{1\times 1}(y)$, which is stored in the global memory of GPU and 2) `RSubline` evaluates the correlation matrices involved in the $1 \times \kappa$ windows centered at pixels of one line (sublines), so that each CUDA block computes the sum of matrices $R_{1\times\kappa}(x) = \sum_{y\in W} R_{1\times 1}(y)$. Both kernels use local registers to improve their performance; shared memory is not utilized; instead, after executing each kernel, the correlation matrices are stored in global memory. Bearing in mind the recurrence relation between the correlation matrices involved in two consecutive lines, described in [15], the matrices of a new line can be computed as a downdating/updating process from the matrices in the previous line. In particular, by subtracting the matrices $R_{\kappa\times\kappa}(x)$ of pixels from the line that disappears from the windows, and adding the matrices of pixels associated with the new line comprised by the new windows. Therefore, after the computation of the correlation matrices for the first line of the hyperspectral image, this stage on the GPU is expressed as an repetitive procedure where the matrices involved in the new/old sublines into/out the new windows are computed via kernels `Rpixel` and `RSubline`, and then they are added/subtracted to update the correlation matrices related to every new line and scaled by the number of elements into the window. Fig. 2 graphically illustrates this procedure. This implementation strategy prevents computational redundancies during the first stage of LRX, at the cost of intensive access to the GPU global memory.

To summarize the previous description, the main task of the initial stage is the computation of a set of correlation matrices for the first line of the hyperspectral image, followed by the correlation matrices associated with the second, third, fourth, etc., lines of the image using the aforementioned recurrence relation, which saves significant computational cost. As a result, in this scheme, Stage 1 is composed by s (the number of pixels in one line) tasks which compute in parallel the s correlation matrices related to the samples in a specific line.

The second stage of LRX is computed on the GPU, with our proposed Batched Cholesky Solver. In particular, in this case, s factorizations and linear systems have to be solved per image line, so we need to tackle hundreds of independent linear systems of small size. For this reason, we leverage a batched GPU kernel to solve these systems on the GPU. Our introspection is that this approach can solve several linear systems efficiently and is more suitable for an optimized implementation in terms of performance.

Our optimized GPU kernel CUDA Batched Cholesly Solver (`CuBCholS`) aggregates the solution of s Cholesky factorizations with their corresponding linear systems, and the inner product, which can be computed in parallel. According to this scheme, in our optimized implementation, the resources of a single CUDA block such as shared memory and thread registers are optimally exploited, and the set of blocks exploit the resources of the entire GPU platform processing in parallel the s pixels of an image line.

In summary, two levels of parallelism are exploited by `CuBCholS` in the GPU (for illustrative purposes, Algorithm 2 shows a pseudocode of this kernel):

a) Internally: Each factorization is decomposed into a set of tasks with dependencies among them. The pixel vector $\mathbf{x}$ and the intermediate vector $\mathbf{z}$ are stored in shared memory during the computation of the system, enabling fast data access. Stage 2 is computed with a fused loop that comprises the factorization, the system solution, and calculation of the norm of vector $\mathbf{z}$; and computed using a thread per linear system. Since the operations are performed in the same loop, the partial products are computed at the same time the elements of the intermediate vector z are obtained, take advantage of the memory cache, the shared memory, and data locality.

b) Externally: The computation can proceed independently and in parallel for each block in execution, where the number of blocks depends on the number of pixels in a line of the image. Each CUDA block reads from global memory its corresponding correlation matrix computed previously and the pixel vector of the image. These data are organized in memory so as to ensure a coalesced access memory by the threads of each block [49].

IV. Experimental Results

This section is devoted to describing the datasets and the computational platforms used in the performance evaluation of the strategies proposed in this paper. The section ends with an analysis of the obtained experimental results in terms of running time and energy consumption.

A. Hyperspectral Datasets

Two real hyperspectral datasets, collected by different instruments, have been used in our experiments in order to evaluate the computational performance (efficiency) of the proposed optimized algorithms in the task of detecting anomalies. These scenarios feature different characteristics, such as the image size, the spectral resolution, the spatial location of the anomalous pixels in the image, the hyperspectral instrument used for data collection, or the size of the anomalies to be detected.

TABLE I
Average Power Requirements (W) for the Execution of the RX/LRX Algorithms in Platforms Targeted in the Experiments

Platform	Idle (W)	1 core (W)	Max. cores (W)
Intel Xeon (8 cores)	**65**	**101**	**179**
Intel i7 (6 cores)	78	135	235
NVIDIA Tesla K20	94	–	220

HYDICE: The scene is extensively described and used in [13]. It is an image consisting of 64×64 pixels and 169 bands with 15 panels in the scene. The sizes (in meters) of the panels in the first, second, and third columns are 3×3, 2×2 and 1×1, respectively. This image was acquired with 210 spectral bands and a spectral coverage from 0.4 to 2.5 microns. Low signal/high noise bands: 1–3 and 202–210; as well as water vapor absorption bands: 101–112 and 137–153, were removed prior to the experiments so a total of 169 bands were finally used. The spatial resolution of the scene is 1.56 m (i.e., the last column of targets are subpixel in size) and the spectral resolution is 10 nm.

WTC: The second real hyperspectral dataset was collected by the AVIRIS instrument, flown by NASA's Jet Propulsion Laboratory over the World Trade Center (WTC) area in New York City on September 16, 2001. The size of the full scene is 614×512 pixels with 224 spectral bands, for a total size of about 140 MB (this is the standard size of the data chunks collected by the AVIRIS instrument before saving the data to disk in the onboard data collection). Extensive reference information, collected by U.S. Geological Survey (USGS), is available for the WTC scene.[8] A subset of this scene was selected for experiments, centered at the region of interest (hot spots zone at the WTC), consisting of 192×192 pixels and 224 spectral bands. This subset is referred as *WTC*160 in the experiments, while the whole image is denoted as *WTC*614.

B. Platforms and Methodology for Measurements

In this section, we describe the computational platforms, techniques, and metrics used in the performance analysis of our implementations in terms of runtime, power requirements, and energy consumption. The following experiments were performed on two multicore systems and one GPU-equipped platform. The first multicore server comprises two Intel Xeon E5504 (8 cores) at 2.0 GHz with 2×4 MB of L3 cache memory and 32 GB of RAM. The second multicore platform is composed by an Intel i7 4960 (6 cores) at 3.6 GHz with 16 GB of RAM. The i7 processor is considerably faster than the Xeon but requires more power (see Table I for details). Finally, the GPU is an NVIDIA Tesla K20 with 2,496 CUDA cores at 706 MHz and 5 GB of RAM; this GPU architecture supports the CUDA capability 3.5, and is connected to the platform with the Intel i7 processor described previously. These multicore processors and GPU considered represent the state of the art in HPC technology.

[8]Available: http://speclab.cr.usgs.gov/wtc.

The operating system was Linux Rocks 64 bits version; the compilers were `icc` 11 for the multicore servers and `nvcc` with the CUDA toolkit 5.5 for the GPU, using in each case the appropriate optimization flags at compilation time; the programming language was C combined with the linear algebra libraries Intel MKL 10.3.9 for the multicore servers, and CUBLAS version 2 for the GPU. Different device configurations were in order to optimize the use of the cache and shared memory for the GPU implementations, but the global performance achieved was basically the same.

Algorithm 2. Pseudocode of the `CuBCholS` kernel.

```
$CuBCholS <<< s, b >>>$ {one CUDA block per sample of a line, where $s = \text{samples}$ and $b = \text{bands}$}
**for** $k = 1$ **to** $b$ **do**
    $U = sqrt(diag(U))$ {sqrt of diagonal elements of correlation matrix}
    $z[k] = H_x[k]/diag(U)$ {solve the linear system using the corresponding pixel image as right hand side}
    $U[k, k+1:b] = U[k, k+1:b]/diag(U)$ {update the triangular matrix U. Paralellized by CUDA threads}
    $H[k+1:b] -= U[k, k+1:b]^T \cdot z[k]$ {update the right hand sides of the equation. Paralellized by CUDA threads}
    **for** $j = k+1$ **to** $b$ **do** {paralellized by CUDA threads}
        **for** $i = k+1$ **to** $j$ **do**
            $U[i, j] -= U[k, i] \cdot U[k, j]$
        **end for**
    **end for**
**end for**
```

For the energy consumption measurements, it was necessary to obtain reliable data of the average power consumption of each computing platform during the execution of our algorithms. Our aim was to obtain the real power dissipation of the entire platform considered when our algorithms are processed. This parameter represents the near maximum power for a thermally significant period due to processing. We used a WattsUp wattmeter directly connected to the power distribution unit (PDU), and the library `pmlib` [50] to automatically collect sampling measures of the power consumption without perturbing the algorithm executions [51], [52].

At this point, it is necessary to highlight that the power requirements can oscillate during the execution time. In order to obtain the real power dissipation of the platform during the execution of our algorithms, we first performed a set of experiments which consisted of repeating the different stages of the algorithms during 5 min and collecting power samples for the last 4 min. The first minute (warm-up time) was discarded to allow the architecture to achieve a close-to-stable power dissipation. These experiments indicate that, after the warm-up time, the power requirements reach a plateau and the value of this constant is the same for all stages and datasets. This is mainly due to the fact that, during the execution of a stage, a significant part of the available resources are used. This fact was verified by the results obtained from several profiling and performance software analysis tools. (Specifically, Vtune[9] and the "top" Linux command for the multicore and Nsight[10] for the GPU version.) Consequently, we will assume that the average value of the power consumptions measured in these experiments is an accurate estimation of the power consumption during the execution of RX or LRX in each platform.

TABLE II
RUNTIME (S) AND ENERGY CONSUMPTION (J) FOR THREE HYPERSPECTRAL IMAGES, PROCESSING THE SEQUENTIAL, AND PARALLEL VERSION OF RX ALGORITHM

Dataset	Platform	Time (s)		Energy (J)	
		1 core	Max. cores	1 core	Max. cores
HYDICE (64×64×169)	Xeon (8)	0.08	0.05	8.08	8.95
	i7 (6)	**0.03**	**0.02**	**4.05**	**4.70**
	Tesla k20		0.02		4.40
WTC160 (160×170×224)	Xeon (8)	0.62	0.16	62.62	28.64
	i7 (6)	**0.19**	**0.08**	**25.65**	**18.80**
	Tesla k20		0.13		28.60
WTC614 (614×512×224)	Xeon (8)	7.08	1.33	715.08	238.07
	i7 (6)	**2.05**	**0.69**	**276.75**	**162.15**
	Tesla k20		1.31		288.20

The average values of the power consumption recorded during these preliminary experiments are provided in Table I. For the multicore processors, we evaluated three different states of the hardware: "idle," when the platform is not computing; "1 core" when the program is executed using a single core; and "Max. cores," using the maximum number of available cores. For the GPU architecture, the options we selected were idle or "Max. cores." Using the average value of the power consumption for each platform given in Table I, the energy consumption of an algorithm is calculated as the product of runtime and the average power.

The results tabulated in Table I indicate that, in terms of power consumption, the Intel Xeon has less requirements than the Intel i7 platform in all cases: idle state, use of a single core, and use of all cores. Notice that Intel Xeon with 8 cores consumes less power than Intel i7 with 6 cores. These differences are in part due to the clock rate of each processor, but are also related with the architecture organization. Moreover, the power required by the GPU is similar to that of the Intel i7 platform using 6 cores.

C. Performance of RX and LRX on Multicore Processors and GPUs

The performance of the RX and LRX algorithms was evaluated using the three described platforms, and the three hyperspectral datasets. Tables II and III collect the experimental results obtained for the performance evaluation of the RX and LRX algorithms, respectively. Table II shows the values of the runtime and energy consumption for the RX algorithm executed using a single core (column "1 core") and the maximum number of cores available in each platform (column "Max. cores"). Similarly, Table III shows the runtime and the energy consumption for the

[9]Available: http://software.intel.com/en-us/intel-vtune-amplifier-xe.

[10]Available: http://www.nvidia.com/object/nsight.html.

TABLE III
RUNTIME (S) AND ENERGY CONSUMPTION (KJ) FOR THREE HYPERSPECTRAL IMAGES, PROCESSING THE SEQUENTIAL, AND PARALLEL VERSION OF LRX ALGORITHM

Dataset	Platform	Time (s)						Energy (kJ)					
		1 core			Max. cores			1 core			Max. cores		
		S1	S2	Total	S1	S2	Total	S1	S2	Total	S1	S2	Total
HYDICE (64×64×169)	Xeon (8)	16.82	1.73	18.56	1.78	0.92	2.71	1.69	0.17	1.87	0.32	0.16	0.48
	i7 (6)	**6.64**	**0.65**	**7.30**	**0.91**	**0.47**	**1.39**	**0.89**	**0.08**	**0.98**	**0.21**	**0.11**	**0.32**
	Tesla k20				1.23	0.72	1.95				0.27	0.16	0.43
					SP_{GPU} =1.4x(Xeon), 0.7x(i7)								
WTC160 (160×170×224)	Xeon (8)	212.82	22.50	235.35	29.57	**4.32**	33.92	21.49	2.27	23.77	5.29	0.77	6.07
	i7 (6)	**102.37**	**7.55**	**109.95**	16.65	6.27	22.94	**13.82**	**1.02**	**14.84**	3.91	1.47	5.39
	Tesla k20				**13.37**	8.52	**21.93**				**2.94**	1.87	**4.82**
					SP_{GPU} =1.5x(Xeon), 1x(i7)								
WTC614 (614×512×224)	Xeon (8)	2402.96	253.70	2656.96	320.39	**44.61**	363.05	242.69	25.62	268.25	57.35	**8.89**	64.98
	i7 (6)	**1345.19**	**87.47**	**1433.10**	199.51	72.89	272.54	**181.60**	**11.81**	**193.46**	46.88	17.13	64.04
	Tesla k20				**159.43**	90.32	**249.98**				**35.07**	19.87	**54.99**
					SP_{GPU} =1.4x(Xeon), 1.1x(i7)								

Also, the speedup for the GPU version (SP_{GPU}) as compared to each multicore version is included.

LRX algorithm, using a window size of $23 \times 23 (\kappa = 23)$ which was demonstrated to yield reasonable anomaly detection results compared with RX [15]. To analyze the behavior of LRX, the values of the runtime and energy consumption for Stages 1 and 2 (columns S1 and S2) are also provided in this table. Notice that these stages comprise most of the computational cost of LRX [more than 99% of the total runtimes (column Total)]. For the GPU version, the values in column Total also include the time of copying the image dataset from the CPU memory to the device memory.

Table II clearly shows that the multicore servers outperform the GPU in terms of runtime and energy consumption for the three image datasets, and the Intel i7 processor is superior than the Intel Xeon in all cases, both when using 1 core and the maximum number of available cores.

In terms of the total runtime, according to the results in Tables II and III, when parallelism is leveraged in the multicore servers, performance can be competitive with that of the GPU [36]. The best platform for the RX algorithm is the Intel i7 multicore processor since it delivers the best results in both metrics for all the experiments. Although the Intel i7 has only 6 cores, its power dissipation rate is higher than that of the Intel Xeon. The vast computational power of the GPU cannot be exploited because this algorithm does not present enough computational load due to the reduced size of the image datasets.

Table III reports experimental results for the LRX algorithm, including data for the most computationally expensive stages (S1 and S2). The cost of the initialization stage is included in the total execution time but are negligible compared with the runtime of Stages 1 and 2. In fact, Stages 1 and 2 exhibit different performances for the multicore processors and GPUs. Notice that the peak performance of the GPU cannot be achieved since the computational load associated to processing the entire image is not high enough. Since the process is computed line-by-line, there are several CUDA kernel calls in the processing of the image. In this case the main bottleneck is the memory latency, especially for Stage 2, since 1) there are several accesses to global memory; 2) for each pixel several matrices need to be stored; 3) there is little reuse of the data; 4) the matrices change at each step of the Cholesky factorization; and 5) irregular accesses to the triangular matrices occur. However, it is remarkable that the occupancy achieved is around 90% for the kernels of Stage 1 and 85% for those in Stage 2.

Focusing now on the multicore servers, using a single core, the Intel i7 implementation outperforms the Intel Xeon version for all the image datasets, stages and globally (Total). For the smallest image dataset, the Intel i7 is always superior to the Intel Xeon and also better than the Tesla K20. For the medium and large size image datasets, the GPU solution outperforms the multicore implementations for Stage 1 and globally (Total), but Intel Xeon is faster for Stage 2. Analogous comments apply to energy consumption.

From the experimental results reported in Tables II and III, it is clear that RX is much faster and consumes less energy than LRX. However, the LRX algorithm has demonstrated to provide better anomaly detection results compared with RX [15]. Also, in Table III, we have included the results of the speed-up of GPU versus each multicore CPU using the maximum number of cores. In order to simplify the comparison of the different architectures (multicore processors and GPUs), and the different metrics (runtime, speed-up, and energy consumption), Fig. 3 displays the acceleration factor for each dataset. Both performance metrics, runtime (leftmost part of the graphics) and energy consumption (rightmost part of the graphics) have been normalized to 1, taking as reference the best result in each case. Also, for each platform, the values are decomposed into the different stages of the algorithm.

This analysis reveals that if a multicore processor is used for sequential processing (1 core), the results in terms of runtime and energy consumption are poor compared to the GPU. On the other hand, according to both metrics the GPU is the best platform for LRX when the image is large enough (i.e., when the computational workload is sufficiently high). When the image is composed by a large number of samples per line, the GPU can deliver high performance because, in this implementation, the GPU accelerates the computation of each line of the image with the batched implementation. Moreover, these results show the advantage of the `CuBCholS` routine in terms of performance when images of large dimension are considered, since in the GPU version all the pixels in a row are executed concurrently. In this context, when large image datasets are processed, the GPU approach outperforms the multicore implementations. It is also worth pointing out that the platforms with higher power requirements (Intel i7 and NVIDIA K20; see Table I) obtain the best result in terms of both runtime and energy consumption.

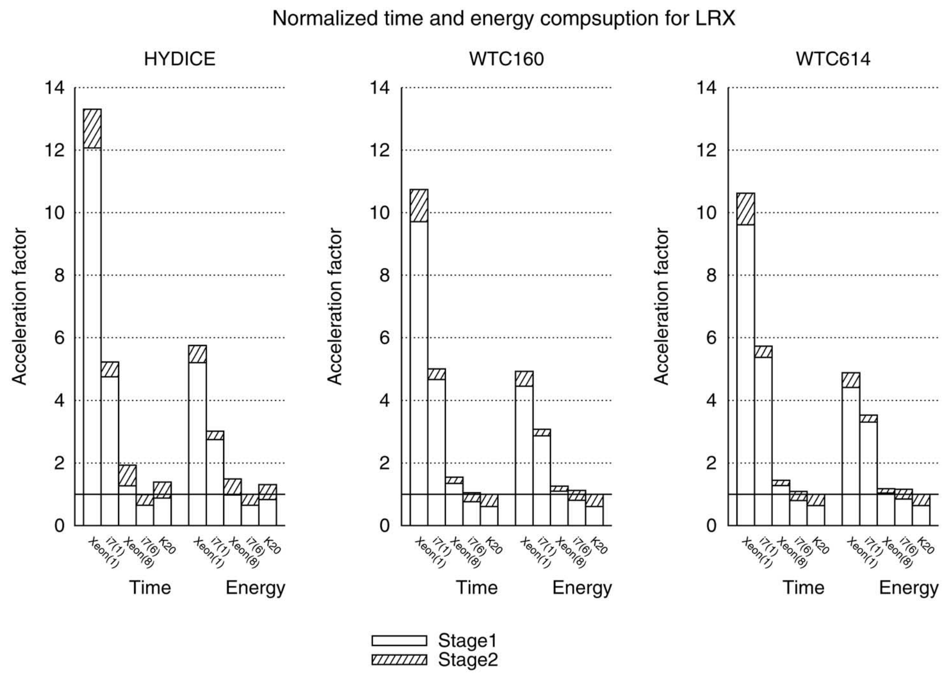

Fig. 3. Comparison of the time and energy normalized for the LRX algorithm. The number of cores used in each case is reported in brackets.

One final observation related to energy consumption is that exploiting all the method's concurrency and the resources of the architectures is crucial, since the impact is critical to reduce energy consumption.

V. Concluding Remarks

In this paper, we have developed several efficient implementations of anomaly detectors for hyperspectral imagery on multicore processors and commodity GPUs. Specifically, we have focused on the RX detector and its local variant LRX, extensively analyzing the computational performance and the energy consumption of the newly developed implementations, which exploit both sequential and parallel, as well as software and hardware, optimizations. An innovative contribution of this work is the use of Cholesky factorization for the solution of thousands of dense linear systems of small dimension, which is particularly important for the efficient implementation of the LRX algorithm. We have also exploited the GPU resources using batched implementations. An additional important contribution of this work is the detailed analysis of the runtime versus the energy consumption of the different solutions discussed, which is quite relevant for practical purposes in real Earth observation missions. Our experimental results, conducted using a variety of hyperspectral scenes and multicore/GPU platforms, indicate that the target architectures can provide high performance with moderate energy consumption, thus opening important future perspectives for the practical exploitation of anomaly detection algorithms in real missions. For instance, it has been observed experimentally that our parallel implementations can execution the RX algorithm in parallel (i.e., simultaneously) with the collection of the data, while the LRX performance is also significantly reduced by the exploitation of the proposed optimizations. Although in this work the RX and the LRX were separately exploited, we plan to develop a hybrid implementation using both algorithms adaptively as part of future work. Another planned future direction consists in developing alternative local variants of the RX algorithm in order to improve detection accuracy. An third topic deserving future research is a more in-depth investigation of strategies to further reduce the runtime, in particular, for the LRX algorithm. For this purpose, we plan to take advantage of multi-GPU implementations. Since each line can be processed independently, the main idea here is to divide the image, and process each subimage independently on a different GPU without significant communications, which generally impose a major bottleneck for multi-GPU implementations.

References

[1] A. Goetz, G. Vane, J. Solomon, and B. Rock, "Imaging spectrometry for earth remote sensing," *Science*, vol. 228, pp. 1147–1153, 1985.
[2] R. Green *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sens. Environ.*, vol. 65, no. 3, pp. 227–248, 1998.
[3] C.-I. Chang, *Hyperspectral Data Processing: Algorithm Design and Analysis*. Hoboken, NJ, USA: Wiley, 2013.
[4] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis: An Introduction*. New York, NY, USA: Springer, 2006.
[5] C.-I. Chang, *Hyperspectral Data Exploitation: Theory and Applications*. Hoboken, NJ, USA: Wiley, 2007.
[6] G. Shaw and D. Manolakis, "Signal processing for hyperspectral image exploitation," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 12–16, Jan. 2002.
[7] C.-I. Chang and S.-S. Chiang, "Anomaly detection and classification for hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 6, pp. 1314–1325, Jun. 2002.
[8] D. W. J. Stein *et al.*, "Anomaly detection from hyperspectral imagery," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 58–69, Jan. 2002.
[9] S. Matteoli, M. Diani, and G. Corsini, "A tutorial overview of anomaly detection in hyperspectral images," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 25, no. 7, pp. 5–28, Jul. 2010.

[10] B. Du and L. Zhang, "Random selection based anomaly detector for hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 5, pp. 1578–1589, May 2011.

[11] D. Borghys, I. Kasen, V. Achard, and C. Perneel, "Comparative evaluation of hyperspectral anomaly detectors in different types of background," in *Proc. SPIE*, 2012, pp. 83 902J–83 902J-12.

[12] I. S. Reed and X. Yu, "Adaptive multiple-band CFAR detection of an optical pattern with unknown spectral distribution," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 38, no. 10, pp. 1760–1770, Oct. 1990.

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

[14] J. M. Molero, E. M. Garzón, I. García, and A. Plaza, "Anomaly detection based on a parallel kernel RX algorithm for multicore platforms," *J. Appl. Remote Sens.*, vol. 6, p. 12, 2012.

[15] J. M. Molero, E. M. Garzón, I. García, and A. Plaza, "Analysis and optimizations of global and local versions of the RX algorithm for anomaly detection in hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 2, pp. 801–814, Apr. 2013.

[16] Y. P. Taitano, B. A. Geier, and K. W. Bauer, "A locally adaptable iterative RX detector," *EURASIP J. Adv. Signal Process.*, vol. 2010, p. 10, 2010, doi: 10.1155/2010/341908

[17] L. Ma, M. M. Crawford, and J. Tian, "Anomaly detection for hyperspectral images based on robust locally linear embedding," *J. Infrared Millimeter Terahertz Waves*, vol. 31, pp. 753–762, 2010.

[18] N. Acito, M. Diani, and G. Corsini, "Computational load reduction for anomaly detection in hyperspectral images: An experimental comparative analysis," in *Proc. IEEE Geosci. Remote Sens. Symp.*, 2009, vol. 1, pp. 3206–3209.

[19] A. Plaza and C.-I. Chang, "Special issue on high performance computing for hyperspectral imaging," *Int. J. High Perform. Comput. Appl.*, vol. 22, no. 4, pp. 363–365, 2008.

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

[21] A. Rossi, N. Acito, M. Diani, and G. Corsini, "RX architectures for real-time anomaly detection in hyperspectral images," *J. Real-Time Image Process.*, pp. 1–15, 2012.

[22] C.-I. Chang, H. Ren, and S.-S. Chiang, "Real-time processing algorithms for target detection and classification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 4, pp. 760–768, Apr. 2001.

[23] C. Lee, S. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.

[24] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*. Boca Raton, FL, USA: CRC Press, 2007.

[25] 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.

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

[27] A. Paz and A. Plaza, "Clusters versus GPUs for parallel target and anomaly detection in hyperspectral images," *EURASIP J. Adv. Signal Process.*, vol. 2010, pp. 35:1–35:18, Feb. 2010.

[28] E. Christophe, J. Michel, and J. Inglada, "Remote sensing processing: From multicore to GPU," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 643–652, Sep. 2011.

[29] J. M. Molero, E. M. Garzón, I. García, E. S. Quintana-Ortí, and A. Plaza, "Accelerating the KRX algorithm for anomaly detection in hyperspectral data on GPUs," in *Proc. 12th Int. Conf. Comput. Math. Methods Sci. Eng. (CMMSE)*, 2012, pp. 860–863.

[30] J. M. Molero *et al.*, "Fast anomaly detection in hyperspectral images with RX method on heterogeneous clusters," *J. Supercomput.*, vol. 58, no. 3, pp. 411–419, 2011.

[31] A. Paz, A. Plaza, and S. Blazquez, "Parallel implementation of target and anomaly detection algorithms for hyperspectral imagery," in *Proc. IEEE Geosci. Remote Sens. Symp.*, 2008, vol. 2, pp. 589–592.

[32] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. San Mateo, CA, USA: Morgan Kaufmann, 2012.

[33] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge, U.K.: Cambridge Univ. Press, 1992.

[34] A. Quarteroni, R. Sacco, and F. Saleri, "Numerical Mathematics," in *Texts in Applied Mathematics*. New York, NY, USA: Springer, 2000.

[35] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: Johns Hopkins Univ. Press, 1996.

[36] V. W. Lee *et al.*, "Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, 2010.

[37] S. Bernabé *et al.*, "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.

[38] A. Paz, A. Plaza, and J. Plaza, "Comparative analysis of different implementations of a parallel algorithm for automatic target detection and classification of hyperspectral images," in *Proc. SPIE*, 2009, vol. 7455, pp. 74 550X–74 550X-11.

[39] D. B. Kirk and W. M. W. Hwu, *Programming Massively Parallel Processors*. San Mateo, CA, USA: Morgan Kaufmann, 2013.

[40] M. Marqués, G. Quintana-Ortí, E. S. Quintana-Ortí, and R. van de Geijn, "Using desktop computers to solve large-scale dense linear algebra problems," *J. Supercomput.*, vol. 58, pp. 145–150, 2011.

[41] Intel. (2013). *Math kernel Library (MKL) Documentation* [Online]. Available: http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/

[42] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. V. D. Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (SIAM), 1990.

[43] R. Farber, *CUDA: Application Design and Development*. San Mateo, CA, USA: Morgan Kaufmann, 2010.

[44] Nvidia. (2013). *Kepler GK110 Architecture Whitepaper* [Online]. Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[45] H. Ortega-Arranz, Y. Torres, D. Llanos, and A. Gonzalez-Escribano, "A tuned concurrent-kernel approach to speed up the APSP problem," in *Proc. 13th Int. Conf. Comput. Math. Methods Sci. Eng. (CMMSE)*, 2013, pp. 1114–1125.

[46] minus 4 Nvidia. (2013). *CUDA C Programming Guide* [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[47] H. Ortega-Arranz, Y. Torres, D. R. Llanos, and A. Gonzalez-Escribano, "A new GPU-based approach to the shortest path problem," in *Proc. Int. Conf. High Perform. Comput. Simul. (HPCS)*, 2013, pp. 505–511.

[48] M. Anderson, D. Sheffield, and K. Keutzer, "A predictive model for solving small linear algebra problems in GPU registers," in *IEEE 26th Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2012, pp. 2–13.

[49] minus 4 Nvidia. (2013). *CUBLAS Manual* [Online]. Available: http://docs.nvidia.com/cuda/cublas/index.html

[50] P. Alonso *et al.*, "Tools for power-energy modelling and analysis of parallel scientific applications," in *Proc. 2012 41st Int. Conf. Parallel Process.*, 2012, pp. 420–429.

[51] M. Castillo *et al.*, "Hyperspectral unmixing on multicore DSPs: Trading off performance for energy," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, pp. 1–8, 2013, to be published.

[52] A. Remón, S. Sanchez, S. Bernabé, E. Quintana-Ortí, and A. Plaza, "Performance versus energy consumption of hyperspectral unmixing algorithms on multi-core platforms," *EURASIP J. Adv. Signal Process.*, vol. 2013, p. 68, 2013.

José M. Molero was born in Almería, Spain, in 1983. He received the M.Sc. degree in computer science from the University of Almería, Almería, Spain, in 2009. He is currently pursuing the Ph.D. degree at the Supercomputing-Algorithms Research Group, University of Almería.

His research interests include high-performance computing and remote sensing, especially in hyperspectral image processing.

Ester M. Garzón received the B.Sc. degree in physics from the University of Granada, Granada, Spain, in 1985, and the Ph.D. degree in computer science from the University of Almería, Almería, Spain, in 2000.

Currently, she is an Associate Professor with the Department of Informatics, University of Almería, and Head of the Research Group Supercomputing Algorithms. Her research interests include high-performance computing, matrix computation, and imaging processing.

Inmaculada García received the B.Sc. degree in physics from the Complutense University of Madrid, Madrid, Spain, in 1977, and the Ph.D. degree in physics from the University of Santiago de Compostela, A Coruña, Spain, in 1986.

From 1977 to 1987, she was an Assistant Professor, Associate Professor during 1987–1997, between 1997 and 2010, a Full Professor with the University of Almería, Almería, Spain and, since 2010, a Full Professor with the University of Málaga, Málaga, Spain. She was a Head of the Department of Computer Architecture and Electronics, University of Almería for more than 12 years. During 1994–1995, she was a Visiting Researcher with the University of Pennsylvania, PA, USA. Her research interests include parallel algorithms for irregular problems related to image processing, global optimization, and matrix computation.

Enrique S. Quintana-Ortí received the Bachelor 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 with the Universidad Jaume I, Castellón, 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.

Antonio Plaza (M'05–SM'07) received the Bachelor and Ph.D. degrees in computer engineering from the University of Extremadura, Spain, in 1999 and 2002, respectively.

He 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).

Prof. Plaza has been a Chair for the IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (2011). He 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 (2011–2012) and the IEEE Geoscience and Remote Sensing Magazine (2013). He was also 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) during 2011–2012, and is currently serving as the 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).