DOI : 10.1109/JSTARS.2015.2394778

Wavelet-Based Classification of Hyperspectral Images Using Extended Morphological Profiles on Graphics Processing Units

Pablo Quesada-Barriuso, Francisco Argüello, Dora B. Heras, and Jón Atli Benediktsson

***Abstract*—The availability of graphics processing units (GPUs) provides a low-cost solution to real-time processing, which may benefit many remote sensing applications. In this paper, a spectral–spatial classification scheme for hyperspectral images is specifically adapted for computing on GPUs. It is based on wavelets, extended morphological profiles (EMPs), and support vector machine (SVM). Additionally, a preprocessing stage is used to remove noise in the original hyperspectral image. The local computation of the techniques used in the proposed scheme makes them particularly suitable for parallel processing by blocks of threads in the GPU. Moreover, a block-asynchronous updating process is applied to the EMP to speedup the morphological reconstruction. The results over different hyperspectral images show that the execution can be speeded up to $8.2\times$ compared to an efficient OpenMP parallel implementation, achieving real-time hyperspectral image classification while maintaining the high classification accuracy values of the original classification scheme.**

***Index Terms*—Feature extraction, graphics processing unit (GPU), image classification, morphological operations, parallel processing, remote sensing, wavelet transforms.**

I. Introduction

In REMOTE sensing (RS), the amount of data stored in the hyperspectral images has increased the computational cost of the techniques used, and high performance computing (HPC) has become vital today [1], [2], especially for time critical applications such as on-board target detection for maritime rescue [3]. For example, in the field of spectral unmixing, specific algorithms as pixel purity index [4] and spectral unmixing chains involving endmembers extraction [5] were successfully parallelized in multicore processors, graphics processing units (GPUs), and field programmable gate arrays (FPGAs). In the applications of target detection, GPU real-time processing has been achieved through artificial neural networks [3] and orthogonal subspace projections [6].

Manuscript received September 15, 2014; revised December 22, 2014; accepted January 14, 2015. Date of publication February 10, 2015; date of current version July 30, 2015. This work was supported in part by the Ministry of Science and Innovation, Government of Spain, TIN2013-41129-P. The work of P. Quesada-Barriuso was supported by the Ministry of Science and Innovation, Government of Spain, under a MICINN-FPI Grant.

P. Quesada-Barriuso, F. Argüello, and D. B. Heras are with the Centro de Investigación en Tecnoloxías da Información, University of Santiago de Compostela, 15842 Santiago de Compostela, Spain (e-mail: pablo.quesada@usc.es; francisco.arguello@usc.es; dora.blanco@usc.es).

J. A. Benediktsson is with the Faculty of Electrical and Computer Engineering, University of Iceland, 107 Reykjavik, Iceland (e-mail: benedikt@hi.is).

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.2015.2394778

Regarding classification of hyperspectral images, the support vector machine (SVM) algorithm is generally recognized as the classifier that offers the best results in classification accuracy, even when the number of training samples is small [7]. This is why a variety of implementations for GPU processing can be found in the literature [8]–[10]. For example, the GPUSVM library [8] was successfully used in the GPU spectral–spatial classification scheme presented in [11].

The research efforts in these areas have led to more complete remote sensing applications, such as parallel open source libraries for remote sensing processing [12] and specific parallel spectral–spatial classification schemes [11], [13].

The classification accuracy can be improved if spatial information is incorporated [2]. Among the different spatial techniques, the extended morphological profile (EMP) [14] based on mathematical morphology has been widely investigated. The EMP is commonly created from the principal components [15], but other techniques as the wavelet transform can be used to reduce the dimensionality of the image. The discrete wavelet transform (DWT), which is usually based on two filters, is also applied in the spatial domain to filter the noise introduced in the acquisition of the image. The spectral and spatial information can be combined before the classification, i.e., by a stacked vector, or after the classification by a majority vote (MV). In [16], a scheme named WT-EMP combines a feature reduction in the spectral domain by wavelets to create an EMP, and a preprocessing to remove noise in the original hyperspectral image, before the classification by an SVM.

In this paper, the WT-EMP scheme proposed in [16] is mapped to the GPU to achieve real-time classification, using the compute unified device architecture (CUDA) developed by NVIDIA. An example of the DWT is shipped in the CUDA software development kit to compute very large one-dimensional (1-D) signals, but only supports signals with a length power of two. In this work, the proposed 1-D-DWT implementation is adapted to compute thousands of transformations (pixel vectors in the spectral domain) in parallel. In the case of the two-dimensional (2-D)-DWT, the GPU implementations [17]–[19] cannot manage more than two filters. Therefore, a new implementation is required to manage the three filters that are used in the denoised step of [16]. An asynchronous reconstruction algorithm, based on the hybrid iterative updating process originally proposed in [20], can speedup the computation of the EMP used in the scheme. This proposal introduces a novelty respect to the GPU morphological reconstruction found in the literature [21], by scanning the image

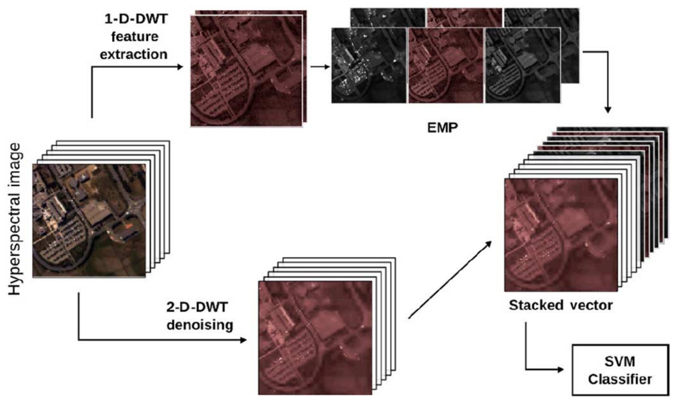


Fig. 1. Spectral–spatial classification scheme (WT-EMP) based on wavelets and EMPs.

in all directions at the same time. Despite the different SVM solutions for GPU found in the literature, the classification is performed by a new implementation adapted for multi-class problems.

The kernels have specially been adapted to optimize the use of the GPU resources, as well as the memory hierarchy. Using techniques that match the computing model of the GPU architecture, such as the local computation of the DWT and the morphological operations, the scheme can efficiently be implemented in the GPU for real-time classification.

This paper is organized as follows. The spectral–spatial classification scheme is described in Secttion II and the GPU processing in Section III. The results are presented in Section IV and finally, Section V presents the conclusion.

II. WT–EMP Spectral–Spatial Scheme

This section briefly describes the WT-EMP spectral–spatial classification scheme proposed in [16], which is summarized in Fig. 1. The spectral part of the scheme comprises the 1-D wavelet feature reduction, and the spatial part the EMP and a 2-D wavelet denoising of the original hyperspectral image. Finally, an SVM performs the classification of the resulting data.

Wavelets are mathematical tools for signal processing analysis at different scales [22]. The DWT of a signal $x[n]$ can be computed as the convolution of the signal with two filters [23]. A low-pass filter generates the approximation coefficients $a[n]$, whereas a high-pass one produces the detail coefficients $d[n]$. This process is recursively applied to the approximation sequence. In the WT-EMP scheme, a wavelet decomposition is applied on the spectral dimension of the hyperspectral image to reduce the number of bands. From the remaining m approximation coefficient-bands resulting from the 1-D-DWT feature reduction, the spatial features are created using an EMP [14].

The EMP captures spatial structures at different scales if increasing sizes of a structuring element (SE) used in the morphological operations are applied. First, for each coefficient-band W_i $(i = 1, \ldots, m)$, a morphological profile (MP) of n openings (γ_r) and n closings (ϕ_r) by reconstruction using an SE of growing size is created as follows:

$$\mathrm{MP}^{(n)}(W_i) = \{\gamma_r^{(n)}(W_i), \ldots, \gamma_r^{(1)}(W_i), W_i, \phi_r^{(1)}(W_i), \ldots, \phi_r^{(n)}(W_i)\}.$$

Second, the EMP is created from the $\mathrm{MP}^{(n)}$ of each coefficient-band W_i as follows:

$$\mathrm{EMP}_m^{(n)}(W) = \{\mathrm{MP}^{(n)}(W_1), \mathrm{MP}^{(n)}(W_2), \ldots, \mathrm{MP}^{(n)}(W_m)\}.$$

In addition to the construction of the EMP, preprocessing is applied to the original hyperspectral image to reduce noise (see bottom of Fig. 1). This preprocessing step is performed by soft-thresholding using a separable 2-D-DWT with the set of filters for perfect reconstruction presented in [24]. Finally, the classification is carried out by an SVM classifier. The input to the classifier is the stacked vector built from the denoised image and the EMP.

III. WT–EMP GPU Processing

This section first describes the GPU architecture and then highlights the challenges of GPU programming in Secttion III-A. The GPU implementation details are described in Section III-B–III-E.

A. GPU Architecture Overview

The recent GPUs provide massively parallel processing capabilities based on a single instruction, multiple data (SIMD) programming model. In particular, the CUDA developed by NVIDIA is organized in a set of streaming multiprocessors (SMXs) with many cores inside. The GPU can manage thousands of threads concurrently and execute the same instruction on different data by grouping 32 threads (*warps*) as the minimum collaborative size. The threads are arranged in 1-D, 2-D, or 3-D blocks that are scheduled to any of the available SMXs.

The memory hierarchy plays a key role in performance. The GPU has a global memory, a texture memory and a constant memory that are available for all the threads at any SMX. There is also an on-chip memory, known as shared memory, that enables extremely rapid read/write access to the data. This shared memory is only available for the threads of the same block.

In the Kepler architecture used in the experiments of this paper, there is also a L1 and L2 cache hierarchy. The caches are managed by the GPU but the programmer can take advantage of this cache hierarchy by exploiting the memory access pattern when reading data from the global memory (the same when writing data to the global memory).

The threads within the same block can be synchronized, e.g., to communicate intermediate results in the shared memory as part of a parallel computation. However, it is not possible to synchronize threads among different blocks. Owing to this restriction, the communication among all the threads must be through the global memory.

Different aspects must be taken in consideration to efficiently exploit the computational capabilities of the GPU [25]. For example: 1) Data transfer between the CPU and the GPU should be minimized. 2) The memory access pattern must be coalesced to consecutive memory locations. 3) Data loaded in shared memory should be reused to reduce read/write accesses to the global memory. 4) The number of threads per block must be optimized to run the maximum concurrent blocks allowed in each SMX. 5) Minimizing the global synchronization among blocks leads to a reduction in the execution time of the program. Paying attention to these aspects is vital for GPU programming.

B. 1-D-DWT on GPU

The 1-D-DWT is applied to each pixel vector in the spectral domain using the Cohen-Daubechies-Feauveau 9/7 wavelet (CDF97) with nine coefficients for the low-pass filter. The number of decompositions applied to each pixel vector is adjusted to produce at the end $m = 4$ coefficients. Only the approximation coefficients are used at the next level of decomposition, whereas the detail coefficients are discarded. Note that in this case, it is not necessary to compute the inverse 1-D-DWT.

The 1-D-DWT proposed in this work is adapted to compute thousands of transformations of the pixel vectors in parallel. The hyperspectral image is copied to the global memory of the GPU as a matrix with one pixel vector per row. Threads are configured in 1-D blocks of size the multiple of 32 nearest to the number of hyperspectral bands.

In the kernel, threads within the block load data from the global to the shared memory, where the convolution for the 1-D-DWT takes place. The kernel performs all the spectral decompositions in shared memory. Extra data are loaded with a circular padding in shared memory for the convolution of the last elements. Finally, the result is copied back to global memory. Data are rearranged as a hyperspectral cube before the next step.

C. EMP on GPU

The EMP is the set of MPs created through opening and closing by reconstruction for each coefficient-band resulting from the previous step. The opening by reconstruction $\gamma_r^{(n)}(I)$ of an image I is defined as the reconstruction by dilation of I from the erosion with an SE of size n of I (in the following, J will denote the eroded image). The reconstruction is an iterative process defined as [26]

$$\delta_I^{(n)}(J) = \underbrace{\delta_I^{(1)} \circ \delta_I^{(1)} \circ \cdots \circ \delta_I^{(1)}}_{\text{until stability}}(J)$$

where $\delta_I^{(1)}(J)$ is the geodesic dilation of the marker J with respect to a mask I defined as

$$\delta_I^{(1)}(J) = \min((J \oplus B), I)$$

where $J \oplus B$ is the dilation of J by an SE B of size 1. Similarly, the closing by reconstruction $\phi_r^{(n)}(I)$ is defined as the reconstruction by erosion of I from the dilation of the image. Although, this approach can easily be implemented, even in parallel for multicore architectures [26], it requires

Algorithm 1. Sequential Reconstruction Algorithm [26]

```
Require: $J$ the marker image and $I$ the mask image.
 1: procedure SR($J, I$)
 2:   repeat
 3:     for each pixel $p$ in forward scanning do ▷ forward scan
 4:       $J(p) = \min(\max\{J(q), q \in N_G^+(p) \cup \{p\}\}, I(p))$
 5:     end for
 6:     for each pixel $p$ in backward scanning do ▷ backward scan
 7:       $J(p) = \min(\max\{J(q), q \in N_G^-(p) \cup \{p\}\}, I(p))$
 8:     end for
 9:   until stability
10: end procedure
```

Algorithm 2. GPU Block-Asynchronous Reconstruction (BAR)

```
Require: $J$ the marker image and $I$ the mask image.
 1: procedure MMRECONSTRUCTION($J, I$)
 2:     repeat                          ▷ inter-block updating
 3:       <BAR>($J, I$)
 4:       Global synchronization among blocks
 5:     until stability
 6: end procedure
 7: kernel <BAR>($J, I$)                ▷ intra-block updating
 8: sharedmem[BLOCK] = $J$[BLOCK] from global memory
 9: repeat
10:     $\text{sharedmem}(p) = \min(\max\{\text{sharedmem}(q), q \in N_G(p) \cup \{p\}\}, I(p))$
11:     Local synchronization among threads ▷ Asynchronous among blocks
12: until stability
13: $J$[BLOCK] = sharedmem[BLOCK] to global memory
14: end kernel
```

a high number of iterations to reach stability. For single-core computers, a sequential reconstruction (SR) algorithm, based on forward and backward scanning that reduces the number of iterations, can be applied. Nevertheless, a hybrid reconstruction algorithm (HRA) based on queues [26] is the best tradeoff.

The SR algorithm is summarized in Algorithm 1. $N_G^+(p)$ and $N_G^-(p)$ are the backward (*left, left-up, up*, and *right-up*) and forward (*right, left-down, down*, and *right-down*) neighborhood of pixel p, respectively. A GPU implementation of the SR algorithm has been proposed in [21], where the scans are performed separately in different kernels. At each iteration, i.e., kernel execution, only one scan (forward or backward) is performed.

The implementation proposed in this paper is based on a block-asynchronous propagation [20], and unlike the SR algorithm, multiple scans are performed in both directions at the same time. Therefore, the data are reused in shared memory. The GPU proposal for the BAR algorithm is shown in Algorithm 2, where the kernel is placed between $< >$ symbols and BLOCK stands for the data region that is processed by a block of threads. In the algorithm, an iterative process executes the reconstruction until stability is reached. Each iteration performs one call to the reconstruction kernel (line 3 in the algorithm) and a global synchronization among blocks (line 4). These iterations are called inter-block updates. Threads are configured in 2-D blocks and threads within the block load data from global to shared memory. The kernel (lines 7–14) consists in a loop where the updating is performed (line 10) requiring only synchronization among the threads inside the block (line 11) using only shared memory. These iterations are called intra-block updates and are asynchronous among different blocks.

The main advantage of this GPU asynchronous implementation for the morphological reconstruction (consisting of intra- and inter-block updates) is that as many updates as possible are performed in shared memory with the available data before performing a global synchronization among thread blocks. Data updated within a block in shared memory can be reused, which is much faster than the global memory updates [25]. In this asynchronous proposal, the number of global synchronization is reduced compared with the synchronous version of the algorithm [20].

The update operation in the reconstruction operation is calculated according to

$$J(p) = \min(\max\{J(q), q \in N_G(p) \cup \{p\}\}, I(p)) \qquad (1)$$

where J is the marker image, I is the mask image, and $N_G(p) = \{N_G^+(p) \cup N_G^-(p)\}$ the complete neighborhood of pixel p. The forward and backward scans are joined in one step as defined in (1).

D. 2-D-DWT Denoising on GPU

The 2-D-DWT has widely been investigated for parallel processing on GPU [17]–[19]. In [19], a 2-D-DWT based on the properties of separable filters, as described in Section II-C, obtained the best results, compared to a lifting-based wavelet transform scheme. In the case of separable 2-D-DWT, the 1-D-DWT is extended by applying the wavelet analysis separately to each dimension. However, the 1-D-DWT directly applied by columns it is inefficient because the memory accesses to global memory are not coalesced and there is no data reuse in the cache.

The 2-D-DWT process of this paper uses the set of filters for perfect reconstruction presented in [24]. Three filters are used: one low-pass filter h and two high-pass filters g_1 and g_2. First, the 1-D-DWT by rows is applied to the original image to produce three temporal subbands (one for each filter) L, H1, H2, as shown in Fig. 2(a). The 2-D-DWT by columns applied to the temporal subbands produces nine new subbands corresponding to the LL, LH1, LH2, H1L, H1H1, H1H2, H2L, H2H1, and H2H2 subbands, as shown in Fig. 2(b). In order to implement this wavelet on GPU, the techniques described above must be adapted.

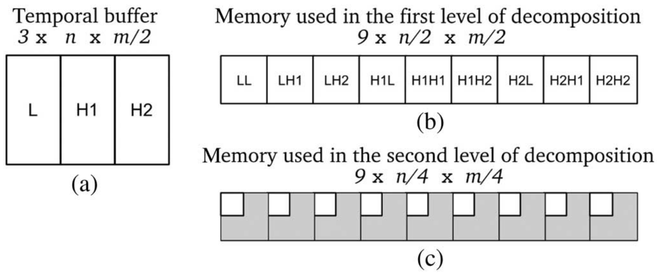
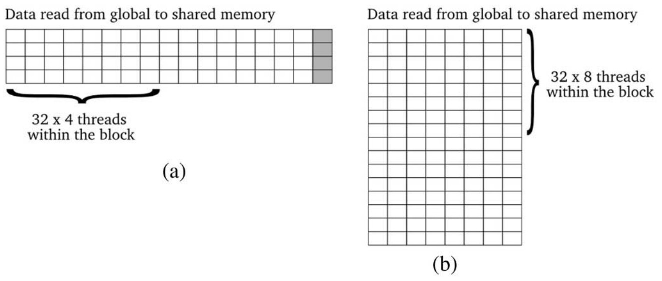


Fig. 2. 2-D-DWT global memory requirements (indicated with a white background), for an image of $n \times m$ pixels, using a set of three filters and two levels of wavelet transform. (a) Temporal buffer for separable wavelet transform. (b) and (c) memory used in the first and the second level of decomposition, respectively.



Data read from global to shared memory
32 x 4 threads within the block
(a)
Data read from global to shared memory
32 x 8 threads within the block
(b)

Fig. 3. Threads within a block and data read from global to shared memory in the separable 2-D-DWT by (a) rows and (b) columns. Each square represents 4×1 data items in the shared memory. The grey squares correspond to the data allocated in the extended shared memory used in the kernel in the case of the 2-D-DWT by rows.

1) Implementation: The 2-D-DWT denoising uses three filters of six coefficients each [24]. All are applied in the same kernel call, so data are reused among the filters. Threads within the block read two data items in shared memory. In the transformation by rows, the last threads of each block load additional extra data from global memory to shared memory, as illustrated in Fig. 3(a). The gray squares correspond to the extra data allocated in shared memory. To achieve this, the shared memory used per block is extended with four values, corresponding to the data necessary to compute all the convolution in shared memory. Note that the four rightmost threads of the block in Fig. 3(a) load three data items.

In the transformation by columns, each thread loads two data items into the shared memory as illustrated in Fig. 3(b). In this kernel, the threads of the first rows perform the convolution in shared memory, whereas the last threads perform the convolution directly with the data read from the global memory. The reason is that the shared memory requirements are higher in the 1-D-DWT by columns and increasing even more the shared memory use for all the threads would reduce the number of active blocks per SMX, decreasing the performance. The access pattern in global memory is coalesced, i.e., consecutive threads access to consecutive memory locations. In addition, if the number of threads within the block in the row dimension is multiple of 32, the GPU can execute the same instruction on different data without any divergence in the code.

After performing L decomposition levels, a soft-thresholding is applied to the detail coefficients at each level, corresponding to the last eight subbands in Fig. 2(b) and (c). The kernel is executed for each hyperspectral band after the forward 2-D-DWT. Each thread computes the soft-thresholding [27] directly in global memory.

After soft-thresholding, an inverse 2-D-DWT is applied first by columns and then by rows to reconstruct the denoised image.

2) Memory Requirements: Fig. 2 also shows the memory requirements of the 2-D-DWT implementation for an image of $n \times m$ pixels and two levels of wavelet transform. The first level of decomposition needs space for nine subbands of $n/2 \times m/2$ pixels, as shown in Fig. 2(b), and the second level needs space for nine subbands of $n/4 \times m/4$ pixels, as shown in Fig. 2(c). In order to simplify the memory access pattern and keep the data aligned in memory, $9 \times n/2 \times m/2 \times L$ subbands are allocated at the beginning of the process, with L the number of levels of decomposition applied to the image. The memory that is allocated but not used in the second and successive decompositions is, for the case of a 610×340 image, less than 1% of the total amount of memory available in a GPU with 1 GB of global memory. This memory space is shaded in gray in Fig. 2(c).

A temporal buffer of $3 \times n \times m/2$, see Fig. 2(a), is used for the results produced by the separable 2-D-DWT when it is applied by rows. All the memory allocated is reused for each hyperspectral band of the original image where the soft-thresholding is applied. The drawback of this 2-D-DWT in terms of memory requirements, compared to the case of two filters, is that the subbands cannot be stored in one quarter of the original image because nine bands are generated instead of the four bands of the general case. Therefore, extra global memory has been allocated to improve the memory access pattern and keep the data memory aligned.

E. SVM Classification on GPU

In this section, the classification by SVMs is first introduced and then the GPU implementation is detailed. Let us consider a set of N pairs $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$ with the pixel vectors $\mathbf{x}_i \in \Re^n$ and the corresponding class $y_i \in \{\pm 1\}$. The standard two-class SVM classifier consists in finding the optimal hyperplane, which separates the two classes, maximizing the minimum distance between any data point of both classes. This is known as the *training phase* of the SVM. The training samples that maximize the distance are called support vectors (SVs). When the training samples are not linearly separable, the SVM approach uses the *kernel trick* to map the data with a nonlinear transformation to a higher dimensional space, in order to find a linear separating surface between the two classes [7]. Several types of kernels, such as linear, polynomial, splines, and the Gaussian radial basis (RBF) are used in the SVM *classification phase*.

For example, the discriminant function to estimate the class of each pixel can be expressed using the RBF kernel as

$$f(\mathbf{x}) = \sum_{i \in S} \alpha_i y_i \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}\|^2) \tag{2}$$

where S is the set of SVs, α_i is the weight of each SV according to its importance in determining the class, $\exp(-\gamma \|\mathbf{x}_i - \mathbf{x}\|^2)$ is the RBF kernel, and γ is a parameter inversely proportional to the width of the kernel. All these parameters are obtained in the training phase. The final decision function to classify a pixel $\mathbf{x}$ into one of the classes is given by the sign of (2)

$$D(\mathbf{x}) = \{+1, \text{if } f(\mathbf{x}) > 0; -1 \text{ otherwise}\}. \tag{3}$$

When there are more than two classes, the multiclass SVM problem is solved one-against-one (OAO), one-against-all, or all-at-once [28]. Hsu and Lin [29] found that the OAO approach is more suitable for practical use than other methods, mainly because the training time is shorter. This multiclass method creates $T = K(K-1)/2$ binary classifiers on all pairs of classes, being K the number of classes. The final prediction for each pixel is given by the MV in the T classifiers.

Although different implementations of the SVM algorithm on GPU are available in the literature [8]–[10], a GPU implementation of the classification phase is proposed in this work. Regarding the training phase, it requires a small number of training samples and the GPU can not be fully exploited [11]. Therefore, the training phase has been executed in CPU using the LIBSVM library [30].

The GPU implementation of the SVM classification phase is as follows: data are copied to the GPU global memory at the beginning of the classification and they are properly aligned to get coalesced accesses. A loop in the host executes the RBF kernel T times, with T the number of binary classifiers necessary to solve the problem using the OAO approach. This kernel is executed in global memory. Each thread computes one pixel vector $\mathbf{x}$ directly in global memory according to (2). As the threads access the same SVs, the global memory accesses are broadcasted to the threads of the same block and also reused among different blocks. At the end of the loop the results of the T classifiers are kept in the global memory.

The second kernel computes (3) in registers and stores the result in shared memory. In this kernel, each thread processes one pixel. It counts the number of positive and negative votes among the binary classifiers. The decision $D(\mathbf{x})$ for each pixel vector is stored in shared memory and used in the same kernel to take the winner by an MV process. The voting corresponding to one pixel is computed also by one thread. Afterward, the final prediction is stored back in global memory.

This implementation of the SVM computes the multiclass problem at once completely on GPU. Other implementations such as [8] consider only the standard two-class SVM problem and require more time to compute a multiclass problem because they need to execute the two-class classification T times independently.

IV. Results

The proposed GPU implementation is evaluated in terms of execution times and speedups. As the GPU performance depends on several factors, such as the number of threads per block or the on-chip shared memory configuration, different configurations of these parameters are analyzed in Section IV-A. Based on the best configuration, the performance

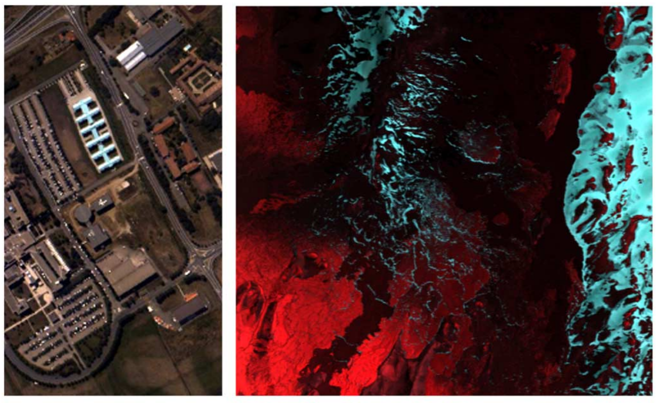


Fig. 4. Hyperspectral images used in the experiments. (a) University of Pavia. (b) Hekla volcano.

TABLE I
CPU AND GPU HARDWARE SPECIFICATIONS

Hardware	# Cores	Core clock (Mhz)	RAM (GB)	L1 (KB)	L2 (KB)	L3 (KB)
Intel Core i7	4	2800	8	64	256	8192
GTX TITAN	2688	837	6	16 / 48	1536	–

of the scheme is analyzed in Section IV-B for real-time hyperspectral image classification over two remote sensing images.[1] The first image was collected by the Reflective Optics System Imaging Spectrometer (ROSIS-03) sensor over the University of Pavia, Italy. The dimensions of this image are 610×340 pixels with 103 spectral bands and nine classes identified in its reference map. The second image was acquired by the AVIRIS sensor over the Hekla volcano in Iceland [31]. This data set has a spatial dimension of 560×600 pixels, 157 spectral bands and 12 classes in the reference map. The color representation of the images is shown in Fig. 4. The reference maps with the name of the classes, as well as the number of training samples can be found in [16] for Pavia and [32] for Hekla. A summary of the overall accuracy (OA), average accuracy (AA), and Kappa coefficient of agreement (k) [33] is also presented in this section to show the accuracy of the resulting GPU classification. The morphological reconstruction used in the EMP step is analyzed in detail in Section IV-C.

Table I summarizes the hardware specifications of the CPU and the GPU used in the experiments. The experiments were executed under Linux using the gcc compiler version 4.6.3 for the OpenMP implementation, and the nvcc compiler version 5.5 for the case of the CUDA implementation, respectively, with full optimization flags (-O3) in both cases. The floating point arithmetic operations have been executed in double precision except the morphological operations that use bytes as the base data type.

A. Analysis of the GPU Parameters

One factor used to measure the performance on the GPU is the *occupancy*, that is defined as the ratio of the number

[1]Additional information related to the hyperspectral images. [Online] Available:http://wiki.citius.usc.es/software/wt-emp-gpu

TABLE II
NUMBER OF ACTIVE BLOCKS PER SMX FOR THE DIFFERENT KERNELS BASED ON THE BLOCK SIZE AND THE SHARED MEMORY REQUIREMENTS

		128×1	32×4	32×8	32×16
1-D-DWT	L1	**16**	na	na	na
2-D-DWT (rows)	L1	na	7	3	1
2-D-DWT (columns)	L1	na	na	4	2
Soft-thresholding	L1	**16**	**16**	8	4
EMP	L1	na	**16**	8	4
SVM	L1	1	na	na	na
1-D-DWT	SM	**16**	na	na	na
2-D-DWT (rows)	SM	na	**16**	8	4
2-D-DWT (columns)	SM	na	na	**8**	4
Soft-thresholding	SM	**16**	**16**	8	4
EMP	SM	na	**16**	8	4
SVM	SM	**5**	na	na	na

L1 indicates that 48 kB are used for the L1 memory and 16 kB for the shared memory. SM states for the opposite configuration. Results for Pavia image. The best occupancy is indicated in bold.

of active warps per SMX to the maximum number of possible active warps. This metric can also be studied in the basis of the total number of concurrent blocks per SMX. The highest number of concurrent blocks is desired as it maximizes the occupancy while hides the latency of memory accesses. The hardware resources that usually limit the occupancy in the GPU are the registers usage, the shared memory requirements and the block size [34]. The GPU used in the experiments is based on the Kepler architecture. In particular, the GTX TITAN has 14 SMXs with the following limits per SMX: 2048 threads, 16 active blocks, 65 536 registers of 32 bits, and 64 kB of on-chip memory that can be configured as a shared memory of 16 and 48 kB for the L1 cache or the opposite configuration. The different kernels used in the GPU implementation have been analyzed according to these resources. The number of registers used per thread in the different kernels was always lower than the maximum number of registers available per thread.

Table II shows the maximum number of active blocks based on the block size and the on-chip memory configuration. The text "na" in the table indicates that the block configuration was not available for the kernel. For example, the 2-D-DWT by columns requires a minimum of eight threads in the second dimension in order to compute the convolution in shared memory. For the 1-D-DWT and the SVM kernels, the block size is adapted to the multiple of 32 nearest to the size of the pixel vector (128 for the image of Pavia).

From Table II can be observed that the 2-D-DWT (by rows) can only be executed by 16 active blocks, the maximum possible for the GTX TITAN, when the GPU is configured to use 48 kB of shared memory. The reason is that the kernel requires 2176 bytes of shared memory per block. Therefore, in order to have 16 active blocks per SMX, 34 816 bytes of shared memory are required for this kernel.

The kernel computing the 2-D-DWT (by columns) is limited by the amount of shared memory and the maximum number of threads per SMX. This kernel uses 4 kB of shared memory. In the SM configuration, there is enough shared memory for 12 blocks. However, as the block size is 32×8, i.e., 256 threads per block, the limit of 2048 threads per SMX is reached before

TABLE III
EXECUTION TIMES IN SECONDS AND SPEEDUPS FOR EACH STEP OF THE WT-EMP SCHEME IN THE CPU USING OPENMP (FOUR THREADS) AND IN THE GPU USING CUDA

	University of Pavia			Hekla		
	OMP	GPU	Speedup	OMP	GPU	Speedup
Data transfer	–	0.071	–	–	0.164	–
1-D-DWT	0.259	0.009	28.7×	0.630	0.021	30×
2-D-DWT	0.748	0.221	3.4×	1.644	0.421	3.9×
EMP	0.714	0.104	6.9×	0.766	0.100	7.6×
SVM	12.611	1.344	9.4×	12.515	1.755	7.1×
TOTAL	14.342	1.749	8.2×	15.555	2.461	6.3×

TABLE IV
EXECUTION TIMES IN SECONDS FOR THE WT-EMP AND THE WSHED-MV SCHEMES ON GPU, EXCEPTS THE TRAINING PHASE PERFORMED IN CPU WITH THE LIBSVM LIBRARY

	University of Pavia		Hekla	
	WT-EMP	WSHED-MV	WT-EMP	WSHED-MV
Preprocess	0.334	0.168	0.542	0.368
LIBSVM (train)	0.398	0.576	0.104	0.098
SVM (classify)	1.415	7.696	1.919	44.871
Postprocess	–	<1 ms	–	<1 ms
TOTAL	2.147	8.272	2.565	45.337

TABLE V
ACCURACIES IN PERCENTAGE FOR THE WT-EMP SCHEME OVER THE HYPERSPECTRAL IMAGES USED IN THE EXPERIMENTS

WT-EMP	OA	AA	k
University of Pavia	98.8	99.0	98.0
Hekla	98.0	98.8	97.7

The CPU and GPU results are exactly the same.

the limit of shared memory, resulting in a maximum number eight active blocks.

The SVM kernel is also limited by the shared memory requirements. In this case, as shown in Table II, a higher number of concurrent blocks is achieved with the SM configuration.

B. Performance Over Real Hyperspectral Images

This section presents the results obtained for the best configuration of each kernel based on the analysis of Section IV-A. In the case of having the same number of active blocks in both configurations, the L1 configuration is preferred in order to reduce the global memory accesses. The soft-thresholding kernel was configured with blocks of 32×4 threads. The execution time for the different steps of the WT-EMP scheme are presented in Table III. This table also includes the time for the initial data transfer of the hyperspectral image to the GPU global memory. The size of the data transferred from the CPU to the GPU is 162.9 MB for the University of Pavia and 402.5 MB for the Hekla image considering double precision. The speedup of the GPU implementation is calculated comparing to an optimized CPU multithreaded implementation using OpenMP. Each execution time is calculated as the average of 10 executions.

The first thing to note is that the GPU total execution time is below 2.5 s. This result represents a speedup of 8.2× and 6.3× for the University of Pavia and the Hekla images, respectively, including the data transfer from the CPU to the GPU.

Regarding the 1-D-DWT, speedup values of 28.7× and 30× are obtained for the two images. These improvements are mainly obtained through the use of the shared memory. All the 1-D-DWT levels of decomposition are applied in the same kernel call; therefore, data are loaded once from global memory to shared memory and reused at each level of the wavelet decomposition. The speedups obtained for the 2-D-DWT implementation, 3.4× and 3.9×, respectively, for each image, are within the expected values for a separable 2-D-DWT convolution by rows and columns. Similar speedups were obtained in [17]–[19] by comparing OpenMP and GPU implementations. Note that in Table III, the time required for the soft-thresholding is also included in the 2-D-DWT step. The EMP obtained speedups of 6.9× and 7.6×, mainly due to the block-asynchronous implementation. The performance of this step is studied in detail in Section IV-C.

The execution time of the SVM classification takes approximately 80% of the overall time. This is the most computationally expensive step of the scheme and it has been speeded up 9.4× and 7.1× for the University of Pavia and the Hekla images, respectively.

The execution time has also been compared to another GPU spectral–spatial classification scheme [11] based on segmentation by watershed and majority vote (WSHED-MV). In the WSHED-MV scheme, the classification is carried out by the GPU implementation of the SVM called GPUSVM of [8]. The experiments of [11] have been repeated in the same conditions but using the GTX TITAN GPU. For these experiments, the values for the parameters were taken from [16] and [32] for the Pavia and the Hekla images, respectively. Table IV shows the execution times, including the data transfer as part of the SVM step. The SVM training has been performed in CPU using the LIBSVM library [30] in both classification schemes. The preprocessing in the WT-EMP takes more time than in the WSHED-MV scheme because the number of steps and the complexity of the operations is higher in this scheme. The main difference in the execution times relies on the SVM classification, that also dominates the total time in the WSHED-MV. This scheme takes 45.337 s for the image of Hekla. This time is considered above the real-time performance for the AVIRIS sensor, which collects 512 full pixel vectors in 8.3 ms [35]. Therefore, the limit can be established in 4.98 s for an image of 512×600 pixels. In the WT-EMP scheme, with a time of 2.565 s for the same image, there may be room for additional pre- and post-processing.

The OA, AA, and k values presented in Table V shows the classification accuracy results of the GPU implementation, which is exactly the same than for the CPU one. A complete study of the classification for remote sensing images using the WT-EMP scheme was presented in [16].

C. Analysis of the Morphological Reconstruction on GPU

The morphological reconstruction used to create the EMP has been analyzed separately because the block-asynchronous implementation is a novelty regarding the state of the art in

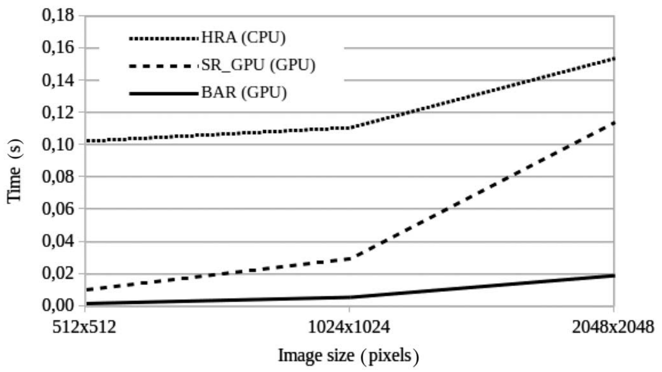


Fig. 5. Execution time of the morphological reconstruction for the HRA, the SR_GPU, and the BAR algorithms for the image of Hekla.

mathematical morphology on GPU. For this purpose, a subset of 512×512 pixels of the image of Hekla has been used in this experiment. This subset was scaled twice maintaining the number of regions in order to study the performance of the implementation when the image size is increased. The GPU BAR algorithm is compared to the Fast Hybrid Reconstruction (HRA) algorithm in CPU [26], and the GPU Sequential Reconstruction (SR_GPU) algorithm proposed in [21]. The marker image J has been created as $J(p) = max\{I(p) - h, 0\}$, that is known as the hmax transform, as in [21]. A value $h = 10$ was used to create the initial marker.

The results for the three algorithms are shown in Fig. 5. Both, SR_GPU and BAR algorithms include the data transfer from CPU to GPU memory. It can be observed that the GPU proposals overcome the HRA algorithm. The GPU BAR algorithm presents a better performance when the image size increases. By performing multiple scans in shared memory at each iteration, this morphological reconstruction in the GPU efficiently exploits the shared memory through the block-asynchronous updating process [20]. The main difference to the SR_GPU algorithm relies on the forward and backward scans. The SR_GPU scans the full image at each iteration in one direction (forward or backward), while the BAR scans the image by blocks, and in both directions at the same time.

V. Conclusion

In this paper a spectral–spatial classification scheme (WT-EMP) of hyperspectral images based on wavelets and mathematical morphology has been specifically adapted for efficient GPU computation. The different steps of the scheme, 1-D-DWT feature reduction, EMP, and 2-D-DWT denoising, were specifically designed for exploding the hardware available in these architectures.

The proposed 1-D-DWT implementation was adapted to compute thousands of transformations in parallel. The different levels of the wavelet decomposition were applied in the same kernel reusing the data of the shared memory. In the case of the 2-D-DWT, the implementation was designed with the purpose of managing the three filters that are used in the denoised step, and the number of times that data were loaded within the kernel was minimized by computing the three filters in the same kernel and by loading larger blocks of data.

A hybrid iterative updating process, adequate for asynchronous computation, was successfully applied to the EMP. The proposed BAR algorithm performs intra-block updates (asynchronous updates that reuse shared memory) and inter-block updates (that require global synchronization operations).

The implementation of the SVM classification stage computed the multiclass problem completely on GPU. Other implementations that consider only the standard two-class SVM problem need to execute the classification $K(K-1)/2$ times independently, being K the number of classes.

The scheme was tested on two real hyperspectral images, captured by the ROSIS-03, and the AVIRIS hyperspectral sensors. As a result, the execution time was speeded up $8.2\times$ in the classification of the Pavia image compared to an OpenMP implementation.

By optimizing the kernels to efficiently exploit the GPU resources, the proposed implementation achieved real-time classification of the considered hyperspectral images.

Acknowledgment

The authors would like to thank P. Karas (Masaryk University) for providing the code of the SR_GPU algorithm [21]. The ROSIS-03 data were obtained from Prof. P. Gamba of the University of Pavia, Italy.

References

[1] J. A. Benediktsson, J. Chanussot, and W. M. Moon, "Very high-resolution remote sensing: Challenges and opportunities," *Proc. IEEE*, vol. 100, no. 6, pp. 1907–1910, Jun. 2012.

[2] M. Fauvel, Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton, "Advances in spectral-spatial classification of hyperspectral images," *Proc. IEEE*, vol. 101, no. 3, pp. 1–24, Mar. 2012.

[3] D. B. Heras, F. Argüello, J. L. Gómez, J. A. Becerra, and R. J. Duro, "Towards real-time hyperspectral image processing, a GP-GPU implementation of target identification," *Proc. IEEE 6th Int. Conf. Intell. Data Acquisit. Adv. Comput. Sys. (IDAACS)*, 2011, pp. 316–321.

[4] A. Plaza, J. Plaza, and H. Vegas, "Improving the performance of hyperspectral image and signal processing algorithms using parallel, distributed and specialized hardware-based systems," *J. Signal Process. Syst.*, vol. 61, no. 3, pp. 293–315, 2010.

[5] C. González *et al.*, "Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing," *VLSI J. Integr.*, vol. 46, no. 2, pp. 89–103, 2013.

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

[7] J. A. Gualtieri and R. F. Cromp, "Support vector machines for hyperspectral remote sensing classification," in *Proc. SPIE*, vol. 3584, pp. 221–232, 1998.

[8] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *Proc. 25th Int. Conf. Mach. Learn. (ICML'08)*, 2008, pp. 104–111.

[9] S. Herrero-López, J. R. Williams, and A. Sanchez, "Parallel multiclass classification using SVMs on GPUs," in *Proc. ACM 3rd Workshop Gener. Purpose Comput. Graph. Process. Units*, 2010, pp. 2–11.

[10] Q. Li, R. Salman, E. Test, R. Strack, and K. Kecman, "GPUSVM: A comprehensive CUDA-based support vector machine package," *Central Eur. J. Comput. Sci.*, vol. 1, no. 4, pp. 387–405, 2011.

[11] P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Computing efficiently spectral–spatial classification of hyperspectral images on commodity GPUs," in *Recent Advances in Knowledge-Based Paradigms and Applications*. New York, NY, USA: Springer, 2014, vol. 234, pp. 19–42.
[12] 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.
[13] S. Bernabé, A. Plaza, P. Reddy Marpu, and J. A. Benediktsson, "A new parallel tool for classification of remotely sensed imagery," *Comput. Geosci.*, vol. 46, pp. 208–218, 2012.
[14] J. A. Benediktsson, J. A. Palmason, and J. R. Sveinsson, "Classification of hyperspectral data from urban areas based on extended morphological profiles," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 480–491, Mar. 2005.
[15] M. Fauvel, J. A. Benediktsson, J. Chanussot, and J. R. Sveinsson, "Spectral and spatial classification of hyperspectral data using SVMs and morphological profiles," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 11, pp. 3804–3814, Nov. 2008.
[16] P. Quesada-Barriuso, F. Argüello, and D. B. Heras "Spectral–spatial classification of hyperspectral images using wavelets and extended morphological profiles," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 4, pp. 1177–1185, Apr. 2014.
[17] J. Franco, G. Bernabé, J. Fernández, and M. Ujaldón, "The 2-D wavelet transform on emerging architectures: GPUs and multicores," *J. Real-Time Image Process.*, vol. 7, no. 3, pp. 145–152, 2011.
[18] F. Argüello, D. B. Heras, M. Bóo, and J. Lamas-Rodríguez, "The split-and-merge method in general purpose computation on GPUs," *Parallel Comput.*, vol. 38, no. 6–7, pp 277–288, 2012.
[19] V. Galiano, O. López, M. P. Malumbres, and H. Migallón, "Improving the discrete wavelet transform computation from multicore to GPU-based algorithms,"in *Proc. Int. Conf. Comput. Math. Methods Sci. Eng.*, 2011, pp. 544–555.
[20] P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient 2-D and 3-D watershed on graphics processing unit: Block-asynchronous approaches based on cellular automata," *Comput. Electr. Eng.*, vol. 39, no. 8, pp 2638–2655, 2013.
[21] P. Karas, "Efficient computation of morphological greyscale reconstruction," *Proc. 6th Doctoral Workshop Math. Eng. Methods Comput. Sci.*, 2011, vol. 16, pp. 54–61.
[22] I. Daubechies, "Ten lectures on wavelets," *Society Indus. Appl. Math.*, vol. 61, pp. 198–202, 1992.
[23] M. Vetterli and C. Herley, "Wavelets and filter banks: Theory and design," *IEEE Trans. Signal Process.*, vol. 40, no. 9, pp. 2207–2232, Sep. 1992.
[24] A. F. Abdelnour and I. W. Selesnick, "Nearly symmetric orthogonal wavelet bases," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2001, vol. 6, pp. 3693–3696.
[25] D. Kirk and W. Hwu Wen-mei, *Programming Massively Parallel Processors: A Hands-on Approach*. Amsterdam, The Netherlands: Elsevier, 2010.
[26] L. Vincent, "Morphological grayscale reconstruction in image analysis: Applications and efficient algorithms," *IEEE Trans. Image Process.*, vol. 2, no. 2, pp.176–201, Apr. 1993.
[27] D. L. Donoho, "Denoising by soft-thresholding," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 613–627, Mar. 1995.
[28] A. Shigeo, *Support Vector Machines for Pattern Classification.* New York, NY, USA: Springer, 2005.
[29] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Netw*., vol. 13, 415–425, Mar. 2002.
[30] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 27:1–27:27, 2011.
[31] J. A. Benediktsson and I. Kanellopoulos, "Classification of multisource and hyperspectral data based on decision fusion," *IEEE Trans. Geosci. Remote Sens.*, vol. 37, no. 3, pp. 1367–1377, May 1999.
[32] K. Bernard, Y. Tarabalka, J. Angulo, J. Chanussot, and J. A. Benediktsson, "Spectral–spatial classification of hyperspectral data based on a stochastic minimum spanning forest approach," *IEEE Trans. Image Process.*, vol. 21, no. 4, pp. 2008–2021, Apr. 2012.
[33] A. J. Viera and J. M. Garrett, "Understanding interobserver agreement: The kappa statistic," *Family Med.*, vol. 37, no. 5, pp. 360–363, 2005.
[34] J. Luitjens and S. Rennich, *CUDA Warps and Occupancy*, GPU Computing Webinars (NVIDIA Corporation), 2011 [Online]. Available: https://developer.nvidia.com/gpu-computing-webinars
[35] A. Plaza, J. Plaza, A. Paz, and S. Sánchez, "Parallel Hyperspectral Image and Signal Processing," *IEEE Signal Process. Mag.*, vol. 28, no. pp. 119–126, Apr. 2011.

Pablo Quesada-Barriuso received the B.S. degree in computer science from the University of Las Palmas de Gran Canaria, Spain, in 2007 and the M.S. degree in graphics, games, and virtual reality from the University Rey Juan Carlos, Spain, in 2010. He is currently pursuing the Ph.D. degree in computer science at the University of Santiago de Compostela, Spain.

His research interests include image processing, parallel algorithms, and GPUs.

Francisco Argüello received the B.S. and Ph.D. degrees in physics from the University of Santiago, Santiago, Spain, in 1988 and 1992, respectively.

He is currently an Associate Professor with the Department of Electronic and Computer Engineering, University of Santiago. His research interests include signal and image processing, computer graphics, parallel and distributed computing, and quantum computing.

Dora B. Heras received the M.S. degree in physics and the Ph.D. degree in physics from the University of Santiago, Santiago, Spain, in 1994 and 2000, respectively.

She is currently an Associate Professor with the Department of Electronics and Computer Engineering, University of Santiago. Her research interests include parallel and distributed computing, software optimization techniques for emerging architectures, computer graphics, and image processing.

Jón Atli Benediktsson received the Cand.Sci. degree in electrical engineering from the University of Iceland, Reykjavik, Iceland, in 1984, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1987 and 1990, respectively.

He is currently a Pro Rector for Academic Affairs, Eugene, OR, USA, and a Professor of Electrical and Computer Engineering with the University of Iceland. His research interests include remote sensing, biomedical analysis of signals, pattern recognition, image processing, and signal processing and he has authored extensively in these fields.

Prof. Benediktsson was the 2011–2012 President of the IEEE GEOSCIENCE AND AND REMOTE SENSING SOCIETY (GRSS) and has been on the GRSS AdCom since 2000. He was the Editor in Chief of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING (TGRS) from 2003 to 2008, and has served as an Associate Editor of TGRS since 1999, the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS since 2003 and IEEE Access since 2013. He is the Editorial Board of the PROCEEDINGS OF THE IEEE and the International Editorial Board of the *International Journal of Image and Data Fusion.* He was the Chairman of the Steering Committee of IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (J-STARS) from 2007 to 2010. He is a Cofounder of the Biomedical Start up Company Oxymap. He is a Fellow of the SPIE, and a member of the 2014 IEEE Fellow Committee. He is a member of the Association of Chartered Engineers in Iceland (VFI), Societas Scinetiarum Islandica, and Tau Beta Pi. He was the recipient of the Stevan J. Kristof Award from Purdue University in 1991 as an outstanding graduate student in remote sensing. In 1997, he was the recipient of the Icelandic Research Council's Outstanding Young Researcher Award, in 2000, he was granted the IEEE Third Millennium Medal, in 2004, he was the corecipient of the University of Iceland's Technology Innovation Award, in 2006, the Yearly Research Award from the Engineering Research Institute, University of Iceland, and in 2007, the Outstanding Service Award from the IEEE Geoscience and Remote Sensing Society. He was the corecipient of the 2012 IEEE Transactions on Geoscience and Remote Sensing Paper Award, and in 2013, the IEEE GRSS Highest Impact Paper Award. In 2013, he was the recipient of the IEEE/VFI Electrical Engineer of the Year Award. In 2014, he was the corecipient of the International Journal of Image and Data Fusion Best Paper Award.