Wavelet-based multicomponent denoising on GPU to improve the classification of hyperspectral images

Pablo Quesada-Barriuso, Dora B. Heras, Francisco Argüello, J. C. Mouriño

Wavelet-based Multi-component Denoising on GPU to Improve the Classification of Hyperspectral Images

Pablo Quesada-Barriuso[a], Dora B. Heras*[a], Francisco Argüello[b], and J. C. Mouriño[c]

[a]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)
[b]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela
[c]Fundación Pública Galega Centro Tecnológico de Supercomputación de Galicia (CESGA)
Spain

ABSTRACT

Supervised classification allows handling a wide range of remote sensing hyperspectral applications. Enhancing the spatial organization of the pixels over the image has proven to be beneficial for the interpretation of the image content, thus increasing the classification accuracy. Denoising in the spatial domain of the image has been shown as a technique that enhances the structures in the image. This paper proposes a multi-component denoising approach in order to increase the classification accuracy when a classification method is applied. It is computed on multicore CPUs and NVIDIA GPUs. The method combines feature extraction based on a 1D-discrete wavelet transform (DWT) applied in the spectral dimension followed by an Extended Morphological Profile (EMP) and a classifier (SVM or ELM). The multi-component noise reduction is applied to the EMP just before the classification. The denoising recursively applies a separable 2D DWT after which the number of wavelet coefficients is reduced by using a threshold. Finally, inverse 2D-DWT filters are applied to reconstruct the noise free original component. The computational cost of the classifiers as well as the cost of the whole classification chain is high but it is reduced achieving real-time behavior for some applications through their computation on NVIDIA multi-GPU platforms.

Keywords: Land cover classification, Hyperspectral analysis, Wavelet transform, Denoising, Spectral-spatial processing, High-Performance computing, Multi-thread, Multi-GPU.

1. INTRODUCTION

In remote sensing, the hundreds of spectral bands acquired by sensors like the Reflective Optics System Imaging Spectrometer[1] offer the possibility of spectral analysis for different applications, such as target recognition, mineral exploration, agricultural assessment and land-cover classification.[2] The high spectral and spatial resolution of the hyperspectral sensors has introduced new challenges in those applications. To deal with these challenges such as the highly correlated features and redundancy in the spectral domain or the Hughes phenomenon,[3] feature extraction/reduction (FE/FR) methods, spatial processing techniques or denoising processes are usually introduced.

Among the different methods for FE/FR that have been applied to hyperspectral images, principal component analysis (PCA) is widely used in remote sensing for computing the maximum amount of data variance in a new uncorrelated data set.[4] Independent component analysis (ICA) and Minimum noise fraction (MNF), as well as wavelet transform methods have also been applied for feature reduction.[4–7] For example, a one dimensional wavelet transform (1D-DWT) has been proven to obtain similar classification accuracy to PCA but with a lower computational cost.[6]

Hyperspectral sensors usually introduce undesirable artifacts in the acquisition of the images due to improper calibration or atmospheric phenomena that can be considered as noise in the image.[8] Wavelets have also been applied with denoising purposes improving the classification accuracy as compared to the result obtained from

* Further author information: (Send correspondence to Dora B. Heras. E-mail: dora.blanco@usc.es)
CiTIUS - Rúa Jenaro de la Fuente Domínguez, s/n, 15782 - Santiago de Compostela, La Coruña, Spain.

High-Performance Computing in Geoscience and Remote Sensing VII, edited by Bormin Huang, Sebastián López, Zhensen Wu, Jose M. Nascimento, Jun Li, Valeriy V. Strotov, Proc. of SPIE Vol. 10430, 104300C
© 2017 SPIE · CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2277960

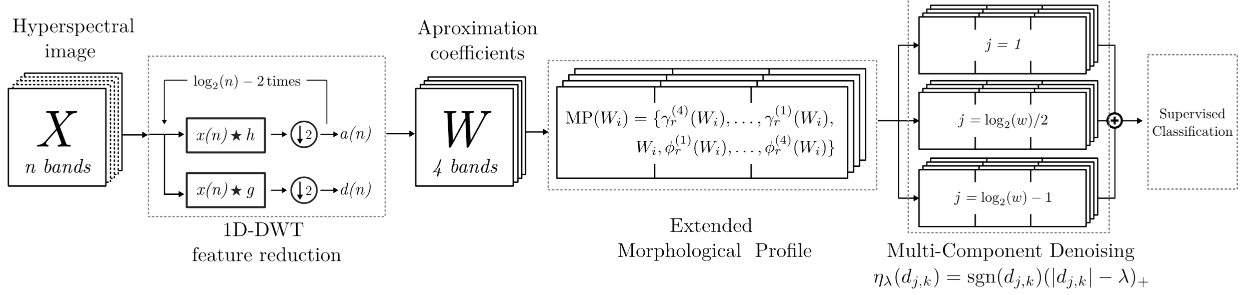

Hyperspectral image
X
n bands
$\log_2(n) - 2$ times
$x(n) \star h$
$\downarrow 2$
$a(n)$
$x(n) \star g$
$\downarrow 2$
$d(n)$
1D-DWT feature reduction
Aproximation coefficients
W
4 bands
$\mathrm{MP}(W_i) = \{\gamma_r^{(4)}(W_i), \ldots, \gamma_r^{(1)}(W_i), W_i, \phi_r^{(1)}(W_i), \ldots, \phi_r^{(4)}(W_i)\}$
Extended Morphological Profile
$j = 1$
$j = \log_2(w)/2$
$j = \log_2(w) - 1$
Multi-Component Denoising
$\eta_\lambda(d_{j,k}) = \mathrm{sgn}(d_{j,k})(|d_{j,k}| - \lambda)_+$
Supervised Classification

Figure 1. Flow-chart of the proposed wavelet-based multi-component denoising approach (W-MCD) including a last stage of classification.

the original hyperspectral image.[7,9,10] In Ref. 10, 2D wavelet-based denoising is applied in the spatial domain, i.e. applied band by band of the image, which is then classified by using a support vector machine (SVM).

Denoising by wavelets has also been investigated in the spectral domain[11] (1D), combining 1D and 2D wavelet transforms,[12,13] as well as applied directly to the hyperspectral cube in 3D.[14,15] The method in Ref. 14 (SarUWT) is based on sparse regularization with a 3D undecimated wavelet transform while the BiShri3D[15] method directly uses a 3D wavelet-based approach using neighboring pixels. The authors in Ref. 13 improve the accuracy results by performing not only a 2D denoising for every spectral band, but also a previous additional 1D spectral signature denoising applied to each pixel vector of the image. Then the method combines the denoised hyperspectral image with spatial features extracted also from wavelets improving the classification accuracy obtained by a SVM.

Remote sensing hyperspectral applications are computationally costly, therefore, good candidates to be projected in high performance computing infrastructures such as clusters or specialized hardware devices.[16] GPUs provide a cost-efficient approach for the processing of remote sensing hyperspectral data for performing operations such as hyperspectral unmixing or classification, among others.[17]

In this paper we propose a denoising approach that consists in applying denoising in the spatial domain but not to the original image but to a extended morphological profile of the image built from features extracted also by wavelets. The method increases the accuracy of the classification obtained by using this initial denoising process. This is a Wavelet-based Multi-Component Denoising (W-MCD) method for hyperspectral image classification based on three stages: feature extraction by wavelets, spatial processing by the EMP and multi-component denoising based on a 2D-DWT. We propose efficient multicore CPU and CUDA multi-GPU implementations of the proposed denoising approach in order to achieve the real-time processing that is required in some aplications.

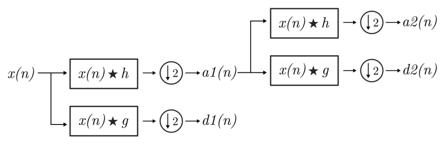
The remainder of the paper is organized as follows. Section 2 describes the three stages of the proposed approach. Section 3 summarizes the GPU architecture and describes the implementation details of the approach for multi-GPU platforms. Section 4 discusses the classification accuracy results over different hyperspectral images and compares the execution times to a multi-thread implementation in a supercomputing cluster. Finally, Section 5 presents the conclusions and future works.

2. WAVELET-BASED MULTI-COMPONENT DENOISING SCHEME

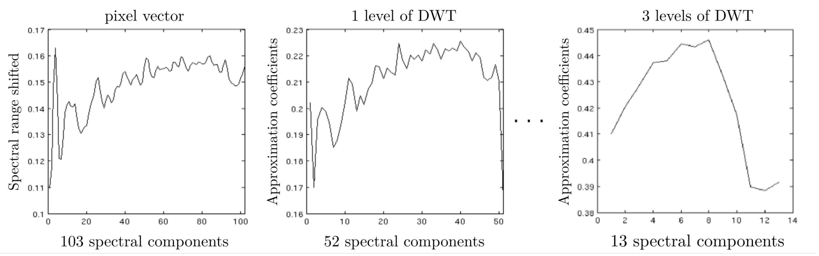
In this section we present the wavelet-based multi-component denoising scheme for classifying hyperspectral images. The proposal combines an Extended Morphological Profile (EMP) built from the features extracted by wavelets in the spectral domain with a spatial noise reduction method applied to each component of the EMP by a 2D-DWT. Figure 1 shows the flow-chart of this approach including a final classification stage. In the following the different stages are described in detail.

2.1 1D-DWT Feature Extraction

Wavelets are mathematical tools for signal processing analysis at different scales.[18] A signal can be approximated and its details highlighted by wavelet transforms. The 1D-discrete wavelet transform (DWT) of a signal $x(n)$

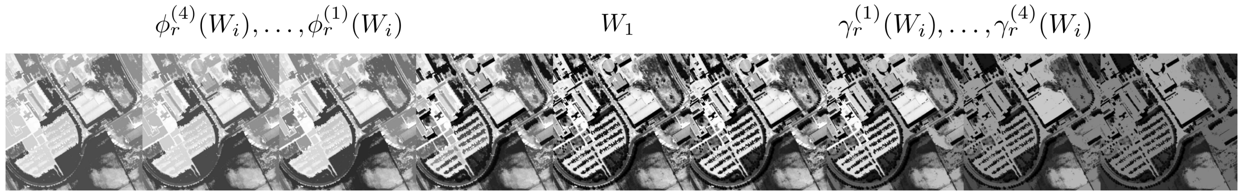


(a)



(b)

Figure 2. 1D-discrete wavelet transform: Mallat-tree decomposition (a); wavelet decomposition applied recursively to a pixel vector of 103 spectral components (b).



$\phi_r^{(4)}(W_i),\ldots,\phi_r^{(1)}(W_i)$
W_1
$\gamma_r^{(1)}(W_i),\ldots,\gamma_r^{(4)}(W_i)$

Figure 3. Morphological profile for the first coefficient band (W_1) extracted from a hyperspectral image known as Pavia University (only a part of the original image is shown in this figure). The MP was built using a disk of radius of increasing size $r \in [1, 3, 5, 7]$. The left and right side show the closing (ϕ_r) and the opening (γ_r), respectively.

can be calculated as the convolution of the signal and a low-pass h and a high-pass g filters[19] as:

$$a(n) = \sum_k h(k)\, x(i-k) = x(n) \star h \;, d(n) = \sum_k g(k)\, x(i-k) = x(n) \star g \;. \tag{1}$$

This wavelet decomposition uses a circular discrete convolution ($\star$) to produce $a(n)$, the approximation coefficients, and $d(n)$ the details coefficients. The signal is down-sampled at each level to produce half the coefficients. This is called the Mallat algorithm or Mallat-tree decomposition[20] and it is illustrated in Fig. 2(a). If a 1D-DWT is applied recursively in the spectral domain over a a hyperspectral image X, the number of spectral components is reduced at each new level of decomposition, as illustrated in Fig. 2(b). In the proposed approach, the 1D-DWT is applied recursively in the spectral domain to extract a small number of spectral components. The features extracted by wavelets are arranged as a hyperspectral cube W of coefficient bands before the next stage.

2.2 Extended Morphological Profile

Mathematical morphology allows extracting spatial structures from images by combining morphological transformations.[21] In particular, the Morphological Profile (MP) contains information on the spatial structures on the image at different scales by using morphological operators, such as the opening (γ_r) and the closing (ϕ_r) by reconstruction. The opening by reconstruction flattens the bright objects of the image, while the closing by reconstruction has the opposite effect. These operators are applied at pixel level using a structuring element (SE) of known shape (usually a square or a disk) followed by a geodesic morphological reconstruction.[22] This reconstruction completely preserves the spatial structures of the image if the SE fits within the objects and removes them otherwise. The second stage of our approach builds a MP from each coefficient band W_i extracted from the previous stage leading to the so called Extended Morphological Profile (EMP).

Let W_i be the set of wavelet coefficients in the ith coefficient band, the wavelet-based MP is built as:

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

where n is the number of morphological operations applied to the set W_i at different scales increasing the size of the structuring element. The EMP is created from each MP as follows:

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

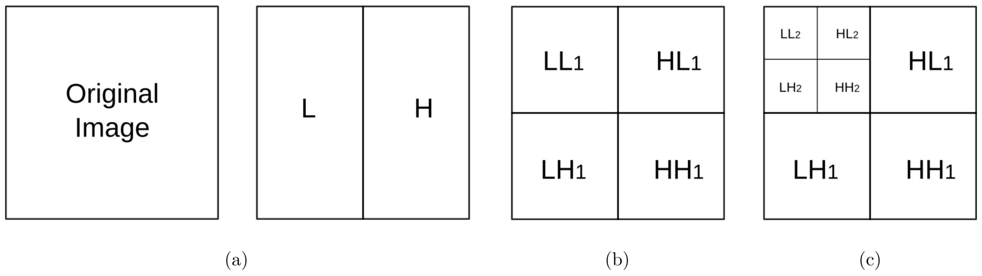


Figure 4. Separable 2D-discrete wavelet transform: Original image and wavelet decomposition by rows (a), wavelet transform applied to L and H by columns (b), and applied recursively by rows and columns to the original image two times to increase the decomposition of the image.

where m is the number of coefficient bands extracted in the first stage. Fig. 3 shows the morphological profile for the first coefficient band (W_1) extracted from a hyperspectral image known as Pavia University. The MP in Fig. 3 was built using four openings (γ_r) and closings (ϕ_r) by reconstruction with a disk of radius of increasing size $r \in [1, 3, 5, 7]$.

2.3 Multi-component Denoising

The third stage of the proposed approach (see Figure 1) applies a wavelet-based multi-component denoising method to the extended morphological profile at different scales using a universal threshold.

Wavelet-based denoising or wavelet thresholding is a common task performed by wavelets. The 1D-DWT described in Sect. 2.1 can be extended to bidimensional images by applying separately Eq. (1) to each dimension. The wavelet decomposition is applied first to one of the dimensions (by rows) resulting in two subbands L and H. These subbands represent the approximation and detail coefficients, respectively, as illustrated in Fig. 4(a). Then, Eq. (1) is applied to L and H in the second dimension (by columns), which results in four subbands LL, HL, LH, HH, corresponding to a low resolution approximation (LL) of the original image and three subbands of details (HL, LH, HH), as shown in Fig. 4(b). The 2D-DWT are recursively applied as shown in Fig. 4(c) to increase the decomposition of the image.

The smallest high frequency detail coefficients $d(n)$ produced by the wavelet transform (see Eq. (1) in Sect. 2.1) are usually considered noise.[13] These coefficients may be removed applying a thresholding formula without substantially affecting the main features of the signal. The idea is to preserve only the details that are above a particular threshold λ. By applying an inverse wavelet transform (IWT) after thresholding, it is possible to reconstruct the original signal with a lower amount of noise.[23] The following nonlinear transformation is used for wavelet thresholding:[24]

$$\eta_\lambda(d_{j,k}) = \operatorname{sgn}(d_{j,k})(|d_{j,k}| - \lambda)_+ \,, \tag{2}$$

where λ is the threshold and $d_{j,k} \in \{\mathrm{HL} \cup \mathrm{LH} \cup \mathrm{HH}\}$ is the kth detail wavelet coefficient at level j to be denoised. The subscript $+$ in Eq. (2) indicates that the coefficients $|d_{j,k}| > \lambda$ are attenuated, whereas the rest are set to zero. Equation (2) is known as soft-thresholding. The noisy free wavelet coefficients can remain unchanged instead of attenuated which is known as hard-thresholding.

Different algorithms can be used to estimate threshold values. Donoho[24] proposed a universal threshold for situations where the noise level is unknown. The universal threshold is calculated as:

$$\lambda = \sigma\sqrt{2\log n}, \tag{3}$$

where n is the length of the signal and σ is the noise standard deviation estimated by $\sigma = \frac{\mathrm{median}(|\mathrm{HH}_1|)}{0.6745}$, with HH_1 the detail wavelet coefficients obtained at the first level of decomposition. Figure 4(c) shows a multi-level 2D wavelet decomposition illustrating the detail coefficients at level one (HH_1) and level two (HH_2).

In Eq. (2), the noise standard deviation (σ) is calculated from the detail wavelet coefficients obtained at the first level of decomposition (HH_1) but the soft-thresholding is applied to all the coefficients $d_{j,k}$ at the different levels j. In this approach, each component of the EMP is denoised using the universal threshold with a 2D-DWT and the maximum number of levels of wavelet decomposition that can be applied to the image. The wavelet-based multi-component denoising approach can be summarized as follows:

1. Transform each component of the extended morphological profile into an orthogonal domain applying a 2D-DWT at levels $j \in \{1, \log_2(w)/2, \log_2(w) - 1\}$, being w the width of the image.
2. Apply soft-thresholding to the detail wavelet coefficients at each level of decomposition.
3. Perform an inverse 2D-DWT to reconstruct each level of decomposition separately.

In the case shown in Figure 1, the approach produces 3 denoised images for each component of the EMP because 3 levels of wavelet are applied. The information is combined in a new stack of features which is finally classified by a supervised method.

3. PARALLEL IMPLEMENTATION ON GPU

This section briefly describes the GPU architecture focusing on highlighting the challenges of GPU programming. Then, the different stages of the proposed approach, corresponding to the feature extraction by wavelets, the spatial processing by the EMP and the multi-component denoising based on 2D-DWT, are described in detail in the following sections.

3.1 GPU Architecture Overview

The GPU provides massively parallel processing capabilities with a high computational throughput due to a large number of cores organized in a set of streaming multiprocessors (SMXs). NVIDIA developed a Compute Unified Device Architecture (CUDA) based on a Single Instruction, Multiple Thread (SIMT) parallel programming model making it possible to execute single instructions simultaneously in many cores by multiple threads. A program written in CUDA and executed on the GPU is known as a kernel and must be configured to exploit the available hardware, such as the number of threads.

The threads are arranged in a grid of 1D, 2D or 3D blocks which are scheduled to any of the available SMXs in order to be executed concurrently. The memory on the GPU plays a key role in performance. A global memory, a texture memory and a constant memory are available for all the threads whereas an on-chip shared memory is available only for threads of the same block. This on-chip memory has only 64 KB for load/store data but a higher bandwidth than the global memory. Therefore, its use is highly recommended. There are mechanisms for synchronizing threads locally within a block for exploiting this specific hardware avoiding the communication hazards that arise in parallel programming.[25] However, the shared memory has block level scope, that is, only threads of the same block can share data through this memory. Owing to this restriction it is not possible to share data among blocks and it must be done through the global memory. Nowadays, the GPUs supported by the latest version of CUDA incorporate a L1 / L2 cache hierarchy for caching load and store requests from global memory.

The main aspects to be taken into consideration to make full use of the GPU computing capabilities can be grouped in four main rules: 1) minimizing data transfer between CPU and GPU, 2) accessing data in global memory must follow a pattern so that multiple threads load/store consecutive memory locations (coalescing accesses), 3) giving the GPU enough work to hide the memory latency, and 4) focusing on data reuse within the GPU.[25,26] These rules also apply to multi-GPU programming, specially the first one. Depending on the multi-GPU configuration, the communications between different devices could be through the CPU, thus becoming an additional challenge when a device needs data which has been generated in a different device.

In the algorithms described in the following sections, the kernels will be placed between `<>` symbols and threads within a block indicated as `tid`. The pseudocode will include the `GM`, `SM` and `CST` acronyms to indicate the use of the global, shared and constant memory spaces on GPU, respectively.

Algorithm 1 GPU 1D-DWT for Feature Extraction

```
Require: $X$ the hyperspectral image arranged as a matrix with one pixel-vector per row in global memory and $W$ the
    output for the approximation coefficients.
 1: kernel <1D-DWT-FR>($X$)
 2: wcoeff[tid] = $X$[tid]                                          ▷ GM → SM
 3: for $i \in \{0, ..., L\}$ do
 4:     if ( tid < N/2 ) then
 5:         $a = \sum_k h(k) * \text{wcoeff}[2*\text{tid} + k]$               ▷ SM,CST
 6:         // local synchronization among threads within the same block
 7:         wcoeff[ tid ] = a;                                       ▷ SM
 8:     end if
 9:     N = N >> 1;
10:     // local synchronization among threads within the same block
11: end for
12: $W$[tid] = wcoeff[tid]                                          ▷ SM → GM
13: end kernel
```

3.2 GPU 1D-DWT for Feature Extraction

The feature extraction by wavelets on GPU implements Eq. (1) in one kernel where successive spectral decompositions are performed in shared memory as illustrated in Fig. 2(a).

The pseudocode in Algorithm 1 summarizes the kernel. Threads are configured in one dimensional blocks according to the number of hyperspectral bands. In the kernel, the threads within the block (tid) load the hyperspectral image X from the global to the shared memory (line 2), where the convolution (lines 5 and 7) takes place. The filter h is allocated in the constant memory of the GPU. The circular padding for computing the convolution with the last elements of each row (line 5) is done with the data loaded at the beginning of the shared memory.

At each new level of decomposition (line 3) there is a local synchronization among the threads of the block and the number of working threads is halved (line 9). Finally, the approximation coefficients are copied back to global memory (line 12). The data are rearranged as a hyperspectral cube before the next step.

3.3 GPU Morphological Reconstruction for EMPs

Opening and closing are dual operators, therefore, in the following we only describe the opening by reconstruction implementation on GPU. The opening by reconstruction is defined as the reconstruction by dilation of an image I from a marker image J. The reconstruction is an iterative process[27] that can be implemented in a single-core computer as summarized in Algorithm 2.

Algorithm 2 Sequential Reconstruction (SR) algorithm[27]

```
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 simulates a data propagation in the forward (lines 3-5) and backward (lines 6-8) directions. This iterative process continues flattening the bright objects of the image (lines 2-9) until stability is reached, that is, until all objects are reconstructed. $N_G^+(p)$ and $N_G^-(p)$ are the backward (*left, left-up, up, right-up*) and forward (*right, left-down, down, right-down*) neighborhood of the pixel p.

A block-asynchronous reconstruction (BAR) algorithm on GPU for morphological transformations (opening and closing by reconstruction) was proposed in Ref. 28 for building extended morphological profiles. The BAR algorithm is based on a GPU block-asynchronous data propagation[29] that can be computed independently by blocks of threads. Unlike the sequential algorithm, with this approach multiple scans can be performed in both directions at the same time.

The BAR algorithm is summarized in Algorithm 3 where the blocks are configured with 32×4 threads. An iterative process (lines 2-5) launches a kernel for the morphological reconstruction on GPU until stability is reached. These iterations are called inter-block updates. In the `<BAR>` kernel, threads within a two dimensional block load data from the global to the shared memory (line 8). The data propagation takes place in a loop (lines 9-12) where the updating is performed completely in shared memory requiring only synchronization among the threads inside the block. These iterations are called intra-block updates and are asynchronous among different blocks. The forward and backward scans for the morphological reconstruction are joined in one step (line 10) where $N_G(\texttt{tid}) = \{N_G^+(\texttt{tid}) \cup N_G^-(\texttt{tid})\}$ represents the complete neighborhood of a pixel p that is processed by the thread `tid`.

Algorithm 3 GPU Block-Asynchronous Reconstruction (BAR) algorithm[28]

```
Require: $J$ the marker image and $I$ the mask image in global memory.
 1: procedure MMReconstruction($J, I$)
 2:     repeat                                                        ▷ inter-block updating
 3:         <BAR>($J, I$)
 4:         // global synchronization among blocks                    ▷ Synchronous among blocks
 5:     until stability
 6: end procedure

 7: kernel <BAR>($J, I$)                                               ▷ intra-block updating
 8: smem[tid] = $J$[tid]                                               ▷ GM → SM
 9: repeat
10:     smem[tid] = min(max{smem[q], q $\in N_G$(tid) $\cup$ {tid}}, $I$[tid])   ▷ SM
11:     // local synchronization among threads within the same block  ▷ Asynchronous among blocks
12: until stability
13: $J$[tid] = smem[tid]                                               ▷ SM → GM
14: end kernel
```

Finally, the result is copied back to global memory (line 13). The inter-block synchronization among blocks (line 4) ensures an up-to-date data communication before launching the kernel again until the morphological reconstruction is completed.

The main advantage of Algorithm 3 (consisting of intra and inter-block updates) is data reuse on the shared memory by computing as many operations as possible within the block. Therefore, multiple forward and backward scans can be performed at the same time reducing data transfers between CPU and GPU.

3.4 2D-DWT Multi-component Denoising on GPU

The wavelet-based multi-component denoising is based on the 2D wavelet transform and soft-thresholding. The Multi-component denoising approach, as shown in Algorithm 4, creates three denoised images (line 3) for each component of the EMP (line 2). The pseudocode in Algorithm 4 summarizes the multi-component denoising steps: 1) apply a forward 2D discrete wavelet transform (lines 4 and 5) to each component C_i of the EMP at different levels j; 2) apply soft-thresholding (line 6) to the detail wavelet coefficients using a universal threshold (λ); and 3) apply an inverse 2D discrete wavelet transform (lines 7 and 8) to reconstruct each level of decomposition separately ($C_{i,j}$).

The 2D-DWT implementation is based on separable filters, as described in Sec. 2.3, applying the one dimensional wavelet analysis separately to each dimension. This approach has been widely investigated for parallel

Algorithm 4 GPU Multi-Component Denoising (MCD) algorithm

```
Require: $EMP$ the extended morphological profile in global memory.
 1: procedure MULTI-COMPONENT-DENOISING($EMP$)
 2:     for each $C_i \in$ EMP do
 3:         for $j \in \{1, \log_2(w)/2, \log_2(w)-1\}$ do
 4:             <FORWARD-DWT-ROWS>($C_i, j$)                ▷ SM,CST
 5:             <FORWARD-DWT-COLS>($C_i, j$)                ▷ GM,SM,CST
 6:             <SOFT-THRESHOLD>($C_i, \lambda$)                ▷ GM
 7:             <INVERSE-DWT-COLS>($C_i, j$)                ▷ SM,CST
 8:             <INVERSE-DWT-ROWS>($C_i, j$)                ▷ SM,CST
 9:         end for
10:     end for
11: end procedure
```

processing on GPU architectures.[30–32] The key aspect of those implementations is the efficient use of the shared memory, specially for the wavelet by columns. The global memory accesses are not coalesced if the circular convolution (see Eq. (1) in Sec. 2.3) for computing the wavelet transform is applied directly by columns. So, the straightforward implementation is inefficient in terms of memory accesses (load/store).

In our implementation, the threads are configured in two dimensional blocks of 32×4 and 32×8 threads for the wavelet by rows and by columns, respectively. Each thread loads two data items from global to shared memory.[31] The 2D blocks ensure coalescence in the access to the global memory. The computation takes place with the data in the shared memory that can be efficiently exploited by rows and by columns.

In the transformation by rows, all arithmetic operations are completely carried out in shared memory with the low-pass h and high-pass g filters previously allocated in constant memory. In the transformation by columns, the convolution with the last elements in the `column` direction is carried out directly with the data from the global memory.

After the forward 2D-DWT, the universal threshold (λ) is computed as indicated by Eq. (3) using *Thrust*,[33] the C++ template library for CUDA. The GPU-accelerated `sort()` algorithm available in this library is used for computing the median and calculating the noise standard deviation estimated by $\sigma = \frac{\text{median}(|\text{HH}_1|)}{0.6745}$. The soft-thresholding (line 6) is applied to each detail coefficient $d \in \{\text{HL} \cup \text{LH} \cup \text{HH}\}$. Each thread accesses one element in global memory.

After soft-thresholding, the inverse 2D-DWT (lines 7 and 8) is applied first by columns and then by rows. These two kernels are configured also using 2D blocks of threads. The procedure is similar to the forward transform explained above.

3.5 Multi-GPU implementation

The classification scheme proposed in this work is designed to be executed also on multi-GPU platforms. In hyperspectral processing, data can be partitioned in the spectral or in the spatial domain.[34] The spectral-partitioning divides the hyperspectral cube into chunks made up of contiguous spectral bands while the spatial-partitioning keeps pixel-vectors as a whole. Figure 5 shows the task distribution between two GPU devices in the multi-GPU implementation. The hyperspectral image X is divided in the spatial domain for the first stage. In the example of Fig. 5, half of the pixel-vectors are copied into the global memory of each GPU. After the `<1D-DWT-FR>` kernel (Algorithm 1), devices are synchronized and the features extracted by wavelets (W) are copied back to the CPU and arranged as a hyperspectral cube.

For the second stage, data are partitioned in the spectral domain. So, in the example illustrated in Fig. 5, half of the hyperspectral bands are moved to each GPU. The EMP is built independently by the `MMReconstruction` procedure (Algorithm 3) where the block-asynchronous reconstruction `<BAR>` kernel is used for the opening and closing by reconstruction operators.

The 2D-DWT multi-component denoising method is computed with the data generated at each GPU. Therefore, there is no data movement between the second and the third stage. The Multi-component-Denoising

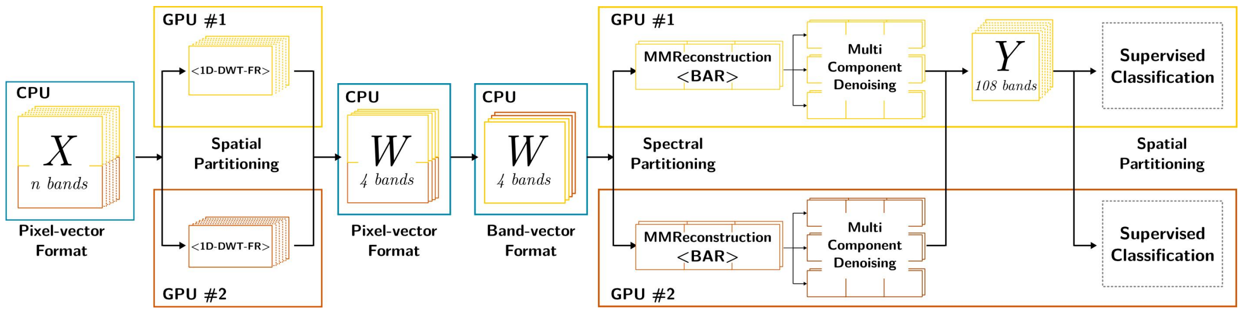


Figure 5. Multi-GPU implementation. Task distribution between two devices, named GPU #1 and GPU #2. Each kernel (placed between <> symbols) is exposed as an independent task.

procedure (Algorithm 4) is executed in each device where the kernels access the components C_i of the morphological profiles allocated in global memory. Finally, data are moved to the first device where the new stack of features Y, arranged as a hyperspectral cube, can be processed by a supervised classification method. The supervised classification can be also implemented in multiple GPUs by partitioning the data Y in the spatial domain.

The task decomposition proposed for this multi-GPU implementation is managed by OpenMP.[35] A thread on the CPU creates a team of threads and forks the execution of the task among the GPUs.

4. RESULTS

In order to evaluate the proposed approach in terms of classification accuracy and execution time we conducted several experiments. Regarding accuracy, we measured the Overall Accuracy (OA), Average Accuracy (AA) and Kappa coefficient of agreement[36] (κ) over three well-known hyperspectral images acquired from two different sensors: the Reflective Optics System Imaging Spectrometer[1] (ROSIS-03) and the Airborne Visible-infrared Imaging Spectrometer[37] (AVIRIS). The hyperspectral images and the configuration parameters of the approach are described in detail in Section 4.1.

Regarding the execution time, we measured the wall-clock time required for running the whole classification. The experiments were executed in the Finisterrae-II (FT2), a Linux based heterogeneous cluster at the Supercomputing Centre of Galicia (CESGA), Spain. For evaluating the execution time, we used a node (c7257) with an NVIDIA Tesla K80 device (Kepler architecture) with a dual-GPU design. The execution time is compared to a multi-threaded OpenMP implementation of the approach running on several cores in the same node.

4.1 Hyperspectral images and experimental setup

The hyperspectral images used in the experiments[*] are the urban areas of Pavia University and Pavia of City from the ROSIS-03 sensor, and the hyperspectral image of crop areas of Salinas Valley taken by the AVIRIS sensor. The number of available samples used for the supervised classification of these images are shown in Table 1. Figure 6 shows the reference maps of each hyperspectral image. The numbers of training and testing samples were chosen accordingly with the state of the art.[38,39]

These hyperspectral images have different spatial and spectral dimensions as well as different resolutions at pixel level. The dimensions of the Pavia University are 610×340 pixels with 103 spectral bands while the Pavia City dimensions are 1096×715 pixels with 102 spectral bands. These two images have a spatial resolution of 1.3 m/pixel and nine classes of interest available in their reference maps, as shown in Fig. 6(a) and Fig. 6(b), respectively. The Salinas Valley image has a spatial resolution of 3.7 m/pixel with 512×217 pixels and 204 spectral bands. The 20 bands covering the region of water absorption ([108-112], [154-167], 224) were discarded. The reference map for this image displays sixteen classes as shown in Figs. 6(c). Table 2 summarizes the spatial

[*]The hyperspectral images used in this work are available at http://www.ehu.es/ccwintco/index.php/HyperspectralRemoteSensingScenes.

Table 1. Number of training and testing samples for the hyperspectral images used in the experiments. The training and testing samples are disjoint sets.

Pavia Univ.			
#	Classes	Train	Test
1.	Asphalt	548	6083
2.	Meadows	540	18109
3.	Gravel	392	1707
4.	Trees	524	2540
5.	Metal	265	1080
6.	Bare Soil	532	4497
7.	Bitumen	375	955
8.	Bricks	514	3168
9.	Shadows	231	716

Pavia City			
#	Classes	Train	Test
1.	Asphalt	548	6083
2.	Meadows	540	18109
3.	Gravel	392	1707
4.	Trees	524	2540
5.	Metal	265	1080
6.	Bare Soil	532	4497
7.	Bitumen	375	955
8.	Bricks	514	3168
9.	Shadows	231	716

Salinas							
#	Classes	Train	Test	#	Classes	Train	Test
1.	Brocoli gr. weeds 1	40	1969	10.	Corn gr. weeds	65	3213
2.	Brocoli gr. weeds 1	74	3652	11.	Lettuce rom 4wk	21	1047
3.	Fallow	39	1937	12.	Lettuce rom 5wk	38	1889
4.	Fallow rough plow	27	1367	13.	Lettuce rom 6wk	18	898
5.	Fallow smooth	53	2625	14.	Lettuce rom 7wk	21	1049
6.	Stubble	79	3880	15.	Vinyard untrain	145	7123
7.	Celery	71	3508	16.	Vinyard ver trelli	36	1771
8.	Grapes untrain	225	11046				
9.	Soil vinyard dev.	124	6079				

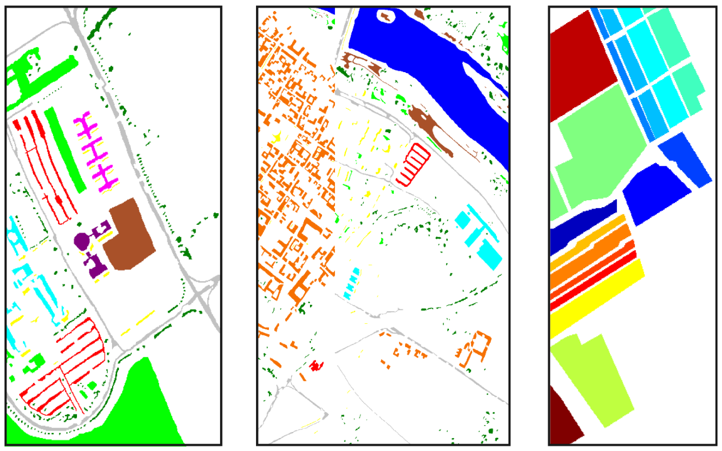

(a) (b) (c)

Figure 6. Reference maps of the hyperspectral images used for the supervised classification. Pavia University. (a), Pavia City (b) and Salinas Valley (c).

and spectral dimensions of the images and the size in megabytes (MB) required for allocating the data in memory using 8 bytes per pixel.

The results (accuracy and execution time) are calculated as the average of 20 executions. For measuring the accuracy we randomly choose at each execution a number of training and testing samples as indicated in Table 1. The number of features extracted by wavelets for each image is 4 and the number of denoising components is 3. The classification is carried out using the LIBSVM library[40] and the Extreme Learning Machine (ELM) algorithm.[41] The LIBSVM implements the SVM supervised classification method that is based on statistical learning techniques. The best parameters (C, γ) for training this classifier are determined for each hyperspectral image in the range $C = [1, 4, 16, 64, 128], \gamma = [0.5, 0.25, 0.125, 0.0625]$ by 5-fold cross-validation. The SVM uses the Gaussian radial basis function (RBF) as the activation function. The second classifier is an ELM. It is configured with a sigmoidal activation function and a number of neurons in the hidden layer in the range $N = [300, 800]$.

The computer platform used was the FT2 at CESGA. Each computing node is an Intel Xeon E5-2680 v3 processor at 2.50 GHz with 128 GB of RAM. The processor has a three-level cache hierarchy with 30 MB shared among all the cores at L3. The node has also an NVIDIA Tesla K80 device with a Kepler compute capability. The K80 has a dual-GPU design with 2496 cores per GPU at 0.82 GHz with a total of 24 GB of memory (12 GB $\times$ GPU). Each GPU incorporates a L1 / L2 cache hierarchy of 1.5 MB fully available for all the threads at L2. The L1 cache is available only for the threads running in the same streaming multiprocessor.

The CPU code has been compiled using the gcc 4.4.7 version with OpenMP 3.0 support and the GPU code using the nvcc (NVIDIA CUDA Compiler) version 8.0. In both cases full optimization flags (-O3) were used. The arithmetic operations have been executed in double precision (8 bytes) whereas the morphological operations use one byte per pixel as base data type. Two cores were used for managing the multi-GPU implementation and four cores were used for the OpenMP implementation. For measuring the execution times we excluded the times required for file I/O operations.

4.2 Accuracy assessment

In order to study the accuracy achieved by the classification based on the proposed multi-component denoising approach (W-MCD), we first evaluate the classification results produced using the SVM and the ELM classifiers. Table 3 summarizes the classification obtained by using each classifier, named W-MCD-SVM and W-MCD-ELM, in terms of Overall Accuracy (OA), Average Accuracy (AA) and Kappa (κ), including the class-specific accuracy (#) indicated by the class number.

It is worth noting that the SVM obtains better results than the ELM in all cases. In addition, lower standard deviation is observed by the SVM in the class-specific as well as in the overall accuracy. In general, the OA obtained by the pixel-wise SVM classification and used for the base comparison is over 90% in all the images, with a k above 80% indicating an almost perfect agreement.[36] The OA values obtained for the W-MCD-SVM approach are 98.47% and 99.56% for the two hyperspectral images of Pavia University and Pavia City and it is close to 95% for the Salinas Valley image. The improvement with this approach is remarkable, especially in the Pavia University image, with a result 5.7 percentual points better than with only the SVM classifier. Based on the better classification results obtained by the SVM, the analysis is focused from now on only on the W-MCD-SVM version of the approach.

4.3 Execution time assessment

This section first presents the results obtained for the OpenMP implementation. The best multi-threaded result will be used as the basis for evaluating the GPU implementation. Then, the results for the GPU and multi-GPU implementations of the approach are discussed.

The OpenMP implementation follows the same flow-chart illustrated in Fig. 5. Each stage of the approach creates a team of threads and forks the execution of the stage among the different threads. The hyperspectral

Table 2. Spatial (rows $\times$ columns) and spectral (bands) dimensions of the images used in the experiments and sizes in megabytes (MB) required for allocating the data in memory using 8 bytes per pixel.

	Pavia University	Pavia City	Salinas Valley
rows $\times$ columns $\times$ bands	$610 \times 340 \times 103$	$1096 \times 715 \times 102$	$512 \times 217 \times 204$
Size (MB)	162.9	609.8	189.8

Table 3. Classification results obtained by the W-MCD scheme using the SVM (W-MCD-SVM) and the ELM (W-MCD-ELM) classifiers, in terms of class-specific accuracy (#), OA, AA, and Kappa coefficient of agreement (κ), indicating the standard deviation ($\pm$). Best OA, AA and κ in bold.

	Pavia University				Pavia City			
#	SVM	ELM	W-MCD-SVM	W-MCD-ELM	SVM	ELM	W-MCD-SVM	W-MCD-ELM
1.	90.56 ±0.61	83.95 ±0.79	98.95 ±0.42	98.05 ±0.37	99.96 ±0.01	99.98 ±0.01	100.00 ±0.00	99.99 ±0.01
2.	93.64 ±0.39	89.90 ±0.60	97.95 ±0.27	95.90 ±0.84	96.43 ±0.20	95.90 ±0.41	95.95 ±0.25	97.67 ±0.34
3.	84.15 ±1.98	76.76 ±1.45	98.91 ±0.34	96.66 ±0.66	97.44 ±0.36	97.21 ±0.36	98.62 ±0.27	98.38 ±0.46
4.	96.95 ±0.80	96.71 ±0.35	99.16 ±0.35	98.66 ±0.30	96.78 ±0.47	92.52 ±1.05	99.82 ±0.09	99.84 ±0.06
5.	99.62 ±0.20	92.67 ±2.89	99.91 ±0.05	99.45 ±0.18	95.86 ±0.55	95.30 ±0.36	99.82 ±0.04	99.83 ±0.08
6.	93.59 ±0.67	91.47 ±0.88	98.17 ±0.46	94.68 ±0.45	98.33 ±0.15	99.01 ±0.09	99.21 ±0.21	99.64 ±0.06
7.	92.61 ±0.69	89.63 ±1.05	99.13 ±0.32	99.34 ±0.20	93.33 ±0.31	90.51 ±0.60	97.09 ±0.34	95.13 ±0.45
8.	87.99 ±1.09	89.17 ±0.74	99.17 ±0.26	98.83 ±0.23	99.33 ±0.08	98.48 ±0.18	99.90 ±0.03	99.17 ±0.20
9.	99.88 ±0.18	99.58 ±0.14	100.00 ±0.00	99.86 ±0.20	99.96 ±0.03	99.97 ±0.02	99.97 ±0.04	99.98 ±0.02
OA	92.74 ±0.28	89.21 ±0.35	**98.47 ±0.15**	96.84 ±0.37	98.95 ±0.03	98.51 ±0.05	**99.56 ±0.02**	99.34 ±0.08
AA	93.22 ±0.24	89.98 ±0.43	**99.04 ±0.08**	97.92 ±0.14	97.49 ±0.04	96.54 ±0.10	**98.93 ±0.05**	98.85 ±0.08
κ	90.19 ±0.01	85.53 ±0.01	**99.92 ±0.01**	95.70 ±0.01	98.48 ±0.00	97.84 ±0.00	**99.36 ±0.00**	99.05 ±0.01

	Salinas Valley			
#	SVM	ELM	W-MCD-SVM	W-MCD-ELM
1.	98.71 ±0.79	98.90 ±0.89	99.08 ±0.34	98.28 ±1.64
2.	99.45 ±0.21	99.70 ±0.09	99.65 ±0.16	99.47 ±0.20
3.	97.96 ±1.98	90.06 ±3.95	97.68 ±1.81	91.50 ±6.74
4.	99.24 ±0.46	98.32 ±0.40	99.24 ±0.70	98.88 ±0.42
5.	96.96 ±1.02	98.75 ±0.39	96.56 ±0.85	96.10 ±1.39
6.	99.53 ±0.34	99.74 ±0.19	99.82 ±0.11	99.64 ±0.23
7.	99.27 ±0.24	99.56 ±0.10	99.30 ±0.37	98.60 ±0.94
8.	85.11 ±2.72	88.26 ±1.46	88.68 ±1.67	87.63 ±1.52
9.	99.20 ±0.48	99.66 ±0.27	99.27 ±0.45	99.63 ±0.16
10.	91.95 ±1.91	93.71 ±2.43	96.74 ±1.80	94.83 ±2.10
11.	92.54 ±3.33	92.76 ±1.08	95.93 ±1.47	96.61 ±1.06
12.	99.88 ±0.12	100.00 ±0.00	99.93 ±0.03	99.78 ±0.18
13.	97.97 ±1.35	98.75 ±0.43	98.05 ±1.20	96.31 ±2.05
14.	93.47 ±3.45	91.94 ±2.10	92.11 ±4.51	87.77 ±3.41
15.	71.34 ±2.89	61.07 ±2.52	83.49 ±2.43	72.55 ±3.74
16.	97.46 ±1.80	97.36 ±1.58	98.61 ±0.39	95.45 ±1.63
OA	91.66 ±0.36	90.92 ±0.29	**94.44 ±0.30**	92.10 ±0.33
AA	95.01 ±0.33	94.28 ±0.34	**96.51 ±0.25**	94.61 ±0.44
κ	90.71 ±0.01	89.86 ±0.01	**93.81 ±0.01**	91.19 ±0.01

Table 4. Execution time (in seconds) and speedup (in parenthesis) for the OpenMP implementation using 1, 2 and 4 threads. The speedup is calculated with respect to the execution time with one thread (1 thread column).

	Pavia City			Salinas Valley		
	1 thread	2 threads	4 threads	1 thread	2 threads	4 threads
Feature Extraction by Wavelets	2.133	1.076 (1.9×)	0.546 (3.9×)	0.609	0.308 (1.9×)	0.157 (3.8×)
Extended Morphological Profile	7.244	3.683 (1.9×)	1.913 (3.7×)	0.938	0.495 (1.9×)	0.281 (3.3×)
Multi-Component Denoising	10.247	5.198 (1.9×)	2.763 (3.7×)	1.174	0.600 (1.9×)	0.319 (3.7×)
Data manipulation	0.616	0.420 (1.4×)	0.255 (2.4×)	0.087	0.054 (1.6×)	0.037 (2.3×)
W-MCD total execution time	20.240	10.377 (1.9×)	5.477 (3.7×)	2.808	1.457 (1.8×)	0.794 (3.5×)
SVM classification	95.931	60.949 (1.6×)	43.492 (2.2×)	5.604	3.749 (1.5×)	2.632 (2.1×)
W-MCD-SVM total execution time	116.171	71.326 (1.6×)	48.969 (2.3×)	8.412	5.206 (1.6×)	3.426 (2.4×)

image is partitioned in the spatial domain for the feature extraction by wavelets and partitioned in the spectral domain for the EMP and the multi-component denoising stages. The pixel-vector to band-vector transformation before building the EMP, is performed by only one thread. The reason is twofold: the reuse of the data stored in the cache is higher if the task is performed by a single thread, and the bands are distributed more easily from one thread.

Table 4 shows the execution times for each stage using 1, 2 and 4 threads for processing the Pavia City and Salinas Valley images, as well as the execution time for the SVM classification. The SVM execution time was obtained using the LIBSVM library compiled with OpenMP support.[40] The row named 'W-MCD total execution time' accumulates the times for the three stages of the scheme and the time spent in data manipulation. We included in the analysis the time required for data manipulation, such as arranging data from pixel-vector to band-vector, normalizing data for morphological operations, and casting data from 1 to 8 bytes for the arithmetic operations. The 'W-MCD-SVM total execution time' shows the time for creating the multi-component denoising approach and classifying it using SVM.

It is observed in Table 4 that the best execution time (the lowest) for creating W-MCD is achieved using four threads, with a speedup of 3.7× (Pavia City image) and 3.5× (Salinas image) as compared to the single-thread execution time.The speedup of the SVM reaches a value of 2.2× in the case of the Pavia City image.

Table 5 shows the execution time and the speedup on GPU using the K80 available in the computing node that consists of two GPUs. For the multi-GPU implementation, each stage of the approach creates one thread per GPU using OpenMP. Therefore, both devices are implicit synchronized after each stage owing to the join of the threads. Threads are only used for sharing work between devices. Only one thread is used for rearranging the features produced by wavelets from pixel-vector to band-vector, as illustrated in Fig. 5.

The speedup shown in Table 5 is calculated over the execution time with 1 GPU. In this table, the data manipulation also includes all copies of data, from the initial copy of the hyperspectral image from CPU to GPU, to the final copy of data in the multi-GPU case for joining the final result.

The execution time for the approach (W-MCD total execution time in Table 5) is reduced by using two GPUs by 1.5× for Pavia City and 2.1× for Salinas. Stage by stage, the speedup for the feature extraction by wavelets in the image of Pavia (first stage) is 2.0× with two devices. The EMP stage obtains also a good speedup (1.7×). We found that the speedup in the Multi-Component Denoising (third stage) is higher in the image of Salinas. By using two GPUs, the execution time is reduced 2.5× in this stage. The reason is found in the size of the image, that is one-third the size of Pavia City (see the MBs in Table 2), so more data are kept in the L1/L2 cache exploiting this memory space for load/store instructions. The data manipulation includes mainly the time required for the data transformation from pixel-vector to band-vector (15%) and for the data copy from host to device (40%).

Regarding the SVM classification time, it takes 18.668 seconds for Pavia and 2.663 seconds for Salinas using one GPU, corresponding to 95% and 88% of the W-MCD-SVM total execution time. This is an expected result because the SVM is the most computationally expensive stage. In this work, we have implemented the SVM classification stage in multiple GPUs by partitioning the data in the spatial domain, reaching a speedup of 1.9× for Pavia.

Finally, the multi-GPU implementation (2 GPUs) and the best multi-threaded implementation (4 threads) are compared in Table 6. The best speedup is obtained for the first stage. As described in Sect. 3.2, successive spectral decompositions are performed in shared memory to achieve 4 features extracted. Therefore, the on-chip memory is exploited repeatedly and the data transfer between CPU and GPU minimized.

The BAR algorithm used in the second stage for building the EMP reduces the execution time by 9.9× for the image of Pavia. The iterative reconstruction requires a communication between CPU and GPU at each

Table 5. Execution time (in seconds) and speedup for the multi-GPU implementation using 1 and 2 GPUs. The speedup is calculated with respect to the execution time with one GPU (1 GPU column).

	Pavia City			Salinas Valley		
	1 GPU	2 GPUs	Speedup	1 GPU	2 GPUs	Speedup
Feature Extraction by Wavelets	0.061	0.031	1.9×	0.016	0.009	1.7×
Extended Morphological Profile	0.335	0.193	1.7×	0.062	0.034	1.8×
Multi-Component Denoising	0.464	0.302	1.5×	0.246	0.097	2.5×
Data manipulation	0.096	0,097	0.9×	0.025	0.023	1.1×
W-MCD total execution time	0.957	0.623	1.5×	0.349	0.163	2.1×
SVM classification	18.668	9.826	1.9×	2.663	1.652	1.6×
W-MCD-SVM total execution time	19.625	9.883	1.9×	3.012	1.815	1.6×

Table 6. Execution time (in seconds) and speedup of the multi-GPU implementation using 2 devices as compared to the OpenMP multi-threaded implementation in CPU using 4 threads.

	Pavia City			Salinas Valley		
	4 threads	2 GPUs	Speedup	4 threads	2 GPUs	Speedup
Feature Extraction by Wavelets	0.546	0.031	17.6×	0.157	0.009	17.4×
Extended Morphological Profile	1.913	0.193	9.9×	0.281	0.034	8.2×
Multi-Component Denoising	2.763	0.302	9.1×	0.319	0.097	3.3×
Data manipulation	0.255	0.097	2.6×	0.037	0.023	1.6×
W-MCD total execution time	5.477	0.623	8.8×	0.794	0.163	4.8×
SVM classification	43.492	9.826	4.4×	2.632	1.652	1.6×
W-MCD-SVM total execution time	48.969	9.883	4.9×	3.426	1.815	1.9×

inter-block update (see Algorithm 3 in Sect. 3.3). However, the block-asynchronous data propagation[29] allows reusing the shared memory by computing as many operations as possible within the block. Thus, the data transfers between CPU and GPU are also minimized and data is reused within the GPU.

In general, the W-MCD total execution time for computing the three stages of the approach on two devices is 8.8× (Pavia City image) and 4.8× (Salinas Valley image) faster than using four threads on CPU. For the latter image, acquired by the AVIRIS sensor, the multi-GPU implementation achieves real-time performance. The cross-track line scan time of the sensor for collecting 512 pixel-vectors is 8.3 ms.[42] Therefore, the time for processing a cube of 512×217 pixels in real time must be less than 1.8 seconds.

5. CONCLUSIONS

This paper presentes a wavelet-based Multi-Component Denoising approach (W-MCD) for preprocesing hyperspectral images. It uses the features extracted by wavelets in the spectral domain for building an Extended Morphological Profile (EMP) which is spatially denoised by a recursively applied 2D-DWT. Thus, the W-MCD approach generates a small number of denoised components thus increasing the spatial information extracted by the EMP.

In this work the W-MCD proposal is evaluated with two supervised classifiers, SVM and ELM, classifying three hyperspectral images from the AVIRIS and the ROSIS sensors for remote sensing applications. The best results in terms of overall accuracy (OA) were obtained for the W-MCD-SVM approach, that is, using the SVM classifier. In particular, 98.47% accuracy was obtained for the Pavia University image and 94.44% for the Salinas Valley image. This work also presents a multi-threaded OpenMP and multi-GPU OpenMP-CUDA implementation of the W-MCD-SVM. Both implementations were executed in nodes at the Supercomputing Centre of Galicia (CESGA) using four threads and the NVIDIA Tesla K80 device with a dual-GPU design. The multi-GPU implementation processing the Pavia City image is 4.9× faster than the OpenMP implementation using 4 threads.

As future work we plan to perform a deeper study on the denoising capabilities of this proposal when it is applied over images obtained by other sensors with different spatial and spectral resolutions and under diferent atmospheric conditions.

ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Science and Innovation, Government of Spain, co-funded by the FEDER funds of the European Union, under contracts TIN2013-41129-P and TIN2015-63646-C5-1-R, and by Xunta de Galicia, Programme for Consolidation of Competitive Research Groups ref. 2014/008. Pablo acknowledges financial support from the Programa de Consolidación e Estructuración de Unidades de Investigación Competitivas da Xunta de Galicia (ref. R2014/041 e R2016/045), under a research contract. The authors would like to thank Prof. D. Manuel Graña Romay of the University of the Basque Country for freely providing on his research website a collection of the hyperspectral scenes.

REFERENCES

[1] Mueller, A. A., Hausold, A., and Strobl, P., "HySens-DAIS/ROSIS Imaging Spectrometers at DLR," *Proc. SPIE* **4545**, 225–235 (2002).

[2] Shaw, G. A. and Burke, H.-H. K., "Spectral imaging for remote sensing," *Lincoln Laboratory Journal* **14**(1), 3–28 (2003).

[3] Hughes, G., "On the mean accuracy of statistical pattern recognizers," *IEEE Transactions on Information Theory* **14**, 55–63 (January 1968).

[4] Dópido, I., Villa, A., Plaza, A., and Gamba, P., "A comparative assessment of several processing chains for hyperspectral image classification: What features to use?," in [*Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*], *3rd Workshop on*, 1–4 (2011).

[5] Bruce, L., Koger, C., and Jiang, L., "Dimensionality reduction of hyperspectral data using discrete wavelet transform feature extraction," *Geoscience and Remote Sensing, IEEE Transactions on* **40**(10), 2331–2338 (2002).

[6] Kaewpijit, S., Moigne, J. L., and El-Ghazawi, T., "Automatic reduction of hyperspectral imagery using wavelet spectral analysis," *Geoscience and Remote Sensing, IEEE Transactions on* **41**(4), 863–871 (2003).

[7] Quesada-Barriuso, P., Argüello, F., and Heras, D. B., "Spectral-spatial classification of hyperspectral images using wavelets and extended morphological profiles," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **7**(4), 1177–1185 (2014).

[8] Schowengerdt, R. A., [*Remote Sensing, Third Edition: Models and Methods for Image Processing*], Academic Press, Inc., Orlando, FL, USA (2006).

[9] Vidal, M. and Amigo, J. M., "Pre-processing of hyperspectral images. essential steps before image analysis," *Chemometrics and Intelligent Laboratory Systems* **117**, 119–148 (2012).

[10] Demir, B. and Ertürk, S., "Improved hyperspectral image classification with noise reduction pre-process," in [*Signal Processing Conference, 2008 16th European*], 1–4 (Aug 2008).

[11] Rasti, B., Sveinsson, J. R., Ulfarsson, M. O., and Benediktsson, J. A., "Hyperspectral image denoising using first order spectral roughness penalty in wavelet domain," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* , 1–10 (2013).

[12] Atkinson, I., Kamalabadi, F., and Jones, D. L., "Wavelet-based hyperspectral image estimation," in [*Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International*], **2**, 743–745 (July 2003).

[13] Quesada-Barriuso, P., Heras, D. B., and Argüello, F., "Exploring the impact of wavelet-based denoising in the classification of remote sensing hyperspectral images," in [*Proc. SPIE Remote Sensing*], **10004**, 100040R–100040R–16 (2016).

[14] Rasti, B., Sveinsson, J. R., Ulfarsson, M. O., and Benediktsson, J. A., "Hyperspectral image denoising using 3D wavelets," in [*2012 IEEE International Geoscience and Remote Sensing Symposium*], 1349–1352 (July 2012).

[15] Chen, G., Bui, T. D., and Krzyzak, A., "Denoising of three-dimensional data cube using bivariate wavelet shrinking," *International Journal of Pattern Recognition and Artificial Intelligence* **25**(03), 403–413 (2011).

[16] Christophe, E., Michel, J., and Inglada, J., "Remote sensing processing: From multicore to GPU," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4** (Sept 2011).

[17] Plaza, A., Du, Q., Chang, Y. L., and King, R. L., "High performance computing for hyperspectral remote sensing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **4**, 528–544 (Sept 2011).

[18] Daubechies, I., [*Ten lectures on wavelets*], vol. 61, Society for Industrial and Applied Mathematics (1992).

[19] Vetterli, M. and Herley, C., "Wavelets and filter banks: theory and design," *IEEE Transactions on Signal Processing* **40**(9), 2207–2232 (1992).

[20] Mallat, S., [*A wavelet tour of signal processing*], Academic press (1999).

[21] Serra, J., [*Image Analysis and Mathematical Morphology*], Academic Press, Inc., Orlando, FL, USA (1983).

[22] Soille, P., [*Morphological image analysis: principles and applications*], Springer-Verlag New York, Inc. (2003).

[23] Jansen, M., [*Noise reduction by wavelet thresholding*], vol. 161 of *Lecture notes in statistics*, Springer New York (2001).

[24] Donoho, D. L., "De-noising by soft-thresholding," *Information Theory, IEEE Transactions on* **41**(3), 613–627 (1995).

[25] Kirk, D. B. and Wen-mei, W. H., [*Programming Massively Parallel Processors: a Hands-on Approach*], Elsevier, 2nd ed. (2012).

[26] Farber, R., [*CUDA Application Design and Development*], Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st ed. (2012).

[27] Vincent, L., "Morphological grayscale reconstruction in image analysis: applications and efficient algorithms," *Image Processing, IEEE Transactions on* **2**(2), 176–201 (1993).

[28] Quesada-Barriuso, P., Argüello, F., Heras, D. B., and Benediktsson, J. A., "Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of* **PP**(99), 1–9 (2015).

[29] Quesada-Barriuso, P., Heras, D. B., and Argüello, F., "Efficient 2D and 3D watershed on graphics processing unit: block-asynchronous approaches based on cellular automata," *Computers & Electrical Engineering* **39**(8), 2638 – 2655 (2013).

[30] Galiano, V., López, O., Malumbres, M., and Migallón, H., "Improving the discrete wavelet transform computation from multicore to gpu-based algorithms," in [*Proceedings of the 11th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE)*], 544–555 (2011).

[31] Franco, J., Bernabé, G., Fernández, J., and Ujaldón, M., "The 2D wavelet transform on emerging architectures: GPUs and multicores," *Journal of Real-Time Image Processing* **7**(3), 145–152 (2012).

[32] Argüello, F., Heras, D. B., Bóo, M., and Lamas-Rodríguez, J., "The split-and-merge method in general purpose computation on gpus," *Parallel Computing* **38**(6–7), 277–288 (2012).

[33] Hoberock, J. and Bell, N., "Thrust, C++ template library for CUDA." https://developer.nvidia.com/thrust. (Accessed: 23 July 2017).

[34] Plaza, A. J., [*Parallel Spatial-Spectral Processing of Hyperspectral Images*], 163–192, Springer, Berlin, Heidelberg (2008).

[35] OpenMP Architecture Review Board, "OpenMP application program interface version 3.1," (2011).

[36] Viera, A. J. and Garrett, J. M., "Understanding interobserver agreement: the kappa statistic," *Fam Med* **37**(5), 360–363 (2005).

[37] Green, R. O., Eastwood, M. L., Sarture, C. M., Chrien, T. G., Aronsson, M., Chippendale, B. J., Faust, J. A., Pavri, B. E., Chovit, C. J., Solis, M., Olah, M. R., and Williams, O., "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment* **65**(3), 227 – 248 (1998).

[38] Fauvel, M., Tarabalka, Y., Benediktsson, J. A., Chanussot, J., and Tilton, J. C., "Advances in spectral-spatial classification of hyperspectral images," *Proceedings of the IEEE* **101**(3), 652–675 (2013).

[39] Marpu, P. R., Pedergnana, M., Mura, M. D., Peeters, S., Benediktsson, J. A., and Bruzzone, L., "Classification of hyperspectral data using extended attribute profiles based on supervised and unsupervised feature extraction techniques," *International Journal of Image and Data Fusion* **3**(3), 269–298 (2012).

[40] Chang, C.-C. and Lin, C.-J., "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology* **2**, 27:1–27:27 (2011).

[41] Huang, G.-B., "An insight into extreme learning machines: Random neurons, random features and kernels," *Cognitive Computation* **6**(3), 376–390 (2014).

[42] Plaza, A., Plaza, J., Paz, A., and Sánchez, S., "Parallel hyperspectral image and signal processing [applications corner]," *Signal Processing Magazine, IEEE* **28**, 119–126 (May 2011).