Efficient ELM-Based Techniques for the Classification of Hyperspectral Remote Sensing Images on Commodity GPUs

Javier López-Fandiño, Pablo Quesada-Barriuso, Dora B. Heras, and Francisco Argüello

***Abstract*—Extreme learning machine (ELM) is an efficient learning algorithm that has been recently applied to hyperspectral image classification. In this paper, the first implementation of the ELM algorithm fully developed for graphical processing unit (GPU) is presented. ELM can be expressed in terms of matrix operations so as to take advantage of the single instruction multiple data (SIMD) computing paradigm of the GPU architecture. Additionally, several techniques like the use of ensembles, a spatial regularization algorithm, and a spectral–spatial classification scheme are applied and projected to GPU in order to improve the accuracy results of the ELM classifier. In the last case, the spatial processing is based on the segmentation of the hyperspectral image through a watershed transform. The experiments are performed on remote sensing data for land cover applications achieving competitive accuracy results compared to analogous support vector machine (SVM) strategies with significantly lower execution times. The best accuracy results are obtained with the spectral–spatial scheme based on applying watershed and a spatially regularized ELM.**

***Index Terms*—Compute unified device architecture (CUDA), extreme learning machine, graphical processing unit (GPU), hyperspectral images, remote sensing, spectral–spatial classification, support vector machine (SVM), watershed.**

I. Efficient ELM-Based Classification of Hyperspectral Datasets

NOWADAYS, hyperspectral datasets are readily available thanks to the recent advances in sensor technology [1], [2]. These datasets provide information on hundreds of spectral bands at different wavelengths for each pixel, allowing to discriminate among different physical materials and objects. This work focuses on classification applied to land cover hyperspectral images and computed on commodity GPU platforms.

Supervised classification methods like support vector machine (SVM) [3] or extreme learning machine (ELM) [4] are suitable to deal with all the information available in these datasets. Additionally, ELM is a suitable algorithm to be implemented on commodity GPUs and other parallel architectures because the required operations are mostly matrix operations that can be computed in independent blocks, i.e., without data dependencies among them.

Different ELM-based techniques have been proposed: online sequential ELM, incremental ELM, pruned ELM [4], kernel ELM [5], or direct kernel perceptron (DKP) [6]. Another approach consists in the use of ensembles, combining the results obtained by different ELM classifiers [7], [8] or by the same classifier with different training datasets [9]. The well-known techniques Bagging and AdaBoost can also be applied to deal with ensembles of ELM [10]. ELM has been recently used in remote sensing [11]–[15] and slightly improves SVM results.

In general, the accuracy of the classification of hyperspectral datasets is improved when spatial information is considered [16]. The most widely used methods introducing spatial information consist in carrying out a segmentation of the image generated through unsupervised methods like partitional clustering [17] or watershed [18]. Spatial information can also be added through a spatial denoising based on wavelets and partial differential equations [19], or by a data fusion process performed via kernels methods [20], [21]. Other works consider mathematical morphology [22] and use the morphological information as an input to other processing stages [23]. Principal component analysis (PCA) is computed in [24] to build the morphological profiles. Independent component analysis (ICA) can also be used as an input to later processing stages. For instance, Dalla Mura *et al.* [25] use ICA as a preprocessing tool for a later spectral–spatial classification using extended morphological attribute profiles and SVM. Another approach consists in the use of multihypothesis prediction to achieve less intraclass variability and greater spatial regularity [26]. Furthermore, a spatial regularization based on the neighborhood can be added after the spectral classification [13] or at the end of the spatial–spectral processing [15].

Regarding computing platforms for a computationally efficient hyperspectral processing, commodity GPUs have been demonstrated to be appropriate. Recent works have explored how well-known algorithms like SVM [27], the maximum likelihood classifier (MLC) [28], or the automatic target detection and classification algorithm (ATDCA) [29] can be adapted to this kind of SIMD architectures. GPUs were also considered to optimize other hyperspectral image related problems

Manuscript received May 06, 2014; revised October 09, 2014; accepted December 05, 2014. Date of publication January 08, 2015; date of current version July 30, 2015. This work was supported in part by the Ministry of Science and Innovation, Government of Spain, in part by the FEDER funds of European Union, under contract TIN2013-41129-P, and in part by Xunta de Galicia, Programme for Consolidation of Competitive Research Groups ref. 2014/008. The work of J. López-Fandiño was supported by Xunta de Galicia, under a predoctoral grant. The work of P. Quesada-Barriuso was supported by the Ministry of Science and Innovation, Government of Spain, under a MICINN-FPI Grant.

The authors are with the Centro de Investigación en Tecnoloxías da Información (CITIUS), University of Santiago de Compostela, Santiago de Compostela 15782, Spain (e-mail: javier.lopez.fandino@usc.es; pablo.quesada@usc.es; dora.blanco@usc.es; francisco.arguello@usc.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTARS.2014.2384133

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

like parallel unmixing [30], target identification [31], or unsupervised band selection [32] achieving also good results. van Heeswijk *et al.* [33] develop a GPU implementation of some parts of a classification scheme based on ensembles of ELM; however, a complete and efficient GPU implementation of ELM to classify hyperspectral images has not been developed until now.

In this work, the first complete GPU implementation of the ELM algorithm is presented and applied to hyperspectral remote sensing images for land cover purposes. Additionally, several techniques to improve the results of the raw ELM classifier are implemented on GPU. We also adopt a spectral–spatial classification scheme for hyperspectral images based on the use of a pixelwise spectral classifier (ELM) and a spatial segmentation process by watershed applied to the result of a robust color morphogical gradient (RCMG). This gradient reduces the dataset to one band. Finally, a majority vote (MV) process is used to combine the spatial and spectral results obtaining the classification results [18].

The remaining sections of this paper are organized as follows: Sections II-A and II-B introduce the ELM algorithm as well as a version based on applying ensembles. Section II-C presents a spatially regularized version of the ELM, while Section II-D explains the spectral–spatial classification scheme. Section III details the GPU implementations of the algorithms. The experimental results are discussed in Section IV. Finally, the conclusion is summarized in Section V.

II. ELM-Based Spectral–Spatial Classification of Hyperspectral Images

This section introduces the ELM algorithm and different variants used to improve the accuracy results. Their GPU implementations will be studied in Section III.

A. ELM-Based Classification Algorithms

The raw pixelwise ELM algorithm was proposed as an efficient learning algorithm for single-hidden layer feedforward neural networks (SLFNs) [34]. Fig. 1 shows the structure of an SLFN. The output function of an SLFN with L hidden nodes, being $\boldsymbol{x}$ the input vector, can be written as

$$f_L(\boldsymbol{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\boldsymbol{a}_i, b_i, \boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d, \ \boldsymbol{\beta}_i \in \mathbb{R}^m \tag{1}$$

where $G(\boldsymbol{a}_i, b_i, \boldsymbol{x})$ denotes the output function of the ith hidden node, being $\boldsymbol{a}_i, b_i$ the hidden node parameters and $\boldsymbol{\beta}_i$ the weight vector connecting the ith hidden node to the output nodes. For the case of additive nodes with activation function g, it can be expressed as

$$G(\boldsymbol{a}_i, b_i, \boldsymbol{x}) = g(\boldsymbol{a}_i \cdot \boldsymbol{x} + b_i), \quad \boldsymbol{a}_i \in \mathbb{R}^d, \ b_i \in \mathbb{R}. \tag{2}$$

An SLFN with L hidden nodes can approximate N arbitrary distinct samples and targets $(\boldsymbol{x}_i, \boldsymbol{t}_i) \in \mathbb{R}^d \times \mathbb{R}^m$, if the following equation system can be solved:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{3}$$

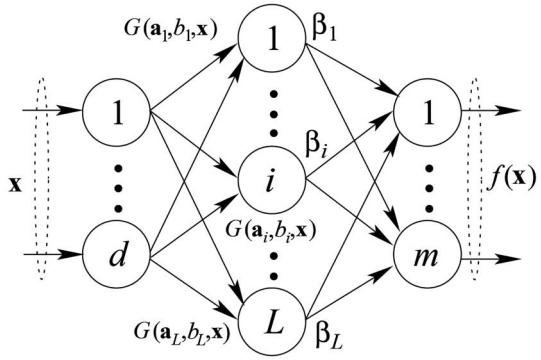

Fig. 1. SLFN as used by ELM.

ELM training algorithm.
Input: training set $\{(\boldsymbol{x}_i, \boldsymbol{t}_i) | \boldsymbol{x}_i \in \mathbb{R}^d, \boldsymbol{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$.
Output: SLFN parameters.
1: Randomly generate hidden node parameters $(\boldsymbol{a}_i, b_i)$, $i = 1, \ldots, L$ where $\boldsymbol{a}_i$ and b_i are the input weight and bias values and L is the hidden node number.
2: Calculate the hidden layer output matrix $\mathbf{H}$.
3: Calculate the output weight vector, $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$.

Fig. 2. ELM training algorithm.

where

$$\mathbf{H} = \begin{bmatrix} \boldsymbol{h}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{h}(\boldsymbol{x}_N) \end{bmatrix} = \begin{bmatrix} G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_1) & \ldots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_1) \\ \vdots & \ldots & \vdots \\ G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_N) & \ldots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_N) \end{bmatrix}_{N \times L} \tag{4}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^T \\ \vdots \\ \boldsymbol{\beta}_L^T \end{bmatrix}_{L \times N} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \boldsymbol{t}_1{}^T \\ \vdots \\ \boldsymbol{t}_N{}^T \end{bmatrix}_{N \times m}. \tag{5}$$

$\mathbf{H}$ is called the hidden layer output matrix of the neural network. Huang *et al.* [4] and [35] have proved that once they are randomly generated, the hidden node parameters $(\boldsymbol{a}_i, b_i)$ can remain fixed and training an SLFN is equivalent to finding a least-squares solution $\hat{\boldsymbol{\beta}}$ of the linear system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$, i.e.,

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T} \tag{6}$$

where $\mathbf{H}^{\dagger}$ is the *Moore–Penrose generalized inverse* of matrix $\mathbf{H}$ [36].

So, ELM can be summarized as shown in Fig. 2 [4], [35]. As it was stated in [5], ELM requires less human intervention than SVM because a single parameter, the number of neurons in the hidden layer, needs to be optimized, since all the other parameters are randomly initialized. In addition, ELM has better scalability and it runs at much faster learning speed than SVM.

B. Voting-Based ELM

An approach based on ensembles [7]–[10], [15], [33] computes a number of independent ELMs with the same number of hidden nodes and the same activation function in each hidden node and combines the results obtained by the different classifiers. The individual ELMs are randomly and independently

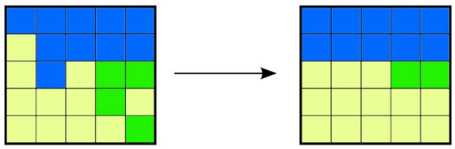

Fig. 3. Example of spatial regularization applied to a classification map.

(I) **Raw ELM pixel-wise classifier.**
Input: hyperspectral dataset.
Output: classification map.
(1) Preprocessing stage.
(2) Training stage.
(3) Test stage.

(II) **Gradient calculation.**
Input: hyperspectral dataset.
Output: single band image.
(1) Calculate the distances among pixels.
(2) Compute the gradient.

(III) **Segmentation.**
Input: single band image from gradient calculation.
Output: segmentation map.
(1) Initialization stage.
(2) Updating stage.

(IV) **Combination of maps.**
Input: ELM classification map, segmentation map.
Output: spectral-spatial classification map.
(1) Voting stage.
(2) Winner stage.
(3) Updating stage.

Fig. 4. Spectral–spatial classification scheme. (I) Spectral stage; (II–III) spatial stage; and (IV) combination stage.

initialized. The individual classification results are combined using a simple method called majority vote [37].

C. Spatially Regularized ELM

This postprocessing technique introduces spatial information of the closest neighborhood of a pixel to the classification map. It is specially helpful, as shown in Fig. 3, to remove disconnected points, which are commonly misclassified, and to smooth the edges among classes, solving the classification accuracy problem frequently present in these areas. Once the pixelwise classification map is generated, an iterative process starts where each pixel checks the class label of its neighbors and all the pixels do it simultaneously. If more than half of the neighbors share the same label, and this label is different from that of the pixel, the pixel updates his own class. This process lasts until stability is reached, i.e., until there are no changes between two consecutive iterations in any of the pixels [13].

D. ELM-Based Spectral–Spatial Classification Scheme

An efficient approach to integrate both the spectral and spatial information could follow the scheme presented in Fig. 4 and illustrated in Fig. 5. On the one hand, a classification map is produced through a pixelwise classifier. On the other hand, a segmentation map is created from a one-band image generated through a gradient calculation. Finally, spectral and spatial results are combined.

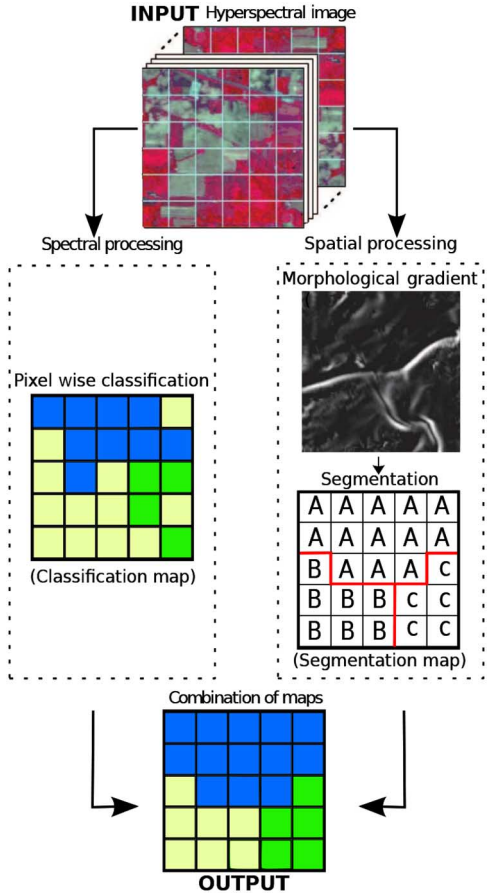

Fig. 5. Spectral–spatial classification scheme.

Different versions of this scheme using SVM [27], and ELM [13] as pixelwise classifiers have been proposed. In this work, the ELM-based scheme of [13] is projected to efficient execution on GPU. Additionally, spatial regularization is also applied to the classification map generated by the ELM before combining it with the segmentation map. The segmentation is calculated through an asynchronous watershed algorithm adequate for its implementation on GPU that we call cellular automaton watershed (CA-Watershed) [38].

At the end of the process, spectral and spatial results are combined through an MV process in which every pixel inside a watershed region is assigned to the most repeated class assigned by the pixelwise classifier within the pixels in the region.

III. Spectral–Spatial Hyperspectral Image Classification in GPU

In this section we introduce some compute unified device architecture (CUDA) programming fundamentals as well as the implementation in GPU of the algorithms proposed in Section II. The MAGMA library [39] is used to achieve optimal GPU linear algebra operations since this library has proven to be more efficient than others like CUBLAS [39], [40].

A. CUDA GPU Programming Fundamentals

CUDA is a combination hardware/software platform that enables NVIDIA GPUs to execute programs invoking parallel

functions called kernels that execute across many parallel threads [41]. These threads are organized into blocks so that each thread executes an instance of the kernel following a single instruction multiple data (SIMD) programming model. The blocks are arranged in a grid that is mapped to a hierarchy of CUDA cores in the GPU.

Threads can access data from multiple memory spaces. First, each thread has a private local memory and registers. Each block of threads has a shared memory visible, exclusively, to the threads within this block and whose lifetime equals the block lifetime. Finally, all threads access the same global memory space (DRAM) which is persistent across kernel launches by the same application. The lower the memory level, the faster the read/write access to the data. Shared memory lifetime makes it difficult to share data among thread blocks thus it implies the use of global memory whose access is slower than shared memory access.

The Kepler architecture [41] includes a two-level cache hierarchy. There are 64 KB of on-chip memory for each streaming multiprocessor (SMX), which can be configured as half each for the shared memory and the L1 cache, 48 KB of shared memory and 16 KB of L1 cache or vice versa. There is also a unified L2 cache of 1536 KB that is shared among all the SMX units.

Different performance optimization strategies have been applied in this work.

1) *Maximize parallel execution.* It is important to organize the algorithm in computational blocks that can be executed independently minimizing communications among them.
2) *Improve the efficiency in the use of the memory hierarchy.* It is essential to perform the maximum number of computations on data already stored in shared memory. This strategy is applied in the RCMG and CA-Watershed algorithms.
3) *Exploit the available optimized CUDA libraries.* The main stages of the ELM algorithm implementation take advantage of these tools.
4) *Add a border to the data regions.* Thus, in the gradient and segmentation stages, processing each pixel requires data from its neighbors, each data region is extended with a border of size one in order to minimize dependencies and contribute to point 1.

B. ELM-Based Classification in GPU

In this section, we describe the GPU implementation of the ELM algorithm introduced in Section II-A. The algorithm has three main phases: preprocessing, training, and test. The pseudocode in Fig. 6 shows the algorithm that has been implemented, including host and device codes. The kernels executed in GPU are placed between <> symbols. The pseudocode also includes the GM and SM acronyms to indicate kernels executed in global memory and shared memory, respectively.

First, all the data are scaled in the range $[0:1]$ (line 1 in the pseudocode). The pixels (pixel vectors) of each dataset are randomly distributed between two nonoverlapping sets: training and test. These two sets are stored in matrices $\mathbf{X}_{train}$ and $\mathbf{X}_{test}$, respectively, where each row represents a sample and each column a spectral band (lines 2 and 3 in the pseudocode). Data matrices are converted to column major format in order to be used by the MAGMA library (line 4 in the pseudocode). The ground truth labels are also split into two target matrices, $\mathbf{T}_{train}$, which is used during the learning phase, and $\mathbf{T}_{test}$, which will be used to check the accuracy results. Finally, the training and test target matrices are processed so that each row represents a sample and each column a class, where a value of 1 indicates membership to a class and a value of -1 is assigned otherwise (line 5 in the pseudocode). The preprocessing phase is computed in CPU and the results are stored in the global memory of the GPU. All the remaining steps will be computed in GPU.

GPU ELM.
Input: hyperspectral dataset $\mathbf{X}$*, label set* $\mathbf{T}$*, L: number of neurons in the hidden layer, C: number of classes, N: number of samples.*
Output: classification map.

1: Scale hyperspectral dataset in [0:1] ▷ Preprocessing phase (1)
2: Randomly choose the training points
3: Take the remaining points for test
4: Store data in column major order matrices $\mathbf{X}_{train}$ and $\mathbf{X}_{test}$
5: Process target matrices $\mathbf{T}_{train}$ and $\mathbf{T}_{test}$
▷ Training phase (2)
6: <Generate random weights ($\mathbf{a}_i$) and biases (b_i), i=1,...,L> ▷ GM
7: <Transpose the weight matrix and multiply by $\mathbf{X}_{train}$ > ▷ SM
8: <Add the biases> ▷ SM
9: <Apply activation (sigmoid) function to obtain $\mathbf{H}$ > ▷ GM
10: <Calculate $\mathbf{H}^{\dagger}$ as the Moore-Penrose pseudoinverse of $\mathbf{H}$ > ▷ SM
11: <Calculate output as $\boldsymbol{\beta} = \mathbf{H}^{\dagger} \times \mathbf{T}_{train}$ > ▷ SM
▷ Test phase (3)
12: <Transpose weight matrix and multiply by $\mathbf{X}_{test}$ > ▷ SM
13: <Add the biases> ▷ SM
14: <Apply activation (sigmoid) function to obtain $\mathbf{H}$ > ▷ GM
15: <Calculate output as $\mathbf{Y} = (\mathbf{H})^T \times \boldsymbol{\beta}$ > ▷ SM
16: <Calculate the estimated output label> ▷ GM
▷ GM: Global Memory, SM: Shared Memory

Fig. 6. Pseudocode for the ELM algorithm [stage (I) in Fig. 4].

The training phase starts by generating random weights and biases (line 6 in the pseudocode). The weights matrix must have values in the range $[-1:1]$, being its dimensions the number of neurons in the hidden layer and the number of input neurons (equal to the spectral band number). The bias vector will have values in the range $[0:1]$ and a size equal to the number of neurons in the hidden layer. These two matrices are then stored in the global memory before calculating the hidden layer output matrix $\mathbf{H}$. To do this, we first multiply the transpose of the weights matrix by the training matrix, then we add the bias vector to the result and, finally, we apply an activation function to each element of the matrix [lines 7–9 in the pseudocode corresponding to (2)]. In our case, a sigmoid function ($g(x) = 1/(1+e^{-x})$) is applied through a CUDA kernel with each thread operating over a single element of the matrix. The final training step consists in calculating the output weights multiplying the transpose of the pseudoinverse of matrix $\mathbf{H}$ by the training targets matrix [line 11 in the pseudocode corresponding to (6)]. The steps to efficiently calculate the pseudoinverse of a matrix will be explained below.

The test phase (lines 12–16 in the pseudocode) starts by multiplying the transpose of the previously generated weights matrix by the data matrix $\mathbf{X}_{test}$. Then, the bias vector is added and the activation function is applied just like in the training phase to obtain the test hidden layer matrix $\mathbf{H}$. These

operations are analogous to those of lines 7–9 in the training phase. Afterward, the output matrix $\mathbf{Y}$ is calculated multiplying $\mathbf{H}^T$ by the output weights matrix β obtained in the training phase. Finally, the estimated output label $\mathbf{T}_i$ is calculated as the class c that maximizes $\mathbf{Y}_i$ for each sample

$$\hat{\mathbf{T}}_i = \underset{c=1,\ldots,C}{\arg\max}\, \mathbf{Y}_{i,c}. \tag{7}$$

1) Efficient GPU Computation of the Moore–Penrose Inverse of a Matrix: The calculation of the pseudoinverse of a matrix is the most computationally costly operation in the training phase (line 11 in the pseudocode). We implement it as described in [36] to run it efficiently in CUDA using the MAGMA library.

We first check the dimensions of the input $\mathbf{H}$ matrix in order to know whether the number of rows is smaller than the number of columns. If this is the case, we use MAGMA to compute a matrix $\mathbf{A}$ as the multiplication of the original matrix by its transpose, otherwise we compute $\mathbf{A}$ as the multiplication of the transpose matrix by the original matrix. This operation ensures that $\mathbf{A}$ is a symmetric positive definite matrix.

The next step consists in calculating the Cholesky factorization of $\mathbf{A}$ with the MAGMA *dpotrf* function and then applying a kernel to nullify the upper triangle of the factorized matrix obtaining matrix $\mathbf{L}$. Unlike the other steps, this last kernel is launched in global memory.

Afterward, an $\mathbf{M}$ matrix is calculated multiplying $\mathbf{L}^T$ by $\mathbf{L}$ and then computing the inverse of this matrix using the MAGMA *dgetrf* and *dgetri* functions.

Finally, once all of these matrices have been calculated, we obtain the inverse of the original $\mathbf{H}$ matrix through a set of consecutive multiplications computed using the MAGMA *dgemm* function. If the dimension check of the $\mathbf{H}$ matrix at start resulted in that the row number is lower than the column number, we compute $\mathbf{H}^\dagger$ as

$$\mathbf{H}^\dagger = \mathbf{H}^T \times \mathbf{L} \times \mathbf{M} \times \mathbf{M} \times \mathbf{L}^T \tag{8}$$

otherwise we compute

$$\mathbf{H}^\dagger = \mathbf{L} \times \mathbf{M} \times \mathbf{M} \times \mathbf{L}^T \times \mathbf{H}^T. \tag{9}$$

C. Voting-Based ELM GPU Implementation

The voting algorithm used in this work comprises a set of independent ELMs whose outputs are stored in a matrix. After all the computations we will obtain an *N* by *C* matrix (being *N* the number of pixels and *C* the number of classes in the dataset) with the vote of each ELM for every pixel of the image. A similar use of ensembles in CPU is [8]. In the MV phase on GPU, the final label is calculated as the most repeated output value produced by the different ELMs for the sample. A CUDA kernel is launched and computed in global memory where each thread computes the MV for one pixel of the image.

D. GPU Projection of the Spatially Regularized ELM

This postprocessing technique compares the label of each pixel to that of its neighbors. It is computed by a single kernel that is executed as many times as it takes to reach stability.

GPU RCMG kernel.
Input: hyperspectral dataset $\boldsymbol{X}$.
Output: single band image.
▷ Distances step (1)
1: **for each** band i of $\mathbf{X}$ **do**
2: Load band i in shared memory
3: **for each** pixel x in band i **do**
4: Compute and accumulate the corresponding term in the euclidean distances $\mathrm{D}_{y,z} \mid y, z \in \mathcal{X}$, where $\mathcal{X}$ is the set of neighbours of pixel x
5: **end for**
6: Synchronize threads within the block
7: **end for**
▷ Gradient step (2)
8: Compute $\mathrm{CMG}(\mathbf{X}) = \max_{y,z \in \mathcal{X}} \mathrm{D}_{y,z}$
9: Find the pair of pixels $r = (y, z) \mid \mathrm{D}_{y,z} = \mathrm{CMG}(\mathbf{X})$
10: Compute $\mathrm{RCMG}(\mathbf{X}) = \max_{y,z \in \mathcal{X}-r} \mathrm{D}_{y,z}$
11: Write RCMG(**X**) to global memory

Fig. 7. Pseudocode for the RCMG kernel (shared memory) corresponding to stage (II) in Fig. 4.

The kernel is launched in bidimensional blocks of threads covering the entire classification map in each call. Each pixel of the image is processed by a thread.

E. Efficient GPU Processing of the Classification Scheme

In the spectral–spatial classification, a pipeline processing scheme is applied in GPU to combine the different stages of the algorithm reducing data movement through global memory.

1) RCMG GPU Implementation: The gradient calculation is divided into two steps whose pseudocode is introduced in Fig. 7 [42]. First, for all the pixel vectors, the threads within the same block cooperate to calculate the distances of the set χ. For each region, data are stored in row-major order for each band. The kernel is configured to work in two-dimensional (2-D) thread blocks. Threads within a block process a region of each spectral band in a sequentially processed loop through all the bands. In each iteration, data corresponding to a new band are loaded in shared memory (line 2 in the pseudocode of Fig. 7) and the partial results are computed and stored (loop in lines 3–5). When the outer loop is finished (lines 1–7), all the distances for each pixel are available in shared memory.

In the second step, each thread finds the maximum of the distances of its set χ (line 8) and the corresponding pair of pixels which generated the maximum (line 9). Once the two farthest pixel vectors are identified and removed, each thread computes the RCMG with the remaining distances (line 10) and writes the result in the global memory of the device (line 11).

2) GPU Implementation of the CA-Watershed: The segmentation map is calculated through the CA-Watershed algorithm [38]. The input data to this algorithm are directly the output obtained by the RCMG. The CA-Watershed can be asynchronously implemented, which is up to five times faster than the CUDA synchronous implementation [43]. The implementation includes intra-block asynchronous updates computed in shared memory and inter-block synchronous updates computed in global memory to efficiently deal with the fact that the cellular automaton (CA) algorithm needs data of the neighbors of each pixel. This implementation has the advantage of reusing information within a block to efficiently exploit the shared and cache memories of the device.

```
   GPU Asynchronous CA-Watershed.
   Input: single band image from RCMG.
   Output: segmentation map.                          ▷ Initialization stage (1)
1: <Initialize CA data (labels, distances and states)>  ▷ GM
                                                       ▷ Updating stage (2)
2: while CA is not stable do                           ▷ inter–block updating
3:     <Asynchronous updating of the CA>               ▷ intra–block updating (SM)
4:     Global synchronization among blocks
5: end while
                                   ▷ GM: Global Memory, SM: Shared Memory
```

Fig. 8. Pseudocode for the host and device asynchronous CA-Watershed scheme corresponding to the stage (III) in Fig. 4.

```
   GPU Asynchronous CA-Watershed updating kernel.
   Input: image CA data (labels, distances and states).
   Output: updated image CA data.                      ▷ intra–block updating
1: Load CA data in shared memory
2: while CA is not stable at block level do
3:     Update labels, distances and states within the block
4:     Local synchronization among threads
5: end while
6: Write CA data to global memory
```

Fig. 9. Pseudocode for the asynchronous updating kernel corresponding to the intra-block updating stage (line 3 in Fig. 8).

```
   GPU MV kernels.
   Input: ELM classification map, watershed segmentation map.
   Output: spectral-spatial classification map.
1: <Count number of watershed regions>                 ▷ GM
                                                       ▷ Voting stage (1)
2: <Count number of pixels of each class in each region>  ▷ GM
                                                       ▷ Winner stage (2)
3: <Obtain the winner class for each region>           ▷ GM
                                                       ▷ Updating stage (3)
4: <Update the pixels inside each region to the winner class>  ▷ GM
                                                       ▷ GM: Global Memory
```

Fig. 10. Pseudocode for the MV corresponding to stage (IV) in Fig. 4.

The algorithm used (a pseudocode of the algorithm is shown in Figs. 8 and 9), as further described in [38], comprises two kernels implementing the initialization and updating stages of the CA-Watershed which are configured to work in 2-D thread blocks operating in same-size pixel regions of the image. The updating stage is an iterative process that lasts until no modifications are made to the available data inside the region (lines 2–5 in Fig. 8). This implementation generates a segmentation map where the pixels are connected so that every pixel in the same region has the same label.

3) MV GPU Implementation: In the spectral–spatial scheme, the MV processes the pixels within each segmented region. In this implementation, a region can be assigned to different thread blocks as long as all the pixels belonging to the same region are connected.

A MV per watershed region is needed. As the number of regions is unknown beforehand, they are counted before the voting step (line 1 in Fig. 10). Afterward, a 2-D data structure is allocated in global memory whose dimensions are the number of watershed regions and the number of spectral classes.

The MV algorithm consists of three steps: *voting*, *winner,* and *updating* that are listed in Fig. 10. In the first step (line 2), for each watershed region the number of pixels for each class is counted. In the voting kernel, the voting is done by atomic operations where each thread adds one vote to the appropriate class in the region as more than one thread can vote in the same region to the same class with no predictable order. Then, the winner step finds the most repeated class inside each region (line 3). Finally, the updating kernel assigns all the pixels inside a region to the winning classes producing a new spectral–spatial classification map (line 4). Each step is performed by a separate kernel configured to work in one-dimensional (1-D) blocks of threads. In the first and third step kernels (lines 2 and 4), one thread operates on one pixel, while in the second one each thread operates on the information collected for one region of the segmentation map.

IV. Remote Sensing Classification Results

This section is intended to show the experimental results obtained by the ELM classifier in GPU as well as the different variations used to improve the accuracy results that have been previously presented. First, the experimental conditions will be described in Section IV-A. Then, the pixelwise ELM results will be detailed in Section IV-B. Finally, Section IV-C is devoted to the spectral–spatial scheme.

A. Hyperspectral Dataset and Experimental Conditions

The proposed algorithms have been evaluated on a PC with a quad-core Intel Core i5-3470 at 3.20 GHz and 8 GB of RAM. The code has been compiled using the gcc 4.6.3 version with OpenMP 3.0 support under Linux. Regarding the GPU implementation, CUDA codes run on an NVIDIA GeForce GTX Titan with 14 SMXs and 192 CUDA cores each. The CUDA code has been compiled using nvcc with version 5.5 of the toolkit under Linux.

The accuracy results are expressed in terms of overall accuracy (OA), average accuracy (AA), and kappa coefficient [44]. We provide the average and standard deviation of one hundred independent executions for all the previously introduced metrics. The performance results are expressed in terms of execution times and speedups compared to an OpenMP CPU optimized version of the algorithms parallelized using four threads and whose algebra operations are accelerated with the LAPACK library [45]. For comparison purposes, the accuracies of the ELM scheme are compared to the SVM results available in [17], [18], and [46]. Regarding performance, the speedups of the ELM implementations are also calculated comparing to an SVM implementation with the parameter values and number of training samples taken from [46] to [47]. This SVM implementation is an optimized version of [27] using OpenMP and CUDA for the CPU and GPU implementations, respectively, in the classification stage. In [27], the execution times were obtained by using publicly available libraries for both the training and the classification stages. The parameters chosen for ELM will be described in the next section.[1]

The tests were run on three hyperspectral airborne datasets [48]. A 103-band ROSIS image of the University of Pavia

[1] [Online]. Available: http://wiki.citius.usc.es/software/gpu-elm-rs

TABLE I
CLASSIFICATION ACCURACY AS PERCENTAGES (AND STANDARD DEVIATIONS BETWEEN BRACKETS)

	Pavia Univ.			Indian Pines			Salinas		
	OA	AA	kappa	OA	AA	kappa	OA	AA	kappa
SVM [18]	81.01	88.25	75.86	78.76	69.66	75.75	81.25	–	–
ELM	86.75	89.55	82.61	**80.72**	**85.48**	**77.70**	**91.55**	95.97	**90.55**
	(0.71)	(0.26)	(0.87)	**(0.58)**	**(1.31)**	**(0.64)**	**(0.27)**	(0.13)	**(0.29)**
V-ELM-1	**90.32**	**91.81**	**85.77**	79.84	90.62	72.40	90.74	**96.22**	89.18
	(0.31)	**(0.20)**	**(0.43)**	(0.83)	(2.82)	(1.08)	(0.11)	**(0.15)**	(0.13)

The ELMs contained 500, 950, and 350 nodes in the hidden layer, respectively.

(Pavia Univ.) with a spatial dimension of 610×340 pixels, a 220-band AVIRIS image of 145×145 pixels taken over Northwest Indiana (Indian Pines), and a 204-band AVIRIS image of 512×217 pixels taken over the Salinas Valley, California (Salinas).

B. ELM-Based Classification Results

We compare two different GPU optimized configurations using ELM.

1) A single ELM trained with 200 samples for each class (ELM).
2) A V-ELM comprising 8 ELMs trained with 200 samples for each class for each one of the ELMs, so that each ELM is the same as in the first configuration (V-ELM-1).

The number of training samples for the ELM is 200 per class, or half the number of samples in the class if there are not enough samples. These samples are randomly chosen and all the remaining samples are used for test. The number of hidden layer neurons employed is 500 for Pavia Univ., 950 for Indian Pines, and 350 for Salinas in all the cases [13].

Table I shows accuracy results for the images in terms of OA, AA, and kappa. The best results are highlighted in bold in the table. The first thing to highlight is that both configurations obtain acceptable accuracy results, being slightly better than the SVM for the three datasets.

For the Pavia Univ. image, the V-ELM-1 configuration clearly improves on the ELM configuration in terms of accuracy results while for the Indian Pines and Salinas images both configurations obtain similar results, being the ELM configuration only slightly better. Finally, it is worth noting than the standard deviation values remain low in all the cases. Fig. 11 shows the ground truths and false color classification maps obtained by the ELM algorithm.

The performance results in terms of execution times and speedups calculated over the OpenMP multicore implementations are detailed in Table II. It has been observed in the experiments that the V-ELM-1 configuration provides more stable accuracy results than a single ELM at the cost of slightly higher execution times.

The speedups of the ELM as compared to the SVM in both, the CPU and GPU architectures, are shown in Table III. For the three images, the single ELM configuration is faster than SVM, achieving, for the Pavia Univ. image, a speedup of 8.8× in CPU and 8.4× in GPU. The V-ELM-1 configuration is more adequate when the dataset size is large because otherwise (as in the case of Indian Pines) there are not enough samples to take advantage of the voting to improve accuracy results.

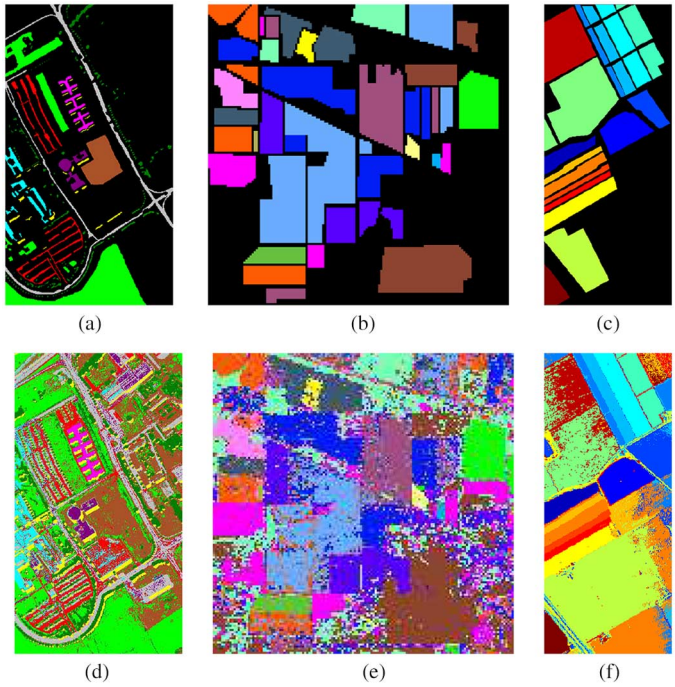

Fig. 11. (a–c) Ground truths and (d–f) classification maps for the (a and d) Pavia Univ.; (b and d) Indian Pines; and (c and f) Salinas images.

TABLE II
PERFORMANCE RESULTS

Pavia Univ.	SVM	ELM	V-ELM-1
OpenMP CPU	20.5876s	2.3304s	18.9022s
CUDA GPU	2.5834s	**0.3063s**	2.4501s
Speedup	8.0×	7.6×	7.7×
Indian Pines	**SVM**	**ELM**	**V-ELM-1**
OpenMP CPU	3.0084s	1.1653s	9.6903s
CUDA GPU	0.7652s	**0.3096s**	2.6058s
Speedup	3.9×	3.8×	3.7×
Salinas	**SVM**	**ELM**	**V-ELM-1**
OpenMP CPU	5.6018s	1.1023s	8.7055s
CUDA GPU	0.8708s	**0.3439s**	3.0114s
Speedup	6.4×	3.2×	2.9×

TABLE III
SPEEDUPS AGAINST SVM

	Pavia Univ.		Indian Pines		Salinas	
	CPU	GPU	CPU	GPU	CPU	GPU
ELM	**8.8**×	**8.4**×	**2.6**×	**2.5**×	**5.1**×	**2.5**×
V-ELM-1	1.1×	1.1×	0.3×	0.3×	0.6×	0.3×

Summarizing, on the one hand, for the remote sensing datasets considered the raw ELM algorithm described in this paper is significantly faster than SVM and, on the other hand, the V-ELM-1 algorithm always approaches or improves the raw ELM accuracy although it requires a higher number of training samples. This last one is a good configuration if we want to prioritize execution times.

C. Spectral–Spatial Classification Results

In this section, the experimental results obtained by the application of the spectral–spatial classification scheme introduced

TABLE IV
CLASSIFICATION ACCURACY AS PERCENTAGES (AND STANDARD DEVIATIONS BETWEEN BRACKETS)

	Pavia Univ.			Indian Pines			Salinas		
	OA	AA	kappa	OA	AA	kappa	OA	AA	kappa
SVM [18]	81.01	88.25	75.86	78.76	69.66	75.75	81.25	–	–
SVM+reg [17]	84.27	90.89	79.90	88.58	77.27	86.93	–	–	–
SVM+wat con(8) [18]	85.42	91.31	81.30	92.48	77.26	91.39	–	–	–
SVM+EM [17]	93.59	94.39	91.48	87.25	70.34	85.43	–	–	–
SVM+EM+reg [17]	94.68	95.21	92.02	88.83	71.90	87.24	–	–	–
V-ELM [15]	89.18	–	–	70.08	–	–	**93.88**	–	–
ELM	86.75(0.71)	89.55(0.26)	82.61(0.87)	80.72(0.58)	85.48(1.31)	77.70(0.64)	91.55(0.27)	95.97(0.13)	90.55(0.29)
ELM+reg	95.13(0.65)	95.51(0.40)	93.50(0.86)	91.04(0.82)	92.32(1.25)	89.54(0.94)	93.56(0.28)	**97.02(0.15)**	92.78(0.32)
ELM+wat con(4)	93.84(0.83)	94.05(0.47)	91.79(1.08)	88.73(0.67)	90.76(1.55)	86.90(0.76)	92.91(0.25)	96.15(0.18)	92.06(0.27)
ELM+wat con(8)	95.09(0.71)	95.14(0.47)	93.44(0.93)	91.41(0.97)	93.91(1.32)	89.98(1.12)	93.31(0.33)	96.52(0.17)	92.51(0.37)
ELM+reg+wat con(4)	95.37(0.67)	95.00(0.47)	93.81(0.88)	90.90(0.96)	91.47(1.63)	89.38(1.10)	93.46(0.31)	96.48(0.18)	92.67(0.35)
ELM+reg+wat con(8)	95.65(0.77)	95.52(0.52)	94.18(1.02)	**92.67(1.08)**	94.29(1.22)	**91.43(1.24)**	93.70(0.35)	96.78(0.16)	**92.95(0.39)**
V-ELM-1+reg+wat con(8)	**96.66(0.28)**	**95.92(0.29)**	**95.00(0.42)**	90.41(1.06)	**95.35(1.83)**	86.21(1.43)	92.43(0.31)	96.75(0.16)	91.15(0.36)

"reg" indicates that the pixelwise classifier was spatially regularized.
"wat" indicates spatial processing by watershed.
"con(x)" indicates connectivity of "x" neighbors.

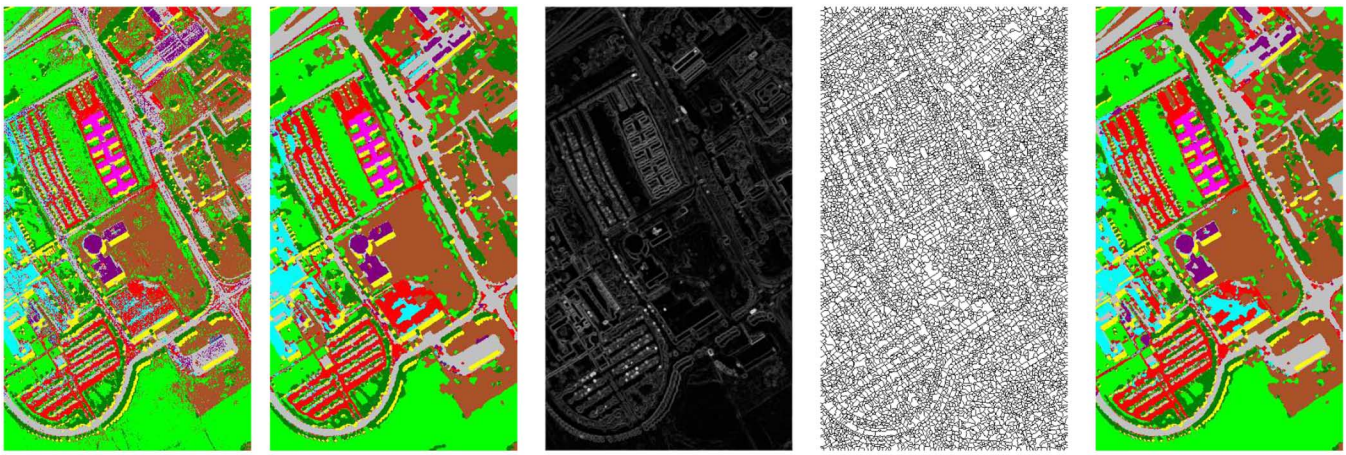

Fig. 12. Spectral–spatial phases for the Pavia Univ. image. (a) ELM; (b) Spatial regularization; (c) RCMG; (d) Watershed; and (e) MV.

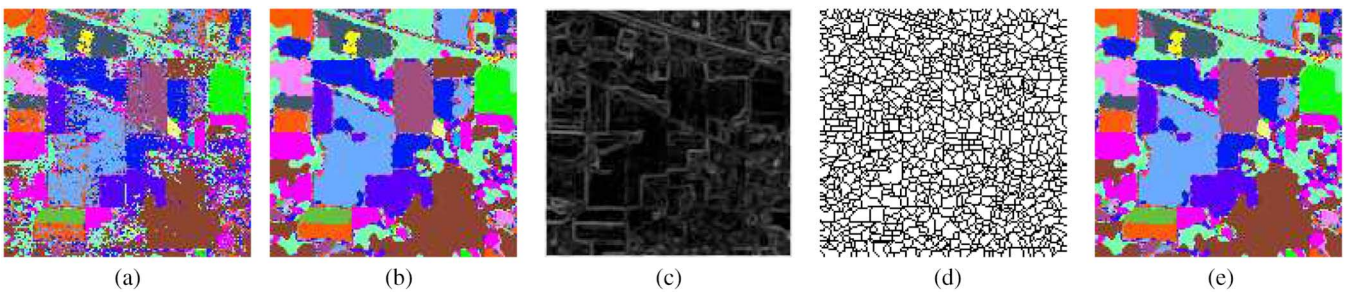

Fig. 13. Spectral–spatial phases for the Indian Pines image. (a) ELM; (b) Spatial regularization; (c) RCMG; (d) Watershed; and (e) MV.

in Fig. 5 are shown. The impact of spatial regularization over an ELM classification map is also studied.

The classification accuracy of the proposed method is compared to results published in the literature, as the pixelwise spectral classification by an SVM, spatial regularization (SVM+reg) [17], and the similar spectral–spatial schemes based on segmentation (SVM+wat) [18] and (SVM+EM) [17]. In addition, the combination of segmentation and spatial regularization (SVM+EM+reg) [17] is also included in the results. SVM+wat denotes that the segmentation map of the spectral–spatial scheme is created by watershed, and SVM+EM the same but using expectation maximization (EM) [49] for segmentation by partitional clustering. In all the schemes, the spectral–spatial information is combined by the majority vote algorithm within each segmented region. The spatial regularization of SVM+reg and SVM+EM+reg is done using Chamfer connectivities of eight and sixteen neighbors [17]. Results for another work based on ensembles of ELM and a similar spectral–spatial scheme (V-ELM) are also included [15].

TABLE V
PERFORMANCE RESULTS OF THE SPECTRAL–SPATIAL SCHEME BASED ON ELM (ELM+REG+WAT CON(8))

Pavia Univ.	ELM	Reg	RCMG	Watershed	MV	Total
OpenMP CPU	2.3164s	0.1674s	0.3740s	0.0232s	0.0010s	2.8820s
CUDA GPU	0.3066s	0.0070s	0.1710s	0.0020s	0.0004s	0.4870s
Speedup	7.6×	23.9×	2.2×	11.6×	2.5×	5.9×
Indian Pines	ELM	Reg	RCMG	Watershed	MV	Total
OpenMP CPU	1.1701s	0.0194s	0.0871s	0.0016s	0.0004s	1.2786s
CUDA GPU	0.3293s	0.0011s	0.0402s	0.0007s	0.0001s	0.3714s
Speedup	3.6×	17.6×	2.2×	2.3×	4.0×	3.4×
Salinas	ELM	Reg	RCMG	Watershed	MV	Total
OpenMP CPU	1.1017s	0.0984s	0.3957s	0.0159s	0.0006s	1.6123s
CUDA GPU	0.2110s	0.0039s	0.1829s	0.0017s	0.0001s	0.3996s
Speedup	5.2×	25.2×	2.2×	9.4×	6.0×	4.0×

Table IV shows the accuracy obtained using the developed classification scheme (best results for each dataset in bold). Results from the literature obtained using an SVM pixelwise classifier are also included for comparison purposes.

As it can be observed in Table IV, the ELM-based strategy obtains better results for the three datasets. Therefore, it can be stated that in accuracy terms ELM is suitable to replace SVM in this spectral–spatial scheme. The connectivity of eight neighbors, as expected, improves the results of the four neighbors one. Table IV also shows that the spatially regularized configurations always give better results. It is worth noting that the spectral–spatial scheme using a spatially regularized ELM [ELM+reg+wat con(8)] requires less computation time that the one based in ensembles of ELM [V-ELM-1+reg+wat con(8)] but achieves better results.

Figs. 12 and 13 show the results of the spectral–spatial scheme using a spatially regularized ELM [ELM+reg+wat con(8)] for the Pavia Univ. and Indian Pines datasets, respectively, while performance results are detailed in Table V. For all the datasets, the GPU implementation improves the CPU implementation achieving speedups up to 5.9×. It can also be noticed that the larger the number of pixel vectors in the dataset, the better the speedup. This is due to, specially, the ELM and watershed phases, that achieve better speedup when the dataset is larger in the spatial domain as it can be observed in Table V.

V. CONCLUSION

In this paper, the first complete CUDA GPU implementation of the ELM algorithm to efficiently classify remote sensing hyperspectral datasets is presented. Different ELM-based variations designed to improve the accuracy results are also explored. The GPU implementation takes advantage of the thousands of threads available, using shared memory to make an effective use of the memory hierarchy, and exploiting a linear algebra library in order to take advantage of the GPU architecture. First, we have explored the performance of a raw-ELM pixelwise classification. A ensemble configuration was also considered to achieve better classification accuracies as well as a spatially regularized version of the algorithm. Finally, we have studied the results of incorporating the ELM to a spectral–spatial classification scheme. A comparison to similar schemes based on SVM was also performed.

In accuracy terms, the spectral–spatial scheme using a spatially regularized ELM [ELM+reg+wat con(8)] has been confirmed as a good candidate for the classification of hyperspectral datasets applied to remote sensing. In our experiments, accuracy results up to ten percentage points better than a raw ELM were achieved. The best accuracy result was 96.66% for the Pavia Univ. image.

Results have shown that commodity GPUs are good candidates to reduce computation times in order to achieve real-time hyperspectral processing. For the ELM+reg+wat con(8) configuration, an execution time of 0.4870 seconds in GPU and a speedup of 5.9× compared to an OpenMP multicore CPU classification were achieved for the Pavia Univ. image.

REFERENCES

[1] F. D. van der Meer *et al.*, "Multi and hyperspectral geologic remote sensing: A review," *Int. J. Appl. Earth Observ. Geoinformat.*, vol. 14, no. 1, pp. 112–128, 2012.

[2] J. Bioucas-Dias *et al.*, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geosci. Remote Sens. Mag.*, vol. 1, no. 2, pp. 6–36, Jun. 2013.

[3] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 8, pp. 1778–1790, Aug. 2004.

[4] G.-B. Huang, D. H. Wang, and Y. Lan, "Extreme learning machines: A survey," *Int. J. Mach. Learn. Cybern.*, vol. 2, no. 2, pp. 107–122, 2011.

[5] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 42, no. 2, pp. 513–529, Apr. 2012.

[6] M. Fernández-Delgado, E. Cernadas, S. Barro, J. Ribeiro, and J. Neves, "Direct kernel perceptron (DKP): Ultra-fast kernel ELM-based classification with non-iterative closed-form weight calculation," *Neural Netw.*, vol. 50, pp. 60–71, 2014.

[7] N. Liu and H. Wang, "Ensemble based extreme learning machine," *IEEE Signal Process. Lett.*, vol. 17, no. 8, pp. 754–757, Aug. 2010.

[8] J. Cao, Z. Lin, G.-B. Huang, and N. Liu, "Voting based extreme learning machine," *Inf. Sci.*, vol. 185, no. 1, pp. 66–77, 2012.

[9] Y. Lan, Y. C. Soh, and G.-B. Huang, "Ensemble of online sequential extreme learning machine," *Neurocomputing*, vol. 72, no. 13, pp. 3391–3395, 2009.

[10] A. Samat, P. Du, S. Liu, J. Li, and L. Cheng, "E2LMs: Ensemble extreme learning machines for hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 4, pp. 1060–1069, Apr. 2014.

[11] M. Pal, "Extreme-learning-machine-based land cover classification," *Int. J. Remote Sens.*, vol. 30, no. 14, pp. 3835–3841, 2009.

[12] M. Pal, A. E. Maxwell, and T. A. Warner, "Kernel-based extreme learning machine for remote-sensing image classification," *Remote Sens. Lett.*, vol. 4, no. 9, pp. 853–862, 2013.

[13] D. B. Heras, F. Argüello, and P. Quesada-Barriuso, "Exploring ELM-based spatial–spectral classification of hyperspectral images," *Int. J. Remote Sens.*, vol. 35, no. 2, pp. 401–423, 2014.

[14] R. Moreno, F. Corona, A. Lendasse, M. Graña, and L. S. Galvão, "Extreme learning machines for soybean classification in remote sensing hyperspectral images," *Neurocomputing*, vol. 128, pp. 207–216, 2014.

[15] B. Ayerdi, I. Marqués, and M. Graña, "Spatially regularized semisupervised ensembles of extreme learning machines for hyperspectral image segmentation," *Neurocomputing*, 2014 [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2014.01.068i

[16] 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. 652–675, Mar. 2013.

[17] Y. Tarabalka, J. A. Benediktsson, and J. Chanussot, "Spectral–spatial classification of hyperspectral imagery based on partitional clustering techniques," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 8, pp. 2973–2987, Aug. 2009.

[18] Y. Tarabalka, J. Chanussot, and J. A. Benediktsson, "Segmentation and classification of hyperspectral images using watershed transformation," *Pattern Recognit.*, vol. 43, no. 7, pp. 2367–2379, 2010.

[19] S. Velasco-Forero and V. Manian, "Improving hyperspectral image classification using spatial preprocessing," *IEEE Geosci. Remote Sens. Lett.*, vol. 6, no. 2, pp. 297–301, Apr. 2009.
[20] G. Camps-Valls, L. Gomez-Chova, J. Muñoz-Marí, J. Vila-Francés, and J. Calpe-Maravilla, "Composite kernels for hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 3, no. 1, pp. 93–97, Jan. 2006.
[21] M. Fauvel, J. Chanussot, and J. A. Benediktsson, "A spatial–spectral kernel-based approach for the classification of remote-sensing images," *Pattern Recognit.*, vol. 45, no. 1, pp. 381–392, 2012.
[22] A. Plaza *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, pp. S110–S122, 2009.
[23] P. R. Marpu *et al.*, "Classification of hyperspectral data using extended attribute profiles based on supervised and unsupervised feature extraction techniques," *Int. J. Image Data Fusion*, vol. 3, no. 3, pp. 269–298, 2012.
[24] 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.
[25] M. Dalla Mura, A. Villa, J. A. Benediktsson, J. Chanussot, and L. Bruzzone, "Classification of hyperspectral images by using extended morphological attribute profiles and independent component analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 3, pp. 542–546, May 2011.
[26] C. Chen *et al.*, "Spectral–spatial preprocessing using multihypothesis prediction for noise-robust hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 4, pp. 1047–1059, Apr. 2014.
[27] 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, pp. 19–42.
[28] S. Rosario-Torres and M. Vélez-Reyes, "Speeding up the MATLAB hyperspectral image analysis toolbox using GPUs and the jacket toolbox," in *Proc. IEEE 1st Workshop Hyperspectral Image Signal Process. Evol. Remote Sens. (WHISPERS'09)*, 2009, pp. 1–4.
[29] S. Bernabe, 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.
[30] S. Sánchez, A. Paz, G. Martín, and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units," *Concurrency Comput. Pract. Exper.*, vol. 23, no. 13, pp. 1538–1557, 2011.
[31] D. B. Heras, F. Argüello, J. L. Gomez, J. Becerra, and R. J. Duro, "Towards real-time hyperspectral image processing, a GP-GPU implementation of target identification," in *Proc. IEEE 6th Int. Conf. Intell. Data Acquis. Adv. Comput. Syst. (IDAACS)*, 2011, vol. 1, pp. 316–321.
[32] H. Yang, Q. Du, and G. Chen, "Unsupervised hyperspectral band selection using graphics processing units," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 660–668, Sep. 2011.
[33] M. van Heeswijk, Y. Miche, E. Oja, and A. Lendasse, "GPU-accelerated and parallelized ELM ensembles for large-scale regression," *Neurocomputing*, vol. 74, no. 16, pp. 2430–2437, 2011.
[34] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
[35] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, Jul. 2006.
[36] P. Courrieu, "Fast computation of moore-penrose inverse matrices," arXiv preprint arXiv: 0804.4809, 2008.
[37] L. Lam and C. Y. Suen, "Application of majority voting to pattern recognition: An analysis of its behavior and performance," *IEEE Trans. Syst. Man. Cybern. A*, vol. 27, no. 5, pp. 553–568, Sep. 1997.
[38] P. Quesada-Barriuso, D. B. Heras, and F. Arguello, "Efficient GPU asynchronous implementation of a watershed algorithm based on cellular automata," in *Proc. IEEE 10th Int. Symp. Parallel Distrib. Process. Appl. (ISPA)*, 2012, pp. 79–86.
[39] S. Tomov, R. Nath, P. Du, and J. Dongarra, *MAGMA Users Guide*. ICL, UTK, 2011 [Online]. Available: http://icl.cs.utk.edu/magma/
[40] Nvidia, *CUBLAS Library User Guide*, 2013 [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf
[41] Nvidia, *NVIDIA Kepler GK110 Architecture Whitepaper*, 2012 [Online]. Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-kepler-GK110-Architecture-Whitepaper.pdf
[42] P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Efficient segmentation of hyperspectral images on commodity GPUs," in *Proc. Conf. Knowl.-Based Intell. Inf. Eng. Syst. (KES)*, 2012, vol. 243, pp. 2130–2139.
[43] P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient 2D and 3D watershed on graphics processing unit: Block-asynchronous approaches based on cellular automata," *Comput. Elect. Eng.*, vol. 39, no. 8, pp. 2638–2655, 2013.
[44] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis*, vol. 3. New York, NY, USA: Springer, 1999.
[45] E. Anderson *et al.*, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999, ISBN 0-89871-447-8.
[46] J. Plaza, A. J. Plaza, and C. Barra, "Multi-channel morphological profiles for classification of hyperspectral images using support vector machines," *Sensors*, vol. 9, no. 1, pp. 196–218, 2009.
[47] Y. Tarabalka, J. Chanussot, and J. A. Benediktsson, "Segmentation and classification of hyperspectral images using minimum spanning forest grown from automatically selected markers," *IEEE Trans. Syst. Man Cybern. B*, vol. 40, no. 5, pp. 1267–1279, Oct. 2010.
[48] C. I. G. from the Basque University (UPV/EHU). (2014). *Hyperspectral Remote Sensing Scenes* [Online]. Available: http://www.ehu.es
[49] A. P. Dempster *et al.*, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc.*, vol. 39, no. 1, pp. 1–38, 1977.

Javier López-Fandiño received the B.S. degree in computer science and the M.S. degree in information technologies from the University of Santiago de Compostela, Santiago de Compostela, Spain, in 2012 and 2014, respectively. He is currently pursuing the Ph.D. degree in computer science from the same university.

His research interests include image processing, parallel algorithms, and computer graphics.

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

He joined the Computer Architecture Group, University of Santiago de Compostela, Santiago de Compostela, Spain, as a Research Assistant. His research interests include image processing, parallel algorithms, and GPUs.

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

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

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

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