GPU Parallel Implementation of Spatially Adaptive Hyperspectral Image Classification

Zebin Wu, *Member, IEEE*, Linlin Shi, Jun Li, *Member, IEEE*, Qicong Wang, Le Sun, *Member, IEEE*, Zhihui Wei, Javier Plaza, *Senior Member, IEEE*, and Antonio Plaza, *Fellow, IEEE*

***Abstract*—Image classification is a very important tool for remotely sensed hyperspectral image processing. Techniques able to exploit the rich spectral information contained in the data, as well as its spatial-contextual information, have shown success in recent years. Due to the high dimensionality of hyperspectral data, spectral-spatial classification techniques are quite demanding from a computational viewpoint. In this paper, we present a computationally efficient parallel implementation for a spectral-spatial classification method based on spatially adaptive Markov random fields (MRFs). The method learns the spectral information from a sparse multinomial logistic regression classifier, and the spatial information is characterized by modeling the potential function associated with a weighted MRF as a spatially adaptive vector total variation function. The parallel implementation has been carried out using commodity graphics processing units (GPUs) and the NVIDIA's Compute Unified Device Architecture. It optimizes the work allocation and input/output transfers between the central processing unit and the GPU, taking full advantages of the computational power of GPUs as well as the high bandwidth and low latency of shared memory. As a result, the algorithm exploits the massively parallel nature of GPUs to achieve significant acceleration factors (higher than 70x) with regards to the serial and multicore versions of the same classifier on an NVIDIA Tesla K20C platform.**

***Index Terms*—Graphics processing units (GPUs), hyperspectral image, parallel, sparse multinomial logistic regression (SMLR), spatially adaptive Markov random fields (MRFs), spectral-spatial classification.**

I. Introduction

HYPERSPECTRAL image classification intends to assign the pixel vectors of a scene into a set of predefined classes. Different from panchromatic and multispectral remote sensing images, hyperspectral images contain hundreds of narrow spectral bands, spanning the visible to infrared spectrum, and exhibit a wealth of information in the spectral domain. In the literature, it has been shown that techniques able to exploit both the spectral and the spatial information contained in the scene represent successful approaches to hyperspectral image classification [1].

However, a remaining challenge is to take full advantage of the spectral and spatial information contained in the hyperspectral data. Since each pixel is given by a high-dimensional vector, supervised and semisupervised classification requires a sufficient number of training samples, which is difficult to satisfy in many remote sensing applications [2]. Recently, many (supervised and unsupervised) methods have been presented for hyperspectral image classification [1]–[3], among which those based on machine learning methods such as the support vector machine (SVM) [4] or the multinomial logistic regression (MLR) [5], [6] have been proven to be able to deal with limited training samples in a robust way.

Another important challenge in hyperspectral image classification is the extremely high dimensionality of hyperspectral data cubes. The dimensionality and volume of hyperspectral data is ever increasing, and high-speed classification is very important for applications with time-critical constraints, such as target detection for military purposes, monitoring of chemical contamination, and wildfire tracking. Fortunately, recent advances in high performance computing [7]–[15] have allowed for the exploitation of specialized hardware devices such as field-programmable gate arrays (FPGAs) [11], Beowulf clusters and distributed computers [12], multicore central processing units (CPUs) [13], and graphics processing units (GPUs) [13]–[17] in hyperspectral imaging applications. Specifically, it is possible to greatly accelerate the hyperspectral image processing on a GPU-based parallel computing platform, benefit from its capacity of performing compute-intensive, massively

Manuscript received May 31, 2017; revised August 15, 2017; accepted September 14, 2017. Date of publication October 11, 2017; date of current version April 11, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61772274, Grant 61471199, Grant 61701238, Grant 91538108, and Grant 11431015, in part by the Fundamental Research Funds for the Central Universities under Grant 30917015104, in part by the Jiangsu Province Six Top Talents project of China under Grant WLW-011, in part by the Jiangsu Provincial Natural Science Foundation of China under Grant BK20170858, and in part by the Research Fund of Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks under Grant WSNLBKF201507. *(Corresponding Author: Zebin Wu.)*

Z. Wu is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China, Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing 210003, China, and also with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Cáceres E-10003, Spain (e-mail: zebin.wu@gmail.com).

L. Shi, Q. Wang, L. Sun, and Z. Wei are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: 1353622861@qq.com; wqclandy@163.com; gswei@njust.edu.cn).

J. Li is with the Guangdong Provincial Key Laboratory of Urbanization and Geo-Simulation, and the Center of Integrated Geographic Information Analysis, School of Geography and Planning, Sun Yat-Sen University, Guangzhou 510275, China (e-mail: lijun48@mail.sysu.edu.cn).

J. Plaza and A. Plaza are with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Cáceres E-10003, Spain (e-mail: jplaza@unex.es; aplaza@unex.es).

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

Digital Object Identifier 10.1109/JSTARS.2017.2755639

1939-1404 © 2017 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.

parallel computations [9]. For example, a GPU implementation of a sparse MLR (SMLR) classifier has been recently presented as one of the first techniques achieving real-time classification performance [7].

Nevertheless, the SMLR is a pixel-based classifier that only exploits the spectral information contained in the scene. To further improve the classification accuracy and robustness [18], the spatial-contextual information has been successfully included in many classification techniques, such as composite kernels (CK) [19], [20], graph kernels [21], morphological filters [22], [23], partitional clustering [24], [25], and joint sparse representation [26], [27]. Among these techniques, Markov random fields (MRFs) [28]–[31] have achieved great popularity and effectiveness in the task of incorporating the spatial information to spectral information under a Bayesian inferring framework.

High classification accuracy is extremely important for proper decision-making in many critical scenarios [3], including, for instance, precision agriculture, urban planning, or military reconnaissance. Take into consideration that neighboring pixels in natural scenes usually comprise materials with similar spectral characteristics, especially in homogeneous regions, many techniques have included the spatial-contextual information in addition to spectral-based analysis in order to improve the classification accuracy and robustness [18]. This is, for instance, the case of techniques such as CK [19], [20], graph kernels [21], morphological filters [22], [23], partitional clustering [24], [25], and joint sparse representation [26], [27]. Among these techniques, MRFs [28]–[31] have achieved great popularity and effectiveness in the task of incorporating the spatial information to spectral information under a Bayesian inference framework.

In order to further exploit the spatial structure and contextual information to improve the classification accuracy in hyperspectral image classification, Sun *et al.* [32] proposed a novel spectral-spatial approach based on spatially adaptive MRFs. In this approach, the spectral information is learnt by an SMLR classifier, and the spatial information is characterized by modeling the potential function associated with a weighted MRF as a spatially adaptive vector total variation function, which is defined on the real-valued hidden marginal probabilities of the posterior distribution, where the weights are calculated by the gradients of the original hyperspectral image to model the spatial structure of the original data. This classifier has the potential to outperform other spectral-spatial approaches, as shown in [32]. Meanwhile, the utilization of spatial information leads to a significant computational burden, and its execution is computationally too expensive to achieve real-time performance, which compromises its application in time-critical scenarios.

In this paper, we develop an efficient parallel implementation of a spatial-spectral classification method based on spatially adaptive MRFs, implemented on commodity GPUs. The proposed parallel implementation has been developed using NVIDIA's Compute Unified Device Architecture (CUDA), as well as the cuFFT library. It optimizes the work allocation and input/output (I/O) transfers between the CPU and the GPU, taking full advantages of the computational power of GPUs as well as the high bandwidth and low latency of shared memory. Both classification accuracy and computational performance are evaluated, using two different GPU platforms by NVIDIA: Tesla C2075 and Tesla K20C. The experimental results, conducted on two real hyperspectral images, reveal remarkable acceleration factors while retaining exactly the same classification accuracy achieved by the corresponding serial and multicore versions.

The remainder of this paper is organized as follows. Section II briefly describes the spatially adaptive hyperspectral image classification method based on weighted MRFs. Section III describes its GPU parallelization in detail. Section IV evaluates the proposed parallel implementation in terms of both classification accuracy and computational performance. Section V concludes with some remarks and hints at plausible future research lines.

II. Hyperspectral Image Classification Based on Spatially Adaptive MRFs

Let us assume that $\boldsymbol{x} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N}] \in \boldsymbol{R}^{L\times N}$ is a hyperspectral image with N pixels and L bands (features), $\boldsymbol{S} \equiv \{1, 2, \ldots, N\}$ denotes the set of indexes of the N pixels, and $\boldsymbol{y} = [y_1, y_2, \ldots, y_N] \in \mathrm{K}^N$ is an image of class labels, where $\mathrm{K} \equiv \{1, 2, \ldots, K\}$ denotes a set of K class labels, $\boldsymbol{x}_i \in \boldsymbol{R}^L$ is an L-dimensional hyperspectral pixel observation, and each $y_i = [{y_i}^{(1)}, {y_i}^{(2)}, \ldots, {y_i}^{(K)}]$ denotes a "1-of-K" encoding of the K classes (${y_i}^{(j)} \in \{0, 1\}$, for$j \in \mathrm{K}$).

In a Bayesian framework, hyperspectral classification can be interpreted as estimating $\boldsymbol{y}$ given the observed image $\boldsymbol{x}$. The maximum a *posteriori* (MAP) classification [32], [33] can be modeled as

$$\begin{aligned}\hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}\in \mathrm{K}^N} (\log p(\boldsymbol{y}|\boldsymbol{x})) \\ &= \arg\max_{\boldsymbol{y}\in \mathrm{K}^N} \left\{ \sum_{i=1}^{N} (\log p(y_i|\boldsymbol{x}_i) - \log p(y_i)) + \log p(\boldsymbol{y}) \right\}. \end{aligned} \tag{1}$$

The densities $p(y_i|\boldsymbol{x}_i)$ are learnt by a probabilistic classifier, and the density $p(\boldsymbol{y})$ is usually modeled to impose a spatial prior on the labels $\boldsymbol{y}$. The classification task can be defined by the MAP marginal solution [34],

$$\hat{y}_i \equiv \max_k {q_i}^{(k)} = \max_k p(y_i = k|\boldsymbol{x}_j, j \in N_i), i \in \boldsymbol{S}. \tag{2}$$

In our previous work [32], we used SMLR [35] to model $p(y_i|\boldsymbol{x}_i)$ as a spectral data fidelity term, and modeled the spatial prior on the implicit marginal probability of the posterior distribution $\boldsymbol{q}$, instead of using a Gibbs distribution on the discrete labels $\boldsymbol{y}$.

Let us denote by $\boldsymbol{p}_C \equiv [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N] \in R^{K\times N}$ the probability matrix obtained through the SMLR, where $\boldsymbol{p}_i = [{p_i}^{(1)}, {p_i}^{(2)}, \ldots, {p_i}^{(K)}]^T$, and let us denote by $\boldsymbol{q} \equiv [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_N] \in R^{K\times N}$ the implicit marginal matrix, where $\boldsymbol{q}_i = [{q_i}^{(1)}, {q_i}^{(2)}, \ldots, {q_i}^{(K)}]^T$. Since the SMLR classifier does not utilize the spatial-contextual information, it often leads to classification outliers in homogeneous areas. Here we model the SMLR outliers as additive Gaussian white noise, and define the posterior probability $p(\boldsymbol{q}|\boldsymbol{p}_C)$ to infer $\boldsymbol{q}$ from $\boldsymbol{p}_C$ as follows:

$$\boldsymbol{q} = \arg\max_{\boldsymbol{q}} p(\boldsymbol{q}|\boldsymbol{p}_C) = \arg\max_{\boldsymbol{q}} p(\boldsymbol{p}_C|\boldsymbol{q}) p(\boldsymbol{q}) / p(\boldsymbol{p}_C) \tag{3}$$

whose solution is equivalent to

$$\boldsymbol{q} = \arg\max_{\boldsymbol{q}} p(\boldsymbol{q}|\boldsymbol{p}_C) = \arg\min_{\boldsymbol{q}}(-\log p(\boldsymbol{p}_C|\boldsymbol{q}) - \log p(\boldsymbol{q})). \tag{4}$$

Bearing in mind that modeling the spatial structure is important for classification purposes [36], a spatially adaptive MRF [32] is used here to model the spatial prior term, taking full advantage of the spatial-contextual information of the hyperspectral image as follows:

$$p(\boldsymbol{q}) = \frac{1}{Z}\exp\left\{-\mu_s \cdot a(\boldsymbol{x}_i)\sum_{|i-j|<\delta}\|\boldsymbol{q}_i - \boldsymbol{q}_j\|_1\right\} \tag{5}$$

where Z is a normalization constant for the density, μ_s tunes the degree of homogeneity of each region in the hyperspectral image. $|i-j|<\delta$ indicates that pixel i and j are a pair of neighbors in spatial sense, and $a(x_i)$ is a spatially adaptive regularization parameter imposing the edge structure information and adjusting the power of the spatial smoothness in different pixel locations as follows:

$$a(\boldsymbol{x}_i) = \frac{1}{1+\sqrt{\sum_{j=1}^{L}\left(\left(\nabla^h x_i^j\right)^2 + \left(\nabla^v x_i^j\right)^2\right)}} \tag{6}$$

where $\nabla^h x_i^j$ and $\nabla^v x_i^j$ are the horizontal and vertical first-order gradients of $\boldsymbol{x}_i$ at the jth band. Moreover, the true labels of the training samples used for the learning stage can be fixed as an additional constraint to characterize their spatial structure by spreading the class information to their neighbors. The probability distribution $\boldsymbol{q}$ must be nonnegative, and its columns must be sum-to-*1*. Then, the proposed hyperspectral image classification model based on spatially adaptive MRFs can be summarized as follows:

$$\hat{\boldsymbol{q}} = \arg\min_{\boldsymbol{q}}\left\{\|\boldsymbol{q}-\boldsymbol{p}_C\|_F^2 + \mu_s \cdot a(\boldsymbol{x}_i)\sum_{|i-j|<\delta}\|\boldsymbol{q}_i - \boldsymbol{q}_j\|_1\right\}$$
$$\text{s.t. } \boldsymbol{q} \geq 0, \boldsymbol{q}_{\Lambda_l} = y_{\Lambda_l}, 1^T\boldsymbol{q}_i = 1, i = 1,2,\ldots,N \tag{7}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, and $\boldsymbol{\Lambda}_l$ is the index set of training samples.

By defining $\boldsymbol{Hq} \equiv \begin{bmatrix}\boldsymbol{H}_h\boldsymbol{q}\\ \boldsymbol{H}_v\boldsymbol{q}\end{bmatrix} = \sum_{|i-j|<\delta}\|\boldsymbol{q}_i - \boldsymbol{q}_j\|_1$ and $\lambda = \mu_s \cdot a(\boldsymbol{x}_i)$, the model can be transformed into

$$\hat{\boldsymbol{q}} = \arg\min_{\boldsymbol{q}}\frac{1}{2}\|\boldsymbol{q}-\boldsymbol{p}_C\|_F^2 + \lambda\boldsymbol{Hq}$$
$$s.t.\ \ \boldsymbol{q} \geq 0, \boldsymbol{q}_{\Lambda_l} = y_{\Lambda_l}, 1^T\boldsymbol{q}_i = 1, i = 1,2,\ldots,N \tag{8}$$

where $\|\cdot\|_1$ denotes the l_1 norm, $\boldsymbol{q}_i$ is the i-th column of $\boldsymbol{q}$ corresponding to pixel i, δ controls the size of the neighborhood, and $\boldsymbol{H}$ is a convolution operator. $\boldsymbol{H}_h$ and $\boldsymbol{H}_v$ denote linear operators computing the horizontal and vertical differences, respectively, between the components of $\boldsymbol{q}$ corresponding to neighboring pixels [37]. Let us take $\boldsymbol{H}_h$ for example, $\boldsymbol{H}_h\boldsymbol{q} = [\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_N]$, where $\boldsymbol{d}_i = \boldsymbol{q}_i - \boldsymbol{q}_{i_h}$, i and i_h denote a pixel and its horizontal neighbor. By introducing variables $\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3, \boldsymbol{V}_4$, and $\boldsymbol{V}_5$, model (8) can be rewritten as

$$\hat{\boldsymbol{q}} = \arg\min_{\boldsymbol{V}_1,\boldsymbol{V}_2,\boldsymbol{V}_3,\boldsymbol{V}_4,\boldsymbol{V}_5}\left\{\frac{1}{2}\|\boldsymbol{V}_1 - \boldsymbol{p}_C\|_F^2 + \lambda\|\boldsymbol{V_5}\|_{1,1} + l_{R^+}(\boldsymbol{V_2}) + l_{\{1\}}\{\boldsymbol{V_3}\}\right\}$$
$$\text{s.t. } \boldsymbol{V}_1 = \boldsymbol{q}\ \ \boldsymbol{V}_2 = \boldsymbol{q}\ \ \boldsymbol{V}_3 = \boldsymbol{q}\ \ \boldsymbol{V}_4 = \boldsymbol{q}\ \ \boldsymbol{V}_5 = \boldsymbol{HV}_4 \tag{9}$$

where $\|\boldsymbol{x}\|_{1,1} = \sum_{i=1}^{N}\|\boldsymbol{x}_i\|_1$, $\boldsymbol{x}_i$ is the ith column of $\boldsymbol{x}$, and

$$l_S(x) = \begin{cases}0, x \in \mathrm{S}\\ +\infty, x \notin \mathrm{S}\end{cases}$$

.

At this point, the alternating direction method of multipliers [38], [39] can be utilized to efficiently solve model (9). In terms of the probability matrix $\boldsymbol{p}_C$, we use the logistic regression via variable splitting and augmented Lagrangian (LORSAL) algorithm [40] to solve the SMLR model. A detailed algorithmic description of our spatially adaptive hyperspectral image classification algorithm based on weighted MRF (referred to hereinafter as SAHIC) is given in Algorithm 1.

III. GPU Parallel Implementation Based on CUDA

The proposed parallel implementation has been developed following the philosophy that the most time-consuming operations are always carried out by GPU, and that it is particularly important to optimize the memory allocation and to minimize the I/O transfers between the CPU (host) and the GPU (device). First, we analyzed Algorithm 1 in detail to determine the most time-consuming parts that should be implemented in parallel. After analyzing Algorithm 1, the following observations are in place.

1) Algorithm 1 requires serial iterative loop executions for calculating the implicit marginal matrix $\boldsymbol{q}$. In every loop, $\boldsymbol{D}_1, \ldots, \boldsymbol{D}_5, \boldsymbol{V}_1, \ldots, \boldsymbol{V}_5$, and $\boldsymbol{q}$ need to be updated, each of which is a big matrix of size $K \times N$.
2) The algorithm involves many big matrix operations. Every read-write and arithmetic operation involving these big matrices is very time consuming on the CPU, and the multistep iterative process increases the computational burden even more. As a result, it is necessary to particularly optimize the operations relevant to these big matrices on the GPU.
3) The calculation of $\boldsymbol{V}_5$ is the most time-consuming step with computational complexity of $O(LN\log N)$, and the convolution operations in steps 2.7 and 2.8 are the most expensive parts (computationally) of the algorithm. Therefore, we mainly concentrate on the efficient parallelization of these operations in our GPU implementation.

With the aforementioned issues in mind, a GPU parallel implementation of SAHIC (SAHIC_P) has been designed as illustrated by the flowchart in Fig. 1.

Algorithm 1: Serial version of SAHIC (SAHIC_S).

Input: Training samples set $\boldsymbol{A} \in R^{L\times J}$, class labels of training samples $\boldsymbol{Y}_A \in R^{K\times J}$, test samples set $\boldsymbol{X} \in R^{L\times N}$

Initialization: Set $\lambda > 0$, $\lambda_C \geq 0$, $\beta > 0$, $\mu > 0$, $t = 0$, M = *MaxIteration*, $\tilde{\boldsymbol{A}} = h(\boldsymbol{A})$, $\tilde{\boldsymbol{X}} = \boldsymbol{h}(\boldsymbol{X})$, $\boldsymbol{h}(x)$ is the radial basis function (RBF), initialize $\boldsymbol{V}_1^{(0)}$, $\boldsymbol{V}_2^{(0)}$, $\boldsymbol{V}_3^{(0)}$, $\boldsymbol{V}_4^{(0)}$, $\boldsymbol{V}_5^{(0)}$, $\boldsymbol{D}_1^{(0)}$, $\boldsymbol{D}_2^{(0)}$, $\boldsymbol{D}_3^{(0)}$, $\boldsymbol{D}_4^{(0)}$, $\boldsymbol{D}_5^{(0)}$

Step 1. $\boldsymbol{p}_C = LORSAL(\tilde{\boldsymbol{A}}, \boldsymbol{Y}_A, \tilde{\boldsymbol{X}}, \lambda_C, \beta)$

Step 2. Calculate $\boldsymbol{q}$

Do:

Step 2.1.

$$\boldsymbol{q}^{(t+1)} = \frac{1}{4}(\boldsymbol{V}_1^{(t)} + \boldsymbol{D}_1^{(t)} + \boldsymbol{V}_2^{(t)} + \boldsymbol{D}_2^{(t)} + V_3^{(t)} + \boldsymbol{D}_3^{(t)} + \boldsymbol{V}_4^{(t)} + \boldsymbol{D}_4^{(t)})$$

$$\boldsymbol{q}_{\wedge_l}^{(t+1)} = \boldsymbol{Y}_{\wedge_l}$$

Step 2.2. $\boldsymbol{s} = \boldsymbol{q}^{(t+1)} - \boldsymbol{D}_3^{(t)}$

Step 2.3. $\boldsymbol{b}_{(j)} = \left(1 - \sum_{i=1}^{K} \boldsymbol{s}_{(i,j)}\right)/K, j \in (1, \ldots, N)$

Step 2.4. $\boldsymbol{V}_1^{(t+1)} = \frac{1}{1+\mu}(\boldsymbol{p}_C + \mu(\boldsymbol{q}^{(t+1)} - \boldsymbol{D}_1^{(t)})$

Step 2.5. $\boldsymbol{V}_2^{(t+1)} = \max(\boldsymbol{q}^{(t+1)} - \boldsymbol{D}_2^{(t)}, 0)$

Step 2.6. $\boldsymbol{V}_3^{(t+1)} = \boldsymbol{s} + \boldsymbol{1}\cdot\boldsymbol{b}, \boldsymbol{1} = [1, \ldots, 1]^T$, $\boldsymbol{a} = \boldsymbol{q}^{(t+1)} - \boldsymbol{D}_4^{(t)}$

Step 2.7. $\boldsymbol{V}_4^{(t+1)} = (\boldsymbol{H}^T\boldsymbol{H} + \boldsymbol{I})^{-1}$ $(\boldsymbol{a} + \boldsymbol{H}^T(\boldsymbol{V}_5^{(t)} + \boldsymbol{D}_5^{(t)})$

Step 2.8. $\boldsymbol{V}_5^{(t+1)} = \text{soft}(\boldsymbol{D}_5^{(t)} - \boldsymbol{H}\boldsymbol{V}_4^{(t+1)}, \lambda/\mu_s)$

Step 2.9. Update multipliers

$$\boldsymbol{D}_1^{(t+1)} = \boldsymbol{D}_1^{(t)} - (\boldsymbol{q}^{(t+1)} - \boldsymbol{V}_1^{(t+1)})$$

$$\boldsymbol{D}_2^{(t+1)} = \boldsymbol{D}_2^{(t)} - (\boldsymbol{q}^{(t+1)} - \boldsymbol{V}_2^{(t+1)})$$

$$\boldsymbol{D}_3^{(t+1)} = \boldsymbol{D}_3^{(t)} - (\boldsymbol{q}^{(t+1)} - \boldsymbol{V}_3^{(t+1)})$$

$$\boldsymbol{D}_4^{(t+1)} = \boldsymbol{D}_4^{(t)} - (\boldsymbol{q}^{(t+1)} - \boldsymbol{V}_4^{(t+1)})$$

$$\boldsymbol{D}_5^{(t+1)} = \boldsymbol{D}_5^{(t)} - (\boldsymbol{H}\boldsymbol{V}_4^{(t+1)} - \boldsymbol{V}_5^{(t+1)})$$

Step 2.10. $t = t + 1$

While $t > M$

Step 3. $[\boldsymbol{q}_{\max}, \boldsymbol{Y}_x] = \max(\boldsymbol{q})$, $\boldsymbol{q}_{\max} \in R^{1\times N}$ is a vector that consists of the maximum entry of each column in $\boldsymbol{q}$, $\boldsymbol{Y}_x \in R^{1\times N}$ is the corresponding row subscripts of the maximum entries.

Output: $\boldsymbol{Y}_X$, the class labels of training samples $\boldsymbol{X}$.

In the following, we describe the optimizations related to low level architecture and the most relevant parallelization steps conducted in the GPU for developing the computationally efficient parallel implementation of SAHIC in Algorithm 1.

A. Optimization of the Memory Allocation and I/O Transfer

First and foremost, we need to properly arrange the data storage and I/O communication between the host (CPU) and the device (GPU), to minimize the cost of data transfers. The training samples and test samples are stored by columns, and then manually transferred from the CPU to the local GPU memory after initialization. During the parallel optimization process, the data is stored in the GPU memory as much as possible, and the storage space for the intermediate variables of the iterative process is allocated in advance. By taking full advantage of the shared memory to achieve an efficient interaction with low latency, we maximize the memory bandwidth and optimizing accesses. In addition, the I/O communication between the host and the device mainly takes place when updating parameters or determining the termination conditions in the loop iterative process. After the iterative process is completed, the data will be transferred back from device to host, and the device memory will be set free when it is no longer needed.

B. Parallel Optimization for Calculating $\boldsymbol{V}_1, \boldsymbol{V}_2$, and $\boldsymbol{V}_3$

Once the data set is loaded into the GPU memory, we execute the GPU version of the variable splitting and augmented Lagrangian algorithm for sparse multinomial logistic regression (LORSAL_P) in [7] to obtain the probability matrix $\boldsymbol{p}_C$. After that, we define a CUDA kernel function called q_kernel, configured with a grid that consists of as many threads as the size of $\boldsymbol{q}$. Here every thread is responsible for calculating an element of $\boldsymbol{q}$. To fix the true labels of the training samples, a kernel fixq_kernel is realized to perform the execution of $\boldsymbol{q}_{\wedge_l}^{(t+1)} = \boldsymbol{Y}_{\wedge_l}$.

After that, we encapsulate the calculation of $\boldsymbol{s}$ and $\boldsymbol{b}$ into a single CUDA kernel function nu_aux3_sum_kernel (see Fig. 2), and optimize the calculation of $\boldsymbol{b}$ by means of a column batch sum. The kernel function nu_aux3_sum_kernel is designed to launch n blocks $(n = (N + ThreadNum - 1)/ThreadNum)$, and each block includes *ThreadNum* threads and a shared memory with the size of *ThreadNum* and data type of Double. Bearing in mind that the shared memory in the device has much lower latency, higher bandwidth, and smaller memory size than the global memory, we develop a strategy to perform the column batch sum to efficiently calculate the $\boldsymbol{b}$, as Fig. 3 shows, making full use of the shared memory.

Taking into consideration that the calculation of $\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3$ are loosely coupled and involve matrices of the same dimensionality, and that there are no data dependences among the matrix elements, steps 2.4, 2.5, and 2.6 in Algorithm 1 are merged and encapsulated into a single CUDA kernel function Vs_kernel to minimize the startup times of the kernel functions. This kernel launches as many threads as elements in $\boldsymbol{V}_1$, and each thread performs the calculations for one matrix element, thus making better use of the intrinsic concurrency of the CUDA blocks.

C. Parallel Optimization for Calculating $\boldsymbol{V}_4, \boldsymbol{V}_5$

The calculation of $\boldsymbol{V}_4$ and $\boldsymbol{V}_5$ is very time-consuming, and this represents a bottleneck of the algorithm. First of all, we compute $\boldsymbol{L} = (\boldsymbol{H}^T\boldsymbol{H} + \boldsymbol{I})^{-1}$ and store it outside the loop, since its

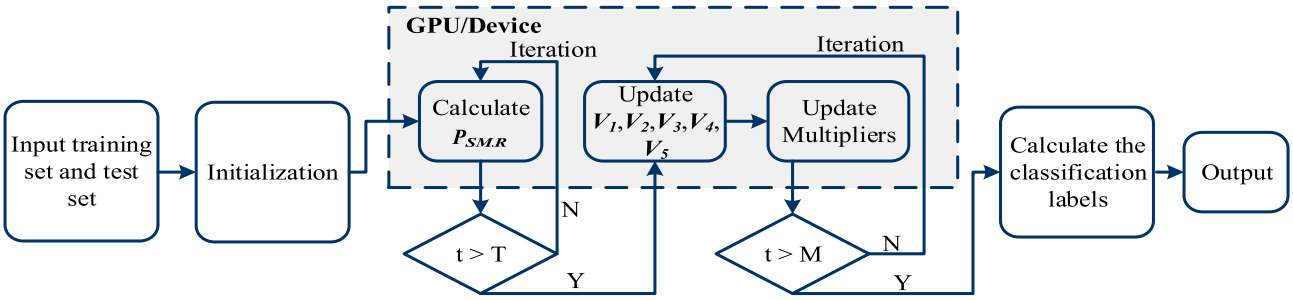


Fig. 1. Flowchart of the proposed GPU parallel implementation (SAHIC_P).

```
__global__ void nu_aux3_sum_kernel(double *U,double *Dmat2,
        double *Vmat2,double *nu_aux3, double *nu_aux3_sum,
        int rows, int cols, int n)
{
    __shared__ double block[THREAD_NUM];
    int idx = blockDim.x*blockIdx.x + threadIdx.x;
    int cacheIndex = threadIdx.x;
    block[cacheIndex] = 0;
    int tid = 0;
    while (idx<cols&&tid<rows)
    {
        int id = tid*cols + idx;
        double Utemp = U[id];
        nu_aux3[id] = Utemp - Dmat2[id];
        block[cacheIndex] += nu_aux3[id];
        tid++;
    }
    __syncthreads();
    if (idx<cols)
        nu_aux3_sum[idx] = block[cacheIndex];
    __syncthreads();
}
```

Fig. 2. CUDA codes for kernel nu_aux3_sum_kernel.

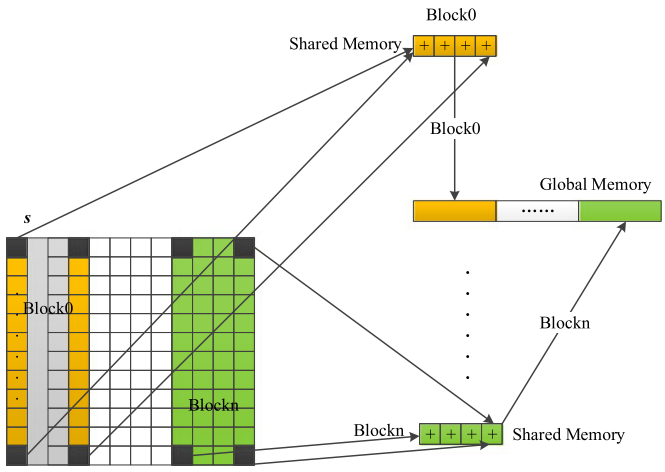


Fig. 3. Diagram that illustrates our efficient strategy to perform the column batch sum of a matrix.

computation remains unchanged in the iterative process. As a result, we define a kernel function named IL_kernel to compute $\boldsymbol{L}$, which launches a block with the same size of $\boldsymbol{H}$ on the GPU, and every thread is in charge of calculating one element of $\boldsymbol{L}$, taking full advantages of parallelism of the GPU device.

After that, we further analyze the steps 2.7 and 2.8. Since $\boldsymbol{H}$ acts only on the spatial domain, it can be independently handled in band-by-band fashion. For each band, we need to perform a convolution operation, which can be solved by means of a discrete Fourier transform (DFT) diagonalization. Therefore, it is crucial to efficiently implement the DFT on the GPU. In this paper, we accomplish this important step using the cuFFT library [41] in the CUDA Toolkit, which provides a simple interface for computing fast Fourier transforms (FFTs) on GPUs [42]. By employing the cuFFT library and applying the configuration mechanism plan, the transform is optimized for the particular GPU hardware. Then, the actual transform takes place following the plan of execution by calling the execution function. This is a key aspect to the performance of our GPU parallel implementation. It is worth noting that, by the plan interface, it is very easy to reuse the thread configurations and GPU resources for different kinds of FFTs. After the configuration of the plan, the function cufftExec$*$ is invoked to conduct the FFT calculation in the GPU. Since the convolution operation on each band is carried out independently, the FFT calculation is configured by means of a cufftPlanMany function, for which the size of the batch is set to the number of bands (i.e., K) in the original hyperspectral image, and every branch performs the two-dimensional Fourier transform with the same scale (i.e., the spatial size of the image), thus leveraging the parallelism of the GPUs and the floating-point power.

Then, a CUDA kernel called V4_kernel is implemented to compute in parallel the operations of matrix addition and subtraction (see Fig. 4). A kernel function called V5_kernel is launched next (see Fig. 5), where the number of threads equals the size of the matrix $\boldsymbol{V}_5$, and each thread implements the operation $\boldsymbol{V}_5^{(t+1)} = \mathrm{soft}(\boldsymbol{D}_5^{(t)} - \boldsymbol{H}\boldsymbol{V}_4^{(t+1)}, \lambda/\mu)$ and the updating of the multipliers for a matrix element. In this way, we can minimize the launch times of the kernel functions by merged refactoring.

The parallel algorithm now repeats from step 2.1 to 2.10 until a maximal number of iterations is reached. We implement the remaining operations in the CPU, since they have much lower computation costs, and can be realized very quickly in the CPU without the need for parallelization.

```
__global__ void V4_kernel(cuDoubleComplex* mid, double *Vmat3,
        int VmatCols, cuDoubleComplex* inner, int rows, int cols, int k)
{
    __shared__ double block[BLOCK_SIZE][BLOCK_SIZE + 1];
    int rowId = blockDim.y*blockIdx.y + threadIdx.y;
    int colId = blockDim.x*blockIdx.x + threadIdx.x;
    if (rowId < rows && colId < cols)
    {
        int idx = rowId*cols + colId;
        int size = rows*cols;
        inner[idx].x /= size;
        block[threadIdx.y][threadIdx.x] = inner[idx].x;
        int mididx = idx;
        cuDoubleComplex temp = inner[idx];
        mid[mididx] = temp;
        mid[mididx + size] = temp;
    }
    __syncthreads();
    rowId = blockDim.x*blockIdx.x + threadIdx.y;
    colId = blockDim.y*blockIdx.y + threadIdx.x;
    if (rowId<cols && colId<rows)
    {
        int ididx=k*VmatCols + rowId*rows + colId;
        Vmat3[ididx] =block[threadIdx.x][threadIdx.y];
    }
        __syncthreads();
}
```

Fig. 4. CUDA codes for kernel V4_kernel.

```
__global__ void V5_kernel(double *VcellMat1, double *VcellMat2,
        double *DcellMat1, double *DcellMat2, cuDoubleComplex* mid,
        double param, int rows, int cols)
{
    int rowId = blockDim.y*blockIdx.y + threadIdx.y;
    int colId = blockDim.x*blockIdx.x + threadIdx.x;
    if (rowId < rows && colId < cols)
    {
        int idx = rowId*cols + colId;
        int size = rows*cols;
        int mididx = idx;
        double auxh = mid[mididx].x / size;
        double auxv = mid[mididx + size].x / size;
        double Dcellmat1 = DcellMat1[idx];
        double Dcellmat2 = DcellMat2[idx];
        double mid1 = auxh - Dcellmat1;
        double mid2 = auxv - Dcellmat2;
        double temp = max(sqrt(mid1*mid1 + mid2*mid2) - param, 0.0);
        mid1 = temp / (temp + param);
        VcellMat1[idx] = mid1*(auxh - Dcellmat1);
        VcellMat2[idx] = mid1*(auxv - Dcellmat2);
        DcellMat1[idx] -= (auxh - VcellMat1[idx]);
        DcellMat2[idx] -= (auxv - VcellMat2[idx]);
    }
}
```

Fig. 5. CUDA codes for kernel V5_kernel.

Fig. 6 gives an overall illustration of our GPU parallel implementation of the spatially adaptive hyperspectral image classification algorithm based on weighted MRFs (SAHIC_P). And a detailed step-by-step algorithm description of the SAHIC_P is summarized in Algorithm 2.

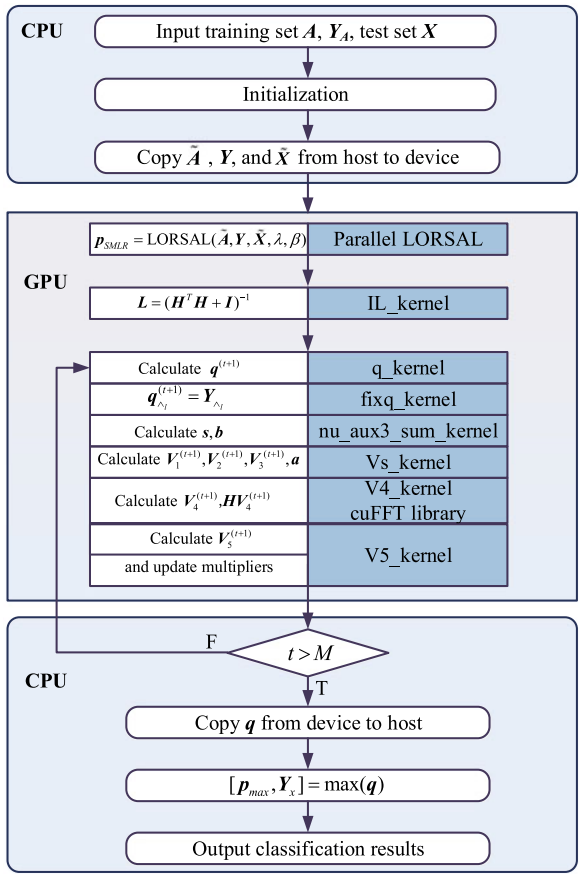


Fig. 6. Detailed illustration of our GPU parallel hyperspectral image classification algorithm based on spatially adaptive MRFs (SAHIC_P).

IV. Experimental Results

A. GPU Platforms and Experimental Datasets

In order to evaluate the performance of our GPU parallel algorithm on different high performance computing architectures, two GPU platforms of different CUDA Compute Capabilities (i.e., NVIDIA Tesla C2075 and Tesla K20C) have been used in our experiments. The hardware specifications and computing capabilities of these GPUs and the corresponding CPU platforms, as well as the software specifications, are described in Tables I and II, respectively.

Two widely used hyperspectral images collected by two different imaging spectrometers: the Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) and the Reflective Optics Spectrographic Imaging System (ROSIS), are used to assess the proposed algorithm. A summary of these images, respectively, collected over the Indian Pines region in NW Indiana USA and the Urban area of Pavia University, Pavia, Italy, is given in Table III, as well as in Figs. 7 and 8.

B. Performance Evaluation

In our experiments, we focus on evaluating both the classification accuracy and the computational performance of our GPU-based SAHIC algorithm, as compared with the corresponding serial version and a multicore version. The serial ver-

Algorithm 2: GPU parallel version of SAHIC (SAHIC_P).

Input: Training samples set $\boldsymbol{A} \in R^{L\times J}$, class labels of training samples $\boldsymbol{Y}_A \in R^{K\times J}$, test samples set $\boldsymbol{X} \in R^{L\times N}$

Initialization: Set $\lambda > 0$, $\lambda_C > 0$, $\beta > 0$, $\mu > 0$, $t = 0$, $M = MaxIteration$, $\tilde{\boldsymbol{A}} = h(\boldsymbol{A})$, $\tilde{\boldsymbol{X}} = \boldsymbol{h}(\boldsymbol{X})$, $\boldsymbol{h}(x)$ is the radial basis function (RBF), initialize $\boldsymbol{V}_1^{(0)}, \boldsymbol{V}_2^{(0)}, \boldsymbol{V}_3^{(0)}, \boldsymbol{V}_4^{(0)}, \boldsymbol{V}_5^{(0)}, \boldsymbol{D}_1^{(0)}, \boldsymbol{D}_2^{(0)}, \boldsymbol{D}_3^{(0)}, \boldsymbol{D}_4^{(0)}, \boldsymbol{D}_5^{(0)}$

Step 1. Copy data from host to device.

Step 2. Invoke $LORSAL\_P(\tilde{\boldsymbol{A}}, \boldsymbol{Y}_A, \tilde{\boldsymbol{X}}, \lambda_C, \beta)$ to calculate $\boldsymbol{p}_C$.

Step 3. Calculate $\boldsymbol{q}$ on GPU

Step 3.1. Invoke IL_kernel to compute $\boldsymbol{L}$.

Do:

Step 3.2. Invoke q_kernel to calculate $\boldsymbol{q}$, and invoke fixq_kernel to perform the execution of $\boldsymbol{q}_{\wedge_l}^{(t+1)} = \boldsymbol{Y}_{\wedge_l}$.

Step 3.3. Invoke nu_aux3_sum_kernel to calculate $\boldsymbol{s}$ and $\boldsymbol{b}$ by means of column batch sum.

Step 3.4. Invoke Vs_kernel to consolidate the calculation of $\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3$

Step 3.5. Invoke V4_kernel to compute in parallel the operations of matrix addition and subtraction in calculating $\boldsymbol{V}_4$.

Step 3.6. Perform the DFT on the GPU by employing the cuFFT library to calculate the $\boldsymbol{HV}_4^{(t+1)}$

Step 3.7. Invoke V5_kernel to perform the operation $\boldsymbol{V}_5^{(t+1)} = \text{soft}(\boldsymbol{D}_5^{(t)} - \boldsymbol{HV}_4^{(t+1)}, \lambda/\mu)$, and update the multipliers.

While $t > M$

Step 4. Copy $\boldsymbol{q}$ from device to host.

Step 5. Compute $[\boldsymbol{q}_{\max}, \boldsymbol{Y}_x] = \max(\boldsymbol{q})$ on CPU.

Output: $\boldsymbol{Y}_X$, the class labels of training samples $\boldsymbol{X}$.

TABLE I
HARDWARE SPECIFICATIONS AND COMPUTING CAPABILITIES OF THE CONSIDERED PLATFORMS

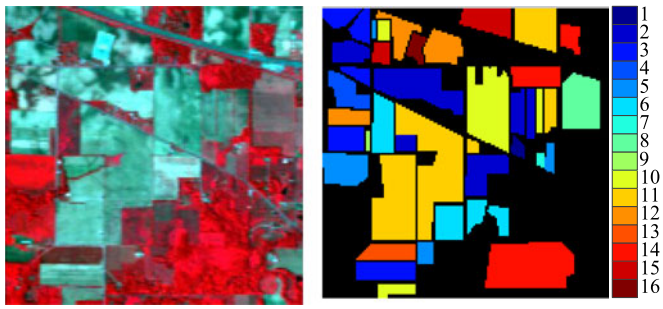
	Specification	NVIDIA Tesla C2075 Platform	NVIDIA Tesla K20C Platform
CPU	Processor Number	Intel Xeon E5-2609	Intel Xeon E5-2620 v2
	Processor Base Frequency	2.4 GHz	2.10 GHz
	Number of Cores	8 in total (2 CPUs)	24 in total (2 CPUs)
	Main Memory	32 GB	16 GB
GPU	Architecture	Fermi	Kepler
	Frequency of CUDA Cores	1150 MHz	706 MHz
	Number of CUDA Cores	448	2496
	Double Precision Floating Point Performance (Peak)	0.515 Tflops	1.17 Tflops
	Single Precision Floating Point Performance (Peak)	1.03 Tflops	3.52 Tflops
	Dedicated Memory	6 GB	5 GB
	Memory Interface	384-bit	320-bit
	Memory Bandwidth	144 GB/sec	208 GB/sec
	CUDA Compute Capability (version)	2.0	3.5

TABLE II
SOFTWARE SPECIFICATIONS OF THE CONSIDERED PLATFORMS

Specification	NVIDIA Tesla C2075 Platform	NVIDIA Tesla K20C Platform
OS	Windows 7 64bit	Windows 7 64bit
CUDA version	5.5	6.0
CULA library	R17	R17
MKL version	11.2	11.2
OpenMP	2.0	2.0
Compiler	Visual C++ 2010	Visual C++ 2010

TABLE III
DETAILS OF THE TWO EXPERIMENTAL HYPERSPECTRAL IMAGES USED IN OUR EXPERIMENTS

Scenes	Indian Pines Dataset	Pavia University Dataset
Instruments	AVIRIS	ROSIS
Image spatial size	145∗145	610∗340
Spectral bands	200 (220 in total, and 20 noise and water absorption bands are removed)	103 (115 in total, and 12 noise bands are removed)
Wavelength range	0.4–2.5 μm	0.43–0.86 μm
Spectral resolution	10 nm	4 nm
Spatial resolution	20 m	1.3 m
Ground-truth classes	16 Details are shown in Fig. 7(b)	9 Details are shown in Fig. 8(b)
Labeled pixels	10366	43923



(a) (b)

Fig. 7. AVIRIS Indian Pines hyperspectral dataset. (a) False color composition. (b) Ground truth as a collection of mutually exclusive classes.

sion (denoted as SAHIC_S) has been implemented in the C++ programming language and used as a basis for the subsequent parallel implementations. This version has been executed in one CPU core of the considered platforms. The multicore version of SAHIC (denoted as SAHIC_M) has been implemented following the design principles in [43] using the OpenMP language and the Intel Math Kernel (MKL), a library of optimized math routines provided by Intel [44]. This version exploits all the cores in the considered CPU platforms. We evaluate the performance improvements (acceleration factors) achieved by the multicore version and the GPU implementation over the serial version.

On the other hand, three metrics have been used to measure the classification accuracies [45].

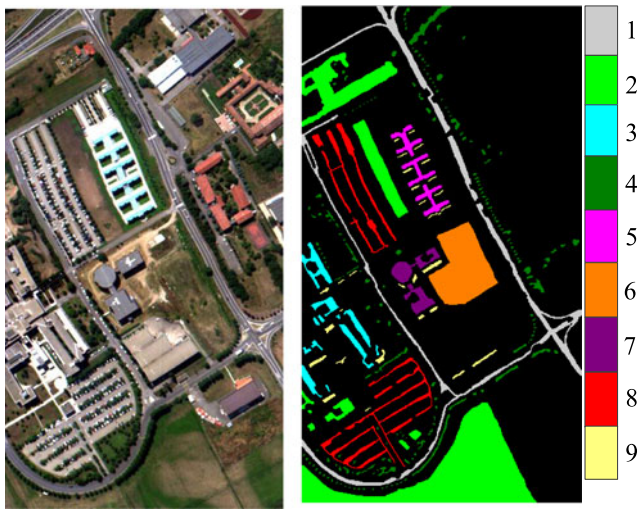


Fig. 8. ROSIS Pavia University hyperspectral dataset. (a) False color composition. (b) Ground truth as a collection of mutually exclusive classes.

1) Overall accuracy (OA), computed as the ratio between the correctly classified test samples and the total number of test samples.
2) Average accuracy (AA), the mean of the accuracies across the different classes.
3) Kappa statistic (Kappa), computed by weighting the measured accuracies. It incorporates both of the diagonal and off-diagonal entries of the confusion matrix and is a robust measure in terms of the degree of agreement.

According to Sun *et al.* [32] and our repeated experiments, the parameters were empirically set to $\lambda_C = 0.001, \beta = 0.0001$, $\mu = 0.05$, $\mu_s = 2$, $\delta = 0.8$ (RBF kernel parameter), and $\lambda = 2$ for the AVIRIS Indian Pines scene, and $\delta = 0.35$ and $\lambda = 1$ for ROSIS Pavia University image. For each value reported in experiments, ten Monte Carlo runs were performed and the average values were reported.

We first evaluated the classification performance using the ROSIS Pavia University dataset. We randomly chose 40 labeled pixels from each class as training samples, and used the remaining labeled pixels as test samples. Moreover, we used exactly the same training-test sets for the three considered versions of SAHIC when compared the achieved classification accuracies in a fair way. Table IV summarizes the classification accuracies (OA, AA, and Kappa), measured after processing the ROSIS Pavia University dataset on the two considered platforms. Some of the obtained classification maps are given in Fig. 9. It is worth noting that the proposed SAHIC_S, SAHIC_M, and SAHIC_P obtain exactly the same classification accuracies (when using exactly the same training and test sets) on the considered platforms, leading to very smooth maps of classification as depicted in Fig. 9. We find that the time they need to complete their calculations is the only difference between the serial and parallel versions. The corresponding timing results (in seconds), and the acceleration factors (speedups) are shown in Table V.

It is obvious from Table V that the parallel version SAHIC_P achieves remarkable acceleration factors on both platforms as compared to both the serial and multicore versions. This is because the SAHIC_P takes full advantages of the computational power of GPUs, the high bandwidth, as well as low latency of shared memory, and benefits from exploiting the massively parallel nature of GPUs. Specifically, the SAHIC_P achieves high speedups of about 81× and 121×, respectively, with regards to the serial version in the two considered GPU platforms.

Now, we evaluate the performance of the considered classifiers using the AVIRIS Indian Pines dataset. In this test, we randomly chose nearly 10% of the labeled pixels of each class (1043 pixels in total) as training samples, and used the remaining labeled pixels as test samples. For illustrative purposes, Fig. 10 shows some of the classification maps and Table VI reports the

TABLE IV
CLASSIFICATION ACCURACIES (%) OBTAINED FOR THE ROSIS PAVIA UNIVERSITY DATASET

Class	Class Name	Training Samples	Test Samples	Accuracy
1	Asphalt	40	6591	93.56
2	Meadows	40	18609	94.71
3	Gravel	40	2059	92.43
4	Trees	40	3024	89.13
5	Metal sheets	40	1305	99.7
6	Bare soil	40	4989	100
7	Bitumen	40	1290	99.92
8	Bricks	40	3642	98.72
9	Shadows	40	907	99.09
	OA (%)			95.36
	AA(%)	360	42416	96.36
	Kappa			0.9394

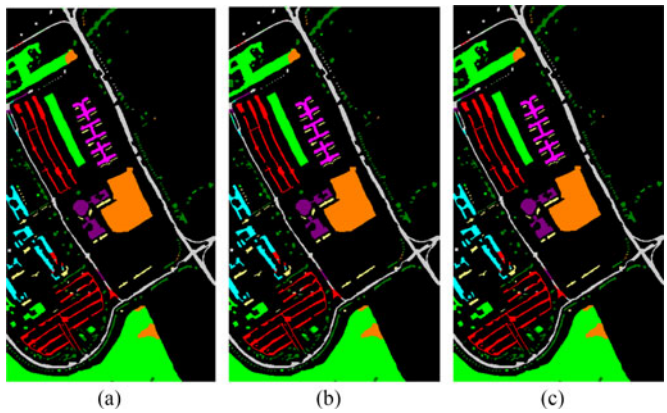


Fig. 9. Some of the obtained classification maps for the ROSIS University of Pavia dataset. (a) Result of SAHIC_S. (b) Result of SAHIC_M. (c) Result of SAHIC_P.

TABLE V
EXECUTION TIMES AND ACCELERATION FACTORS OBTAINED FOR THE ROSIS PAVIA UNIVERSITY DATASET

Implementations	Tesla C2075 Platform			Tesla K20C Platform		
	SAHIC_S	SAHIC_M	SAHIC_P	SAHIC_S	SAHIC_M	SAHIC_P
Time (sec)	1427.69	179.08	17.73	1309.54	157.89	10.85
Acceleration factor (X)	–	7.97	80.52	–	8.29	120.69

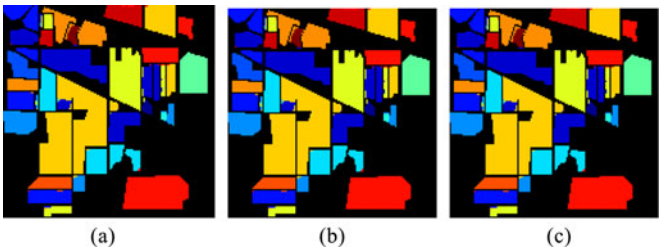


Fig. 10. Some of the obtained classification maps of the AVIRIS Indian Pines dataset. (a) Result of SAHIC_S. (b) Result of SAHIC_M. (c) Result of SAHIC_P.

TABLE VI
CLASSIFICATION ACCURACIES (%) OBTAINED FOR THE AVIRIS INDIAN PINES DATASET

Class	Class Name	Training Samples	Test Samples	Accuracy
1	Alfalfa	6	48	91.67
2	Corn-no till	144	1290	97.05
3	Corn-min till	84	750	98.4
4	Corn	24	210	100
5	Grass/Pasture	50	447	95.3
6	Grass/Trees	75	672	99.11
7	Grass/Pasture-mowed	3	23	78.26
8	Hay-windrowed	49	440	100
9	Oats	2	18	16.67
10	Soybeans-no till	97	871	97.7
11	Soybeans-min till	247	2221	99.77
12	Soybean-clean till	62	552	99.82
13	Wheat	22	190	100
14	Woods	130	1164	100
15	Building-Grass-Trees-Drives	38	342	97.66
16	Stone-steel Towers	10	85	74.12
	OA(%)	1043	9323	98.32
	AA(%)			90.35
	Kappa			0.9808

TABLE VII
EXECUTION TIMES AND ACCELERATION FACTORS OBTAINED FOR THE AVIRIS INDIAN PINES DATASET

Implementation	Tesla C2075 Platform			Tesla K20C Platform		
	SAHIC_S	SAHIC_M	SAHIC_P	SAHIC_S	SAHIC_M	SAHIC_P
Time (sec)	228.66	43.66	4.62	211.02	38.17	2.95
Acceleration factor (X)	–	5.24	49.49	–	5.53	71.53

classification accuracies of ten Monte Carlo runs. The results reported in Table VI demonstrate that the SAHIC_S, SAHIC_M, and SAHIC_P obtain very competitive results on both GPU platforms. Here it should be also noted that the accuracies obtained by the three implementations of SAHIC are exactly the same, when using the same train and test sets.

Table VII reports the execution times and acceleration factors obtained for the AVIRIS Indian Pines dataset. As it can be seen in Table VII, the processing time of SAHIC_P is less than 5 s, including the I/O data transfer times between device and host.

TABLE VIII
EXECUTION TIMES AND ACCELERATION FACTORS OF THE KERNEL FUNCTIONS AND ITS CORRESPONDING SERIAL VERSION IN ONE ITERATION

Pavia University Dataset			
Kernels	Serial Time(s)	Parallel Time(s)	Speedup
q_kernel	0.07368	0.00002	3312.98
fixq_kernel	0.00012	0.00003	3.52
nu_aux3_sum_kernel	0.00692	0.00003	200.26
Vs_kernel	0.06727	0.00004	1841.65
V4_kernel	0.02709	0.00006	476.19
V5_kernel	0.08223	0.00017	477.58
Indian Pines Dataset			
Kernels	Serial Time(s)	Parallel Time(s)	Speedup
q_kernel	0.01734	0.00002	929.66
fixq_kernel	0.00056	0.00004	14.11
nu_aux3_sum_kernel	0.00191	0.00003	57.25
Vs_kernel	0.01482	0.00005	303.34
V4_kernel	0.00538	0.00015	35.29
V5_kernel	0.01442	0.00031	46.41

It shows that our proposed parallel implementation makes a significant improvement with regard to the versions of SAHIC_S and SAHIC_M. The best speedup obtained in our experiments is above 70 × (achieved on the NVIDIA Tesla K20C platform). These results exhibit the potential of GPUs for parallelizing the considered spectral-spatial classification algorithm.

It is worth noting that our proposed GPU parallel implementation is efficient on different GPU computing architectures (for instance, Fermi and Kepler), and is scalable with different CUDA compute capabilities. Specifically, even with the CUDA compute capabilities of 2.0, the SAHIC_P can achieve significant speedup with regards to the serial version in the Tesla C2075 platform.

To further evaluate the efficiency of the proposed parallel approach, we perform the tests on the Tesla K20C Platform, and compare the computation time of kernel functions in our GPU implementation with their corresponding serial versions. As can be seen from Table VIII, most of the kernel functions achieve remarkable acceleration factors, especially for the most time-consuming step of the calculation of $\boldsymbol{V}_5$, which is ideal in terms of parallel efficiency.

Let us now take a closer look at the performance of the parallel spatial-spectral classifier SAHIC_P as compared to the parallel spectral-only classifier LORSAL_P in [7]. For simplicity, we present an illustrative example using the AVIRIS Indian Pines dataset. Fig. 11 compares the classification accuracies and the computational performance of these two classifiers. It can be concluded from Fig. 9 that the proposed SAHIC_P greatly improves the classification accuracy with some additional computational burden, as expected. In particular, as noted by Tables VI and VII, the total processing time of SAHIC_P is less than 3 s on the NVIDIA Tesla K20C, and its accuracy reaches up to 98.32%, which is very compelling for an efficient hyperspectral image classification method.

Fig. 12 shows the percentages of the data transfer time between the host and the device for the proposed parallel method, which is also an important issue for GPU parallelization. It can

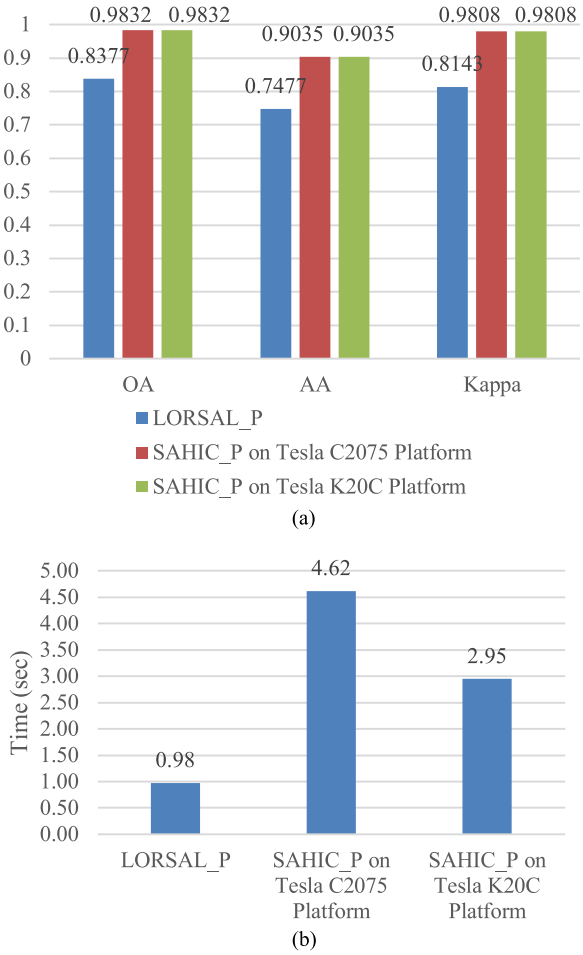


Fig. 11. Comparison between LORSAL_P in [7] and the proposed SAHIC_P on the AVIRIS Indian Pines dataset. (a) Classification accuracy. (b) Computational performance.

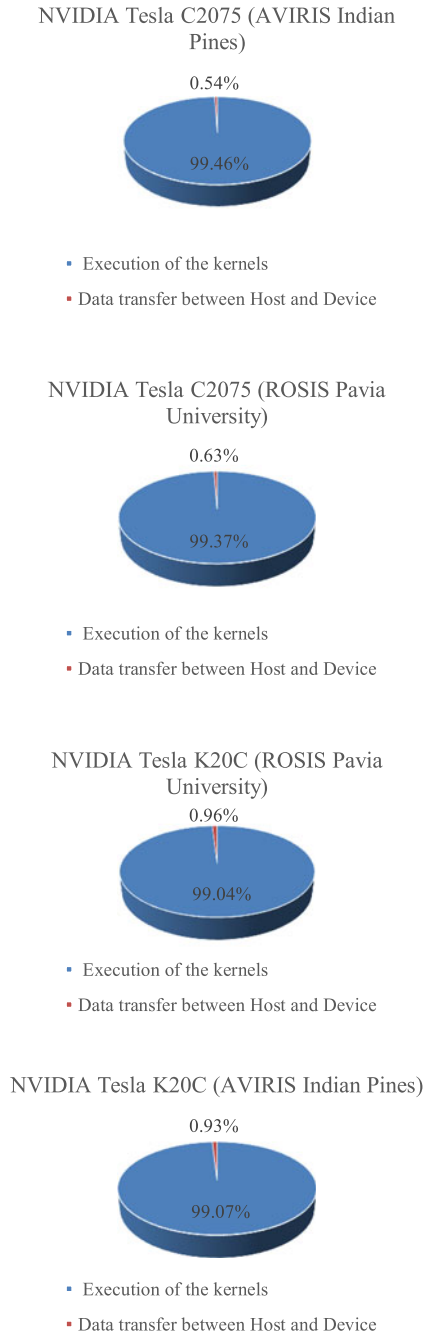


Fig. 12. Percentages of the total GPU time for kernel execution and data transfers.

be observed from Fig. 12 that, in our GPU parallel implementation, the data transfer time is relatively low. In particular, our implementation uses more than 99% of the total GPU time for executing the kernels and less than 1.0% of the time for memory transfers in all cases. Therefore, for SAHIC_P, the most significant portion of the time is taken by pure computation steps, which can be defined as compute-bound. We emphasize that the performance of the algorithm is driven by the processing units themselves (not by data transfers) and, as a result, the algorithm is expected to scale very effectively for larger size problems.

Several conclusions can be obtained from the experiments reported in this section. First and foremost, our parallel SAHIC_P produces good results in terms of both classification accuracy and computational performance. The algorithm achieves significant acceleration factors in the two considered GPU architectures, while in all experiments the classification accuracies achieved by the parallel algorithm were shown to be exactly the same to those obtained by the serial and multicore versions of SAHIC. Additionally, the SAHIC_P offered significant improvements (in terms of classification accuracy and parallel performance) with regards to the GPU implementation of a spectral-based classifier (LORSAL_P). This fact results from the inclusion of spatial-contextual information, which is also accurately modeled and parallelized in the GPU. We believe that these performance improvements for the considered spatial-spectral classifier SAHIC can potentially enhance its applicability in classification problems with time-critical constraints, thus enhancing proper decision-making in these scenarios.

V. Conclusion and Future Work

The incorporation of spatial-contextual information to spectral-based characterization is crucial for hyperspectral image classification. Despite the availability of many techniques

for spectral-spatial classification in the hyperspectral imaging literatures, very few techniques have been effectively implemented in parallel for their application in time-critical scenarios. In this paper, we develop a new GPU parallel implementation of a spatial-spectral hyperspectral image classification method based on spatially adaptive MRFs, taking full advantage of the computational power offered by commodity GPUs. Our experimental assessment, conducted using two GPU platforms, demonstrated the superior performance of the proposed spectral-spatial classifier in terms of both classification accuracy and computational performance, using two widely used benchmark hyperspectral scenes. Specifically, our experiments demonstrate that the proposed method is more accurate and faster than several other available techniques for hyperspectral image classification, including the serial and multicore versions of the same algorithm. In future works, we will explore other parallelization frameworks for spectral-spatial classifiers using hardware platforms with lower power consumption ratios, such as FPGAs.

REFERENCES

[1] 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.

[2] A. Plaza *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, pp. S110–S122, Sep. 2009.

[3] A. Plaza, J. M. Bioucas-Dias, A. Simic, and W. J. Blackwell, "Foreword to the special issue on hyperspectral image and signal processing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 347–353, Apr. 2012.

[4] M. Fauvel, J. A. Benediktsson, J. Chanussot, and J. R. Sveinsson, "Spectral and spatial classification of hyperspectral data using SVMs and morphological profiles," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 11, pp. 3804–3814, Nov. 2008.

[5] B. Krishnapuram, L. Carin, M. A. Figueiredo, and A. J. Hartemink, "Sparse multinomial logistic regression: Fast algorithms and generalization bounds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 6, pp. 957–968, Jun. 2005.

[6] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Semisupervised hyperspectral image classification using soft sparse multinomial logistic regression," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 2, pp. 318–322, Mar. 2013.

[7] Z. Wu, Q. Wang, A. Plaza, J. Li, L. Sun, and Z. Wei, "Real-time implementation of the sparse multinomial logistic regression for hyperspectral image classification on GPUs," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 7, pp. 1456–1460, Jul. 2015.

[8] F. Xiao, G. Ge, L. Sun, and R. Wang, "An energy-efficient data gathering method based on compressive sensing for pervasive sensor networks," *Pervasive Mobile Comput.*, 2017. doi: 10.1016/j.pmcj.2017.02.005, to be published.

[9] A. Plaza, Q. Du, Y. Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 3, pp. 528–544, Sep. 2011.

[10] F. Xiao, Z. Jiang, X. Xie, L. Sun, and R. Wang, "An energy-efficient data transmission protocol for mobile crowd sensing," *Peer-to-Peer Netw. Appl.*, vol. 10, no. 3, pp. 510–518, 2017.

[11] C. Gonzalez, D. Mozos, J Resano, and A. Plaza, "FPGA implementation of the N-FINDR algorithm for remotely sensed hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 2, pp. 374–388, Feb. 2012.

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

[13] S. Bernabe, S. Sanchez, A. Plaza, S. Lopez, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, Mar. 2013.

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

[15] J. M. P. Nascimento, J. M. Bioucas-Dias, J. M. Rodriguez Alves, V. Silva, and A. Plaza, "Parallel hyperspectral unmixing on GPUs," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 3, pp. 666–670, Mar. 2014.

[16] Y. Wu and L. Gao, "Graphics processing unit–accelerated computation of the Markov random fields and loopy belief propagation algorithms for hyperspectral image classification," *J. Appl. Remote Sens.*, vol. 9, no. 1, 2015, Art. no. 097295.

[17] Z. Wu, S. Ye, J. Liu, L. Sun, and Z. Wei, "Sparse nonnegative matrix factorization on GPUs for hyperspectral unmixing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 8, pp. 3640–3649, Aug. 2014.

[18] R. Ji, Y. Gao, R. Hong, Q. Liu, D. Tao, and X. Li, "Spectral-spatial constraint hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 3, pp. 1811–1824, Mar. 2014.

[19] J. Li, P. Reddy Marpu, A. Plaza, J. Bioucas-Dias, and J. Atli Benediktsson, "Generalized composite kernel framework for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 9, pp. 4816–4829, Sep. 2013.

[20] J. Liu, Z. Wu, Z. Wei, L. Xiao, and L. Sun, "Spatial-spectral kernel sparse representation for hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 6, pp. 2462–2471, Dec. 2013.

[21] G. Camps-Valls, N. Shervashidze, and K. M. Borgwardt, "Spatio-spectral remote sensing image classification with graph kernels," *IEEE Geosci. Remote Sens. Lett.*, vol. 7, no. 4, pp. 741–745, Oct. 2010.

[22] M. Fauvel, J. A. Benediktsson, J. Chanussot, and J. R. Sveinsson, "Spectral and spatial classification of hyperspectral data using SVMs and morphological profiles," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 11, pp. 3804–3814, Nov. 2008.

[23] X. Kang, S. Li, and J. Benediktsson, "Spectral-spatial hyperspectral image classificationwith edge-preserving filtering," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 5, pp. 2666–2677, May 2014.

[24] 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, Jul. 2010.

[25] S. Li, B. Zhang, A. Li, X. Jia, L. Gao, and M. Peng, "Hyperspectral imagery clustering with neighborhood constraints," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 3, pp. 588–592, Mar. 2013.

[26] X. Sun, Q. Qu, N. M. Nasrabadi, and T. D. Tran, "Structured priors for sparse-representation-based hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 7, pp. 1235–1239, Jul. 2014.

[27] L. Zhang *et al.*, "Kernel sparse representation based classifier," *IEEE Trans. Signal Process.*, vol. 60, no. 4, pp. 1684–1695, Apr. 2012.

[28] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Spectral–spatial hyperspectral image segmentation using subspace multinomial logistic regression and Markov random fields," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 3, pp. 809–823, Mar. 2012.

[29] Y. Tarabalka, M. Fauvel, J. Chanussot, and J. A. Benediktsson, "SVM-and MRF-based method for accurate classification of hyperspectral images," *IEEE Geosci. Remote Sens. Lett.*, vol. 7, no. 4, pp. 736–740, Apr. 2010.

[30] B. Zhang, S. Li, X. Jia, L. Gao, and M. Peng, "Adaptive Markov random field approach for classification of hyperspectral imagery," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 5, pp. 973–977, May 2011.

[31] G. Moser and S. B. Serpico, "Combining support vector machines and Markov random fields in an integrated framework for contextual image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 5, pp. 2734–2752, May 2013.

[32] L. Sun, Z. Wu, J. Liu, L. Xiao, and Z. Wei, "Supervised spectral-spatial hyperspectral image classification with weighted Markov random fields (MRFs)," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 5, pp. 1490–1503, Mar. 2015.

[33] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Semisupervised hyperspectral image segmentation using multinomial logistic regression with active learning," *IEEE Trans. Geosci. Remote Sens.*, vol. 48, no. 11, pp. 4085–4098, Nov. 2010.

[34] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Spectral–spatial classification of hyperspectral data using loopy belief propagation and active learning," *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 2, pp. 844–856, Feb. 2013.

[35] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Hyperspectral image segmentation using a new Bayesian approach with active learning," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 10, pp. 3947–3960, Oct. 2011.

[36] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Spectral–spatial hyperspectral image segmentation using subspace multinomial logistic regression and Markov random fields," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 3, pp. 809–823, Mar. 2012.
[37] M. D. Iordache, J. M. Bioucas-Dias, and A. Plaza, "Total variation spatial regularization for sparse hyperspectral unmixing," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 11, pp. 4484–4502, Nov. 2012.
[38] J. M. Bioucas-Dias and M. A. T. Figueiredo, "Alternating direction algorithms for constrained sparse regression: Application to hyperspectral unmixing," in *Proc. 2nd Workshop Hyperspectral Image Signal Process. Evol. Remote Sens.*, 2010. pp. 1–4.
[39] W. Deng, W. Yin, and Y. Zhang, "Group sparse optimization by alternating direction method," Dept. Comput. Appl. Math., Rice Univ., Houston, TX, USA, 2011.
[40] J. M. Bioucas-Dias and M. Figueiredo, "Logistic regression via variable splitting and augmented Lagrangian tools," Instituto Superior T´ecnico, Tech. Univ. Lisbon, Lisbon, Portugal, Tech. Rep., 2009.
[41] NVIDIA Developer Zone, "Cufft library user's guide," Mar. 2015. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf
[42] NVIDIA CUDA Zone, "CUDA toolkit," Mar. 2015. [Online]. Available: https://developer.nvidia.com/cuda-toolkit
[43] S. Bernabe, S. Sanchez, A. Plaza, S. Lopez, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, Mar. 2013.
[44] Intel Developer Zone, "Reference manual for intel® math kernel library 11.2," Nov. 2014. [Online]. Available: https://software.intel.com/en-us/mkl_11.2_ref
[45] G. M. Foody, "Classification accuracy comparison: hypothesis tests and the use of confidence intervals in evaluations of difference, equivalence and non-inferiority," *Remote Sens. Environ.*, vol. 113, no. 8, pp. 1658–1663, Aug. 2009.

Zebin Wu (M'13) was born in Zhejiang, China, in 1981. He received the B.Sc. and Ph.D. degrees in 2003 and 2007, respectively, all in computer science and technology, from Nanjing University of Science and Technology.

He is currently a Professor in the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. He was a Visiting Scholar in the Department of Mathematics, University of California, Los Angeles, CA, USA, from August 2016 to September 2016, and from July 2017 to August 2017. From June 2014 to June 2015, he was a Visiting Scholar in the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Cáceres, Spain. His research interests include hyperspectral image processing, high performance computing, and computer simulation.

Linlin Shi was born in Jiangsu, China, in 1993. He received the B.Sc. degree in computer science and technology from the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China, in 2016. He is currently working toward the M.S. degree in computer application, with the School of Computer Science and Engineering, Nanjing University of Science and Technology.

His research interests include hyperspectral image classification, deep learning, and high performance computing.

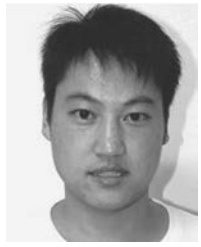
Jun Li (M'13) received the B.S. degree in geographic information systems from Hunan Normal University, Changsha, China, in 2004, the M.E. degree in remote sensing from Peking University, Beijing, China, in 2007, and the Ph.D. degree in electrical engineering from the Instituto de Telecomunicações, Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 2011.

From 2007 to 2011, she was a Marie Curie Research Fellow with the Departamento de Engenharia Electrotécnica e de Computadores and the Instituto de Telecomunicações, IST, Universidade Técnica de Lisboa, in the framework of the European Doctorate for Signal Processing. She has also been actively involved in the hyperspectral imaging network, a Marie Curie Research Training Network involving 15 partners in 12 countries and intended to foster research, training, and cooperation on hyperspectral imaging at the European level. Since 2011, she has been a Postdoctoral Researcher with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Caceres, Spain. She has been a Reviewer of several journals, including the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, *Pattern Recognition*, *Optical Engineering*, *Journal of Applied Remote Sensing*, and *Inverse Problems and Imaging*. Her research interests include hyperspectral image classification and segmentation, spectral unmixing, signal processing, and remote sensing.

Dr. Li received the 2012 Best Reviewer Award of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING.

Qicong Wang was born in Henan, China, in 1988. He received the B.Sc. degree in computer science and technology from the School of Computer, Henan University of Science and Technology, Luoyang, China, in 2012. He is currently working toward the M.S. degree in computer application, with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China.

His research interests include hyperspectral image classification and high performance computing.

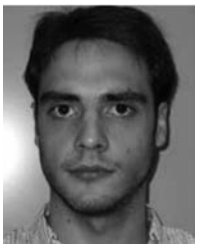


Le Sun (M'13) was born in Jiangsu, China, in 1987. He received the B.S. degree in applied mathematics from the School of Science, Nanjing University of Science and Technology, Nanjing, China, in 2009, where he is currently working toward the Ph.D. degree in pattern recognition, with the School of Computer Science and Engineering.

His research interests include the area of spectral unmixing, hyperspectral classification, image processing, sparse representation, and compressive sensing.

Zhihui Wei was born in Jiangsu, China, in 1963. He received the B.Sc. and M.Sc. degrees in applied mathematics and the Ph.D. degree in communication and information system all from the South East University, Nanjing, China, in 1983, 1986, and 2003, respectively.

He is currently a Professor and Doctoral Supervisor in Nanjing University of Science and Technology, Nanjing, China. His main research interests include partial differential equations, image processing, multiscale analysis, sparse representation, and compressed sensing.



Javier Plaza (M'09–SM'15) received the M.Sc. and Ph.D. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 2004 and 2008, respectively.

He is currently a Member of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 120 publications, including 36 JCR journal papers, ten book chapters, and 80 peer-reviewed conference proceeding papers. He has guest edited two special issues on hyperspectral remote sensing for different journals. His main research interests include hyperspectral data processing and parallel computing of remote sensing data.

Dr. Plaza is an Associate Editor for the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS and an Associate Editor of the IEEE REMOTE SENSING CODE LIBRARY. He received the Outstanding Ph.D. Dissertation Award from the University of Extremadura in 2008. He received the Best Column Award of the IEEE SIGNAL PROCESSING MAGAZINE in 2015 and the Most Highly Cited Paper (during 2005–2010) in the *Journal of Parallel and Distributed Computing*, and also Best Paper Awards at the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology.

Antonio Plaza (M'05–SM'07–F'15) received the M.Sc. and Ph.D. degrees both in computer engineering from the University of Extremadura, Cáceres, Spain, in 1999 and 2002, respectively.

He is the Head of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, Cáceres, Spain. His main research interests include comprise hyperspectral data processing and parallel computing of remote sensing data. He has authored more than 600 publications, including 197 JCR journal papers (more than 140 in IEEE journals), 23 book chapters, and 285 peer-reviewed conference proceeding papers. He has guest edited ten special issues on hyperspectral remote sensing for different journals.

Prof. Plaza is a Fellow of the IEEE "for contributions to hyperspectral data processing and parallel computing of Earth observation data." He received the Best Reviewers of the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS (in 2009) and the Best Reviewers of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING (in 2010), for which he served as Associate Editor during 2007–2012. He is also an Associate Editor for IEEE ACCESS, and was a member of the Editorial Board of the IEEE GEOSCIENCE AND REMOTE SENSING NEWSLETTER (during 2011–2012) and the IEEE GEOSCIENCE AND REMOTE SENSING MAGAZINE (in 2013). He was also a member of the steering committee of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He received the Best Column Award of the IEEE SIGNAL PROCESSING MAGAZINE in 2015, the 2013 Best Paper Award of the JSTARS journal, and the most highly cited paper (during 2005–2010) in the *Journal of Parallel and Distributed Computing*. He received the Best Paper Awards in the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology. He served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS) during 2011–2012, and as a President of the Spanish Chapter of IEEE GRSS during 2012–2016. He has reviewed more than 500 manuscripts for more than 50 different journals. He is currently serving as the Editor-in-Chief of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING.