Parallel Implementation of Sparse Representation Classifiers for Hyperspectral Imagery on GPUs

Zebin Wu, *Member, IEEE*, Qicong Wang, Antonio Plaza, *Fellow, IEEE*, Jun Li, *Member, IEEE*, Jianjun Liu, and Zhihui Wei

***Abstract*—Classification is one of the most important analysis techniques for hyperspectral image analysis. Sparse representation is an extremely powerful tool for this purpose, but the high computational complexity of sparse representation-based classification techniques limits their application in time-critical scenarios. To improve the efficiency and performance of sparse representation classification techniques for hyperspectral image analysis, this paper develops a new parallel implementation on graphics processing units (GPUs). First, an optimized sparse representation model based on spatial correlation regularization and a spectral fidelity term is introduced. Then, we use this approach as a case study to illustrate the advantages and potential challenges of applying GPU parallel optimization principles to the considered problem. The first GPU optimization algorithm for sparse representation classification (SRCSC_P) of hyperspectral images is proposed in this paper, and a parallel implementation of the proposed method is developed using compute unified device architecture (CUDA) on GPUs. The GPU parallel implementation is compared with the serial and multicore implementations on CPUs. Experimental results based on real hyperspectral datasets show that the average speedup of SRCSC_P is more than 130×, and the proposed approach is able to provide results accurately and fast, which is appealing for computationally efficient hyperspectral data processing.**

***Index Terms*—Classification, graphics processing units (GPUs), hyperspectral image, parallel optimization, sparse representation.**

Manuscript received August 21, 2014; revised January 03, 2015; accepted March 13, 2015. Date of publication March 30, 2015; date of current version July 30, 2015. This work was supported in part by the National Natural Science Foundation of China under Grant 61471199, Grant 61101194, and Grant 11431015, in part by the Jiangsu Provincial Natural Science Foundation of China under Grant BK2011701, in part by the China Scholarship Fund under Grant 201406845012, in part by the Research Fund for the Doctoral Program of Higher Education of China under Grant 20113219120024, in part by the Jiangsu Province Six Top Talents project of China under Grant WLW-011, and in part by the China Academy of Space Technology Innovation Foundation under Grant CAST201227.

Z. Wu is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, 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).

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

J. Li is with the Guangdong Provincial Key Laboratory of Urbanization and Geo-simulation and 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).

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

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

Digital Object Identifier 10.1109/JSTARS.2015.2413831

I. Introduction

HYPERSPECTRAL imaging instruments collect hundreds of narrow spectral bands spanning the visible to infrared parts of the electromagnetic spectrum. In hyperspectral image cubes, each pixel can be represented by a vector whose entries correspond to the spectral bands, providing a representative spectral signature of the underlying materials within the pixel [1]. Since different materials often reflect electromagnetic energy differently at specific wavelengths, it is possible to discriminate various materials using the spectral signature characteristics.

During recent years, hyperspectral remote sensing has been widely used in various fields of Earth observation and space exploration. Classification is one of the most popular and important techniques for hyperspectral image interpretation. It amounts at labeling the pixels into a set of predefined classes [2]. Over the last decade, various supervised and unsupervised methods have been developed for hyperspectral image classification [3], [4], including support vector machines (SVMs) [5], [6], artificial neural networks (ANNs) [7], multinomial logistic regression (MLR) [8], orthogonal matching pursuit (OMP) [9], and composite kernels (CK) [10]–[12]. Among this extensive set of techniques, SVMs have been shown to be highly effective for solving the supervised classification problem from the viewpoint of both computational performance and classification accuracy. Several variations of SVM-based methods have also been proposed: multiple kernel SVMs [5], Markov random field (MRF)-based SVMs [14], or adaptive MRF-based SVMs [15], among several others.

With the recent advances in compressed sensing and sparse representation theory and applications [16]–[18], sparse representation-based classification (SRC) methods have received great attention in hyperspectral imaging [20], [21]. This approach relies on the fact that a single pixel in a hyperspectral image can be sparsely represented by a linear combination of a few training samples from a structured dictionary. The sparse representation of an unknown pixel is expressed as a sparse vector whose nonzero entries correspond to the weights of the selected training samples [19]. The class label of the test sample can be easily determined via the sparse vector that is recovered by solving a sparse optimization problem.

Previous work has shown that SRC methods often lead to the state-of-the-art classification accuracy in hyperspectral imaging [19], [20]. However, their application to real problems is compromised by the fact that they often suffer from instability and high complexity of the model solution.

1) On one hand, the instability of the SRC solution is mainly due to the high mutual coherency of the dictionary [22]. Fortunately, this aspect can be improved by either exploring the dependencies of neighboring pixels or by exploiting the inherent dictionary structure. It is well known that in hyperspectral images, neighboring pixels usually comprise similar materials and have similar spectral characteristics, especially in homogeneous regions. The structured priors [19]–[21] have been exploited in this context to enforce spatial correlations and dependencies on the sparse coefficients and thus improve classification accuracy.
2) On the other hand, the high computational complexity of SRC-based methods often limits their application in time-critical scenarios. Unlike SVMs, SRC searches for dedicated atoms in the training dictionary for each test pixel [19], without involving a training stage as it is the case of SVMs. Thus, SRC-based classification is generally more computationally intensive than SVM-based classification.

Recent advances in high-performance computing have opened new avenues to overcome the aforementioned computational challenges. It is important to note that high-performance computing technologies can be used to accelerate hyperspectral image processing algorithms in order to make them suitable for time-critical scenarios [23]–[25], including target detection for military purposes, monitoring of chemical contamination, wildfire tracking, and so on.

In this regard, graphic processing units (GPUs) have recently emerged as a commodity platform for many compute-intensive, massively parallel, and data-intensive computations [13], [26]. GPU-based parallel computing offers a tremendous potential to bridge the gap toward real-time analysis of hyperspectral images. Among recent developments, a novel GPU-based parallel hyperspectral unmixing method combining two widely used algorithms: vertex component analysis (VCA) and sparse unmixing by variable splitting and augmented Lagrangian (SUNSAL) has been reported in [27]. Two efficient implementations of a full hyperspectral unmixing chain on two different kinds of high-performance computing architectures: GPUs and multicore processors, are described in [28]. A near-real-time automatic target detection algorithm has been reported in [29]. An improved implementation of the pixel purity index (PPI) algorithm for endmember extraction has been described in [30]. A new parallel implementation of the VCA algorithm for spectral unmixing on commodity GPUs is given in [31]. However, to the best of our knowledge, and despite the importance of SRC methods in the hyperspectral imaging, there are no available GPU implementations for this category of algorithms in the literature.

In this paper, we propose a novel parallel SRC method for hyperspectral image classification on GPUs. First, we introduce an optimized SRC model based on a spatial correlation regularization term and a spectral fidelity term. The method promotes the stability and accuracy of classification result by spreading the label information of each training sample to its neighbors until achieving a global stable state on the whole dataset. Then, we use this SRC method as a case study to illustrate the advantages and potential challenges of utilizing GPU parallel computing principles to dramatically improve the computation speed of the proposed approach. The parallel implementation is carried out using NVidia's compute unified device architecture (CUDA), and the results of the presented method are compared with serial and multicore CPU implementations.

This paper is organized as follows. Section II describes the optimized SRC model based on spatial correlation. Section III presents its parallel implementation on GPUs using CUDA. Section IV provides an experimental assessment in terms of classification accuracy and processing performance for the proposed GPU-based parallel algorithm, using real hyperspectral data sets. Finally, Section V concludes with some remarks and hints at plausible future research lines.

II. Optimized Sparse Representation Method Based on Spatial Correlation

A. Sparse Representation Classification (SRC)

SRC can be seen as an effective combination of machine learning and compressed sensing. The test samples are directly classified based on training samples, with no need for a detailed learning procedure. In fact, SRC has been proved to be an extremely powerful tool for hyperspectral image classification [20], which often leads to the state-of-the-art performance. In hyperspectral imaging, a test spectral sample can be approximately represented by a few training samples among a given over-complete training dictionary, and the class label of a test pixel is determined by the characteristics of the sparse vector recovered by a sparsity-constrained optimization problem. The fundamental assumption is that the spectral signals in the same class usually lie in a low-dimensional subspace. To define the problem in mathematical terms, let us assume that there are K distinct classes, including J training samples in total. The kth class has N_k training samples, denoted as $\boldsymbol{A}^k = [\boldsymbol{a}_1^k, \boldsymbol{a}_2^k, \ldots, \boldsymbol{a}_{N_k}^k] \in \boldsymbol{R}^{L \times N_k}$, where L is the number of spectral bands. Let $\boldsymbol{x} \in \boldsymbol{R}^L$ be an L-dimensional hyperspectral pixel vector. If x belongs to the kth class, then its spectrum approximately lies in a low-dimensional subspace spanned by the training samples of the kth class, which can be compactly represented by a linear combination of these training samples as follows:

$$\boldsymbol{x} \approx \boldsymbol{a}_1^k s_1^k + \boldsymbol{a}_2^k s_2^k + \cdots + \boldsymbol{a}_{N_k}^k s_{N_k}^k = \boldsymbol{A}^k s^k \tag{1}$$

where $\boldsymbol{s}^k = [s_1^k, s_2^k, \ldots, s_{N_k}^k]^{\mathrm{T}} \in \boldsymbol{R}^{N_k}$ is an unknown sparse vector whose entries are the weights of the corresponding atoms in the subdictionary $\boldsymbol{A}^k$. An unknown test sample $\boldsymbol{x}$ can be modeled as

$$\boldsymbol{x} = \boldsymbol{A}^1 \boldsymbol{s}^1 + \boldsymbol{A}^2 \boldsymbol{s}^2 + \cdots + \boldsymbol{A}^K \boldsymbol{s}^K = [\boldsymbol{A}^1 \ldots \boldsymbol{A}^K][\boldsymbol{s}^1 \ldots \boldsymbol{s}^K]^{\mathrm{T}} = \boldsymbol{A}\hat{\boldsymbol{s}} \tag{2}$$

where $\boldsymbol{A} = [\boldsymbol{A}^1, \boldsymbol{A}^2, \ldots, \boldsymbol{A}^K] \in \boldsymbol{R}^{L \times J}$ is a structured dictionary formed by the concatenation of several class-wise subdictionaries consisting of training samples, $J = \sum_{k=1}^{K} N_k$, and $\hat{\boldsymbol{s}} = [\boldsymbol{s}^1 \ldots \boldsymbol{s}^K]^{\mathrm{T}} \in \boldsymbol{R}^J$ is a sparse vector formed by concatenating the sparse vectors $\{\boldsymbol{s}^k\}_{k=1,2,\ldots,K}$. It is worth noting that in the ideal case, if $\boldsymbol{x}$ belongs to the k-th class, then

$\boldsymbol{s}^j = \mathbf{0} \quad \forall j = 1, \ldots, K, j \neq k$. Thus, the classification problem can be formulated as a l_0 sparse optimization problem [16], which is often relaxed to a linear programming problem

$$\hat{\boldsymbol{s}} = \arg\min_{\boldsymbol{s}} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}\|_2^2 + \lambda\|\boldsymbol{s}\|_1 \tag{3}$$

where $\|\boldsymbol{s}\|_1 = \sum_{i=1}^{J} |s_i|$ is the l_1-norm and λ is a scalar regularization parameter. Then, the class label of test sample $\boldsymbol{x}$ can be determined by the minimum residual [20] between $\boldsymbol{x}$ and its approximation from each class-wise subdictionary as follows:

$$\text{Class}(\boldsymbol{x}) = \min_{k=1,\ldots,K} \|\boldsymbol{x} - \boldsymbol{A}^k \hat{\boldsymbol{s}}^k\|_2 \tag{4}$$

where $\hat{\boldsymbol{s}}^k$ is the portion of the recovered sparse coefficients corresponding to subdictionary $\boldsymbol{A}^k$.

B. Optimized SRC Model Based on Spatial Correlation

In hyperspectral images, it is expected that neighboring pixels usually comprise materials with similar spectral characteristics, especially in homogeneous regions. Previous works indicate that by taking advantage of the spatial-contextual information, classification accuracy can be significantly improved [14], [32]–[35], [47], [49]. Some of these works exploit the spatial correlation via a preprocessing procedure [10], [32], [34], while others combine the spatial and spectral information in the classification stage [3], [4], [9], [35], and finally other methods include spatial information in a postprocessing procedure [8], [15], [33].

Let us assume that the hyperspectral image with $\boldsymbol{L}$ bands and $\boldsymbol{I}$ pixels is denoted by $\boldsymbol{X} \in \boldsymbol{R}^{L\times I}$, in which a pixel is represented by a column vector $\{\boldsymbol{X}_{:,i}\}_{i=1}^{I} \in \boldsymbol{R}^L$. In this context, the spatial correlation can be understood as a kind of inter-pixel smoothness [21]. Hence, a Tikhonov regularization [36] term can be introduced to characterize the spatial correlation as follows:

$$\min_{\boldsymbol{S}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{S}\|_F^2 + \lambda_1\|\boldsymbol{S}\|_{1,1} + \lambda_2\|\boldsymbol{H}\boldsymbol{S}\|_F^2 \tag{5}$$

where $\|\cdot\|_F$ is the Frobenius norm, $\|\cdot\|_{1,1}$ denotes the sum of absolute values of all the matrix elements, $\lambda_1 > 0$ and $\lambda_2 > 0$ are the regularization parameters, $\boldsymbol{S} \in \boldsymbol{R}^{J\times I}$ is the recovered sparse matrix whose row vectors $\{\boldsymbol{S}_{j,:}\}_{j=1}^{J} \in \boldsymbol{R}^I$ are the corresponding weights associated to the training samples (atoms) of the dictionary, and $\boldsymbol{H}\boldsymbol{S} = [(\nabla\boldsymbol{S}_{1,:})^T, (\nabla\boldsymbol{S}_{2,:})^T, \ldots, (\nabla\boldsymbol{S}_{J,:})^T]^T$, whose row column $\nabla\boldsymbol{S}_{j,:}$ denotes the discrete gradient of $\boldsymbol{S}_{j,:}$ in a two-dimensional sense.

Moreover, since the class labels and the spatial location of training samples are known (along with the corresponding sparse coefficient matrix of training samples), these aspects can be included in the SRC model to retain the spectral information of the training samples and spread them to neighboring pixels. Thus, the spectral fidelity term associated with the training samples is combined with a Tikhonov regularization term in order to increase the stability and accuracy of the classification by spreading the label information from each training sample to its neighbors until achieving a global stable state on the whole image. In summary, the optimized SRC model based on spatial correlation (SRCSC), which exploits both spatial and spectral information in the classification stage, can be formulated as follows:

$$\min_{\boldsymbol{S}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{S}\|_F^2 + \lambda_1\|\boldsymbol{S}\|_{1,1} + \lambda_2\|\boldsymbol{H}\boldsymbol{S}\|_F^2 + l_0(\boldsymbol{S}_\Lambda - \boldsymbol{I}) \tag{6}$$

where $l_0(x)$ is an indicative function (if $x = 0$, $l_0(x) = 0$; $x \neq 0$, $l_0(x) = \infty$), $\boldsymbol{S}_\Lambda$ denotes the coefficient matrix of training samples set $\boldsymbol{A}$ in $\boldsymbol{X}$ ($\boldsymbol{X}_\Lambda = \boldsymbol{A}$), and $\boldsymbol{I}$ is the identity matrix.

C. Serial Algorithm for SRCSC Based on Alternating Direction Method of Multipliers (ADMM)

In this section, we describe the solution of the optimization problem (6) using the alternating direction method of multipliers [37], [38]. First, (6) is rewritten as follows:

$$\min_{\boldsymbol{S},\boldsymbol{V}} g(\boldsymbol{S}, \boldsymbol{V}) \text{s.t. } \boldsymbol{G}\boldsymbol{S} + \boldsymbol{B}\boldsymbol{V} = 0 \tag{7}$$

where $\boldsymbol{V}_1 = \boldsymbol{S}, \boldsymbol{V}_2 = \boldsymbol{S}, \boldsymbol{V}_3 = \boldsymbol{H}\boldsymbol{V}_2, \boldsymbol{V} = [\boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3]$, and

$$g(\boldsymbol{S}, \boldsymbol{V}) = \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{S}\|_F^2 + \lambda_1\|\boldsymbol{V}_1\|_{1,1} + \lambda_2\|\boldsymbol{V}_3\|_F^2 + l_0(\boldsymbol{S}_\Lambda - \boldsymbol{I}), G = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I} \\ \mathbf{0} \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} -\boldsymbol{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{H} & -\boldsymbol{I} \end{bmatrix}.$$

The iterative format of the augmented Lagrangian method [48] can be written as

$$(\boldsymbol{S}^{(k+1)}, \boldsymbol{V}^{(k+1)}) = \arg\min_{\boldsymbol{S},\boldsymbol{V}} g(\boldsymbol{S}, \boldsymbol{V}) + \frac{\mu}{2}\|\boldsymbol{G}\boldsymbol{S} + \boldsymbol{B}\boldsymbol{V} - \boldsymbol{D}^{(k)}\|_F^2 \tag{8}$$

$$\boldsymbol{D}^{(k+1)} = \boldsymbol{D}^{(k)} - \boldsymbol{G}\boldsymbol{S}^{(k+1)} - \boldsymbol{B}\boldsymbol{V}^{(k+1)}. \tag{9}$$

This is a typical constrained optimization problem, which can be efficiently solved based on the ADMM. The solution of SRCSC based on ADMM is summarized in Algorithm 1.

Algorithm 1. Serial Algorithm of SRCSC Based on ADMM (SRCSC_S)

Input: Training samples set $\mathbf{A} \in \mathbf{R}^{L\times J}$, test samples set $\mathbf{X} \in \mathbf{R}^{L\times I}$

Initialization: Set $k = 0$, choose regularization parameters $\mu > 0, \lambda_1 > 0, \lambda_2 > 0, \varepsilon > 0$, initialize $S^{(0)}, V^{(0)}, D^{(0)}$

Do:

Step 1. Calculate $S^{(k+1)} \leftarrow \arg\min_S l(S, V^{(k)}, D^{(k)})$

$$S_{\bar{\Lambda}}^{(k+1)} = (A^T A + 2\mu I)^{-1}(A^T X + \mu(V_1^{(k)} + D_1^{(k)} + V_2^{(k)} + D_2^{(k)}))_{\bar{\Lambda}}$$

$$S_{\Lambda}^{(k+1)} = I$$

thereinto, $\bar{\Lambda}$ denotes the complementary set of Λ.

Step 2. Calculate

$$V_1^{(k+1)} \leftarrow \arg\min_{V_1} \lambda_1\|V_1\|_{1,1} + \frac{\mu}{2}\|S^{(k+1)} - V_1 - D_1^{(k)}\|_F^2$$

$$V_1^{(k+1)} = \text{soft}(D_1^{(k)} - S^{(k+1)}, \lambda_1/\mu)$$

thereinto, $\text{soft}(z, u) = \text{sign}(z)\max\{|z| - u, 0\}$

Step 3. Calculate

$$V_2^{(k+1)} \leftarrow \arg\min_{V_2} \frac{\mu}{2}\|S^{(k+1)} - V_2 - D_2^{(k)}\|_F^2 + \frac{\mu}{2}\|HV_2 - V_3^{(k)} - D_3^{(k)}\|_F^2$$

Step 4. Calculate

$$V_3^{(k+1)} \leftarrow \arg\min_{V_3} \lambda_2\|V_3\|_F^2 + \frac{\mu}{2}\|HV_2^{(k+1)} - V_3 - D_3^{(k)}\|_F^2$$

Step 5. Calculate $D^{(k+1)}=D^{(k)}-GS^{(k+1)}-BV^{(k+1)}$

Step 6. Calculate $k = k + 1$

While $\frac{\|S^{(k+1)}-S^{(k)}\|_F^2}{|S^{(k)}\|_F^2} > \varepsilon^2$

Step 7. $Class(\boldsymbol{x}) = \min_{k=1,\ldots,K} \|\boldsymbol{x} - \boldsymbol{A}^k\hat{\boldsymbol{s}}^k\|_2$, $\forall x, x$ is a column of $\boldsymbol{X}$.

Output: Class labels of $\boldsymbol{X}$.

End

In Steps 3 and 4 of Algorithm 1, the H operator independently transforms the matrix for each row; thus, the optimization problems of Steps 3 and 4 can be decomposed into several optimization subproblems, which are solved by the Gauss–Seidel [39], [40] method. In the following section, we provide an efficient parallel implementation of the method presented in Algorithm 1 on GPUs using CUDA.

III. Algorithm Optimization for SRCSC on GPUs

The particular structure of GPUs makes them suitable for accelerating highly intensive linear algebra calculations. According to the CUDA programming paradigm, for data-intensive computing problems like those involved in hyperspectral image processing, when executing a function (or kernel) on the device (GPU), one has to allocate memory on it, transfer data from the host (CPU) to the GPU, and finally transfer data back to the CPU, freeing the device memory. The actual kernel can be either manually defined or implemented by an optimized routine, like those offered by libraries such as CULA [41] developed by EM photonics in partnership with NVidia, and CUBLAS [51] provided by NVidia CUDA. As can be seen from Fig. 1, CULA achieves better performance than CUBLAS for matrix multiplication, especially for larger matrices. Thus, we choose CULA to realize the main matrix operations in this paper.

With the aforementioned issues in mind, to further optimize the SRCSC algorithm on GPUs, it is now necessary to deeply analyze Algorithm 1. The inputs to the algorithm are the hyperspectral data matrix $\mathbf{X}_{L\times I}$, the training samples matrix $\mathbf{A}_{L\times J}$, and the initial coefficient matrix $\mathbf{S}_{J\times I}^0$. There are serial loop executions, and the coefficient matrix $\mathbf{S}_{J\times I}^{(k+1)}$ needs to be updated in every loop. The calculations performed by the algorithm include matrix multiplication, matrix addition, matrix subtraction, matrix transposition, matrix inversion, and sum of matrix elements. For the higher dimensional data instances such as matrix $\mathbf{X}_{L\times I}$ and $\mathbf{S}_{J\times I}$, every read-write and arithmetic-type operation on the CPU is very time consuming, and the multistep iterative process increases the computational burden even more. Furthermore, we use Intel VTune [50] to analyze the execution performance of the serial algorithm using an AVIRIS

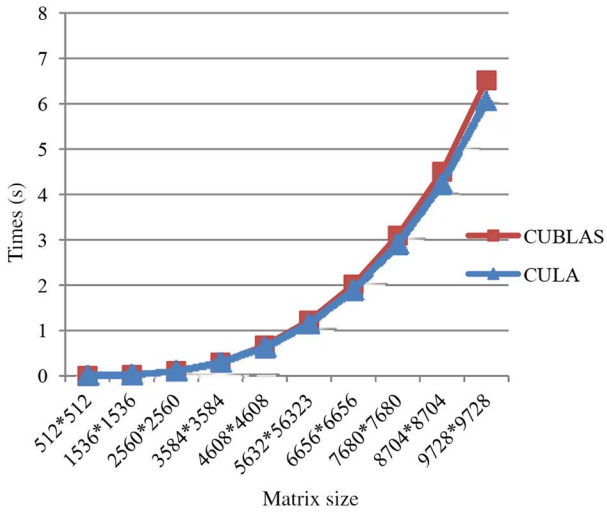


Fig. 1. Comparison between CUBLAS (CUDA5.5) and CULA (R17) when performing matrix multiplication.

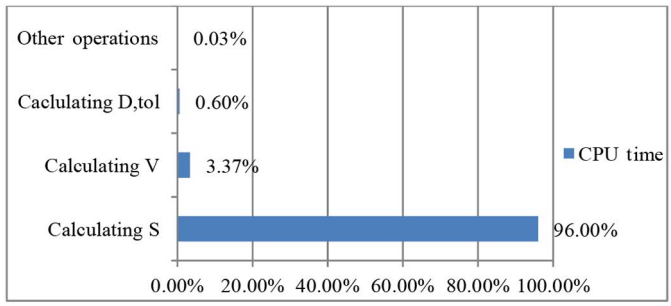


Fig. 2. Percentage of total CPU time consumed by different operations when processing a real hyperspectral image.

hyperspectral dataset with 1043 training samples (details of the dataset will be given in Section IV-A). As shown in Fig. 2, the calculation of $\boldsymbol{S}$ is the most time-consuming part of the SRCSC_S algorithm, in which the operations of matrix multiplication consume the most CPU time. The calculation of $\mathbf{S}_{J\times I}^{(k+1)}$ is a succession of transpositions, multiplications, and additions of big matrices. It is thus necessary to optimize the operations relevant to $\mathbf{X}_{L\times I}$, $\mathbf{S}_{J\times I}$ (and other elements with large data volume).

In order to improve the computing performance of SRCSC, a GPU-based parallel hyperspectral classification algorithm (SRCSC_P) has been developed as described in the flowchart given in Fig. 3. In the following, we describe the most relevant steps of the parallelization accomplished by the proposed SRCSC_P algorithm in Fig. 3.

A. Optimizing the Calculation of $S^{(k+1)}$

The terms $S^{(k+1)}$, $F = (A^TA + 2\mu I)^{-1}$, and $P = A^TX$ in Fig. 3 can be calculated in advance, since these elements remain unchanged in each loop. The operations of matrix multiplication and inversion are realized using functions `culaDeviceDgemm`, `culaDgetrf`, and `culaDgetri`, respectively. These functions are included in the high-efficient GPU-accelerated linear algebra libraries of CULA.

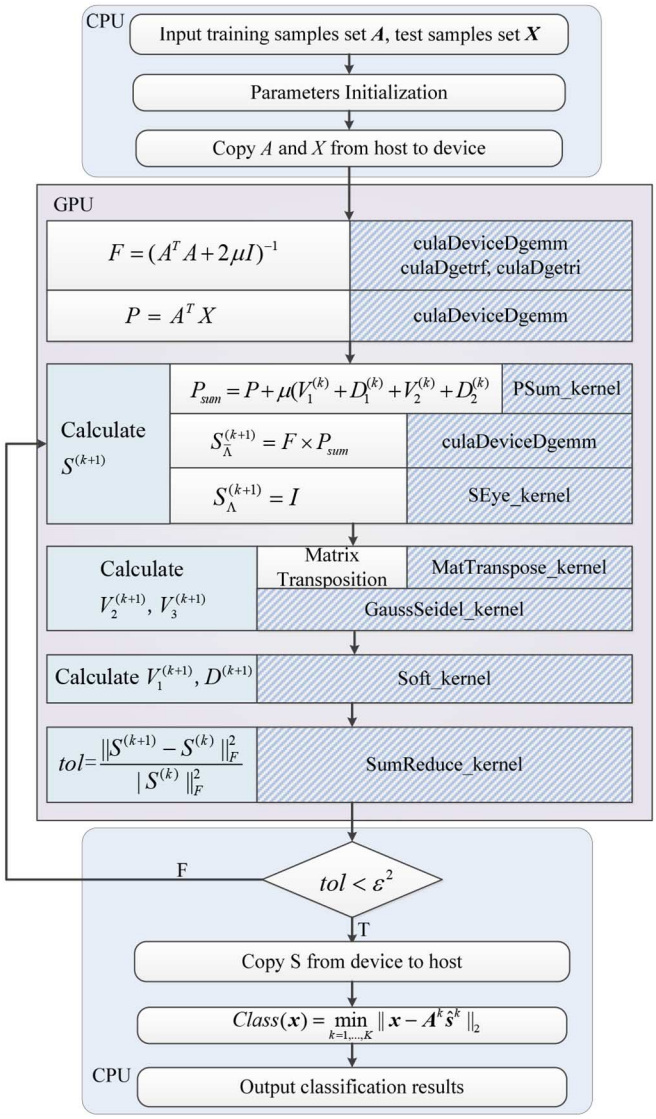


Fig. 3. Parallel hyperspectral classification algorithm based on SRCSC on GPUs (SRCSC_P).

The operation $P+\mu(V_1^{(k)}+D_1^{(k)}+V_2^{(k)}+D_2^{(k)})$ in Fig. 3 is the summation of matrices with same numbers of dimensions, and has been implemented by a kernel function called PSum_kernel (as shown in Fig. 4). The complexity of operation $S_\Lambda^{(k+1)}=F\times P_{\rm sum}$ in Fig. 3 is closely related to the number of training samples J, and is the most time-consuming operation in the serial Algorithm 1 (SRCSC_S). It is also optimized using function culaDeviceDgemm of CULA library in the parallel version SRCSC_P.

The kernel function SEye_kernel in Fig. 3 is implemented to do the operation of $S_\Lambda^{(k+1)}=I$ (as shown in Fig. 5). We predefine variable $idx_{1\times J}$ in the CPU to record the location of training samples, and then start a $J\times J$ thread grid on the GPU.

It is a crucial issue to decide the number of computing threads in every block (denoted as THREAD_SIZE in this paper) to be launched on the GPU and the number of blocks (denoted as BLOCK_SIZE in this paper) in every grid that allocate those threads. Normally, the device occupancy should be improved as high as possible, to make sure that there are more threads executing on every stream multiprocessor (SM). However, the number of threads concurrently executed on an SM is limited by the hardware capabilities (such as registers, shared memories, and so on). Furthermore, the high occupancy does not always lead to the best performance according to [52]. Therefore, the THREAD_SIZE should be chosen based on the practical resource requirements of kernel, and the BLOCK_SIZE varies along with the scale of problem. According to the computing capabilities of NVIDIA Tesla C2075 (the platform used in this paper, as shown in Table I), the THREAD_SIZE and BLOCK_SIZE variables, respectively, denoting the number of processsing *threads* per block and processing *blocks* per grid, are set to 32×32 and $((J+32-1)/32)\times((J+32-1)/32)$ in order to optimize the allocation of resources in the GPU.

```
__global__ void PSum_kernel (double* result, double* devB, double* devs,
            double* devd, double* devu, double* devb, double mu,size_t size)
{
    size_t idx = blockIdx.x*blockDim.x+threadIdx.x;
    if(idx<size)
    {
    result[idx]=devB[idx]+mu*(devs[idx]+devd[idx]+devu[idx]+devb[idx]);
    }
}
```

Fig. 4. CUDA code for Kernel PSum_kernel.

```
__global__ void SEye_kernel(double* dS, int ScolNum, int* xidx,
        int* xidxflag, int xidxSize, int* yidx, int* yidxflag, int yidxSize)
{
    int rowIdx=blockIdx.y;
    int colIdx=blockIdx.x;
    if(rowIdx<xidxSize&&colIdx<yidxSize)
    {
        if(xidx[rowIdx]==1&&yidx[colIdx]==1)
        {
            if(xidxflag[rowIdx]==yidxflag[colIdx])
                dS[rowIdx*ScolNum+colIdx]=1;
            else
                dS[rowIdx*ScolNum+colIdx]=0;
        }
    }
}
```

Fig. 5. CUDA code for Kernel SEye_kernel.

TABLE I
COMPUTING CAPABILITIES OF NVIDIA TESLA C2075

Threads/warp	32
Maximum warps/SM	48
Maximum threads/SM	1536
Maximum blocks/SM	8
32-bit registers/SM	32 768

B. Optimizing the Calculation of $V_1^{(k+1)}$, $V_2^{(k+1)}$, and $V_3^{(k+1)}$

Since the calculation of $V_1^{(k+1)}$ is independent of the calculation of $V_2^{(k+1)}$ and $V_3^{(k+1)}$, we compute $V_2^{(k+1)}$ and $V_3^{(k+1)}$ first. In the procedure of calculating $V_2^{(k+1)}$ and $V_3^{(k+1)}$, the H operator independently transforms the matrix for each row,

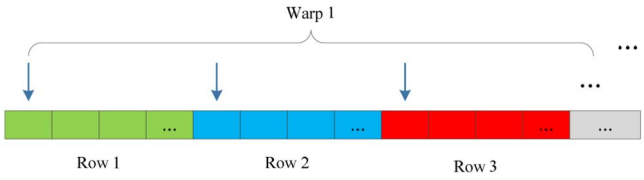


Fig. 6. Uncoalesced memory accesses.

Fig. 7. Coalesced memory accesses after matrix transposition.

```
__global__ void Soft_kernel(double* dS, double*ds, double* dd,
        double* du, double* db, int size, double tau){
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    if(idx<size){
        double sub=dS[idx]-db[idx]; double temp=0;
        if(sub>0)temp=1;
        else if(sub<0) temp=-1;
        else temp=0;
        du[idx]=temp*fmax(0,fabs(sub)-tau);
        db[idx]=db[idx]-(dS[idx]-du[idx]);
        dd[idx]=dd[idx]-(dS[idx]-ds[idx]);
    }
}
```

Fig. 8. CUDA code for Kernel `Soft_kernel`.

and the optimization problems are decomposed into several optimization subproblems. Since the transformations of rows are independent while solving these optimization subproblems using the Gauss–Seidel method, it is possible to optimize them per row in parallel. The kernel function `GaussSeidel_kernel` in Fig. 3 is defined to calculate $V_2^{(k+1)}$ and $V_3^{(k+1)}$, in which J threads are initiated and each thread transforms a row of the matrix. However, since the matrix is stored by rows, there are intervals in which adjacent threads in a warp access the shared memory in parallel, which leads to the problem of uncoalesced memory accesses depicted in Fig. 6. This can drastically influence the computing performance. To address this issue, we implement a kernel function `MatTranspose_kernel` to transpose the matrix in advance, so as to meet the requirement of coalesced memory accesses as illustrated in Fig. 7.

After that, for the calculation of $V_1^{(k+1)}$, we use a kernel `Soft_kernel` (as shown in Fig. 8) that creates $J \times I$ threads according to the matrix size, and each thread implements the computation of $\text{soft}(z,u) = \text{sign}(z)\max\{|z|-u, 0\}$ for a matrix element. The `THREAD_SIZE` and `BLOCK_SIZE` variables are set to 32×32 and $((J+32-1)/32) \times ((I+32-1)/32)$, respectively.

C. Optimizing the Calculation of $\frac{\|S^{(k+1)}-S^{(k)}\|_F^2}{|S^{(k)}\|_F^2}$

In this procedure, the main operation is the squared sum of matrix elements, which can be efficiently realized by a summation reduction model on the GPU as illustrated in Fig. 9. The kernel function `SumReduce_kernel` in Fig. 3 is implemented

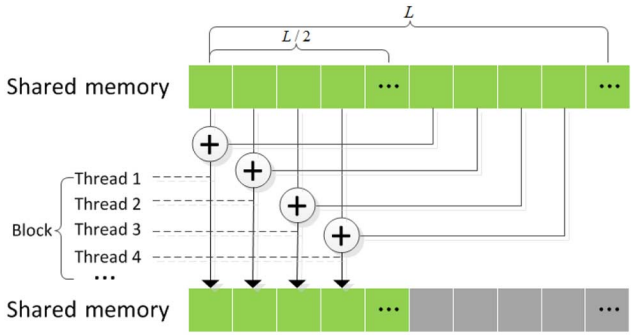


Fig. 9. Summation reduction model on the GPU.

to compute $\sum_i^J \sum_j^I (S_{i,j}^{(k+1)} - S_{i,j}^{(k)})^2$ and $\sum_i^J \sum_j^I (S_{i,j}^{(k)})^2$ simultaneously based on the summation reduction model depicted in Fig. 9, and then the results are copied from device (GPU) to the host (CPU) for analyzing the loop termination condition and subsequent operations.

To sum up, we provide a detailed step-by-step algorithm description of the parallel hyperspectral classification algorithm based on SRCSC on GPUs in Algorithm 2.

Algorithm 2. Parallel Hyperspectral Classification Algorithm Based on SRCSC on GPUs (SRCSC_P)

Input: Training samples set $\mathbf{A} \in \mathbf{R}^{L \times J}$, test samples set $\mathbf{X} \in \mathbf{R}^{L \times I}$
Initialization: Set $k = 0$, choose regularization parameters $\mu > 0, \lambda_1 > 0, \lambda_2 > 0, \varepsilon > 0$, initialize $S^{(0)}, V^{(0)}, D^{(0)}$
Step 1. Copy data from host to device
Step 2. Invoke `culaDgetrf`, `culaDgetri` and `culaDeviceDgemm` to calculate $F = (A^T A + 2\mu I)^{-1}$ and $P = A^T X$ on GPU
Do
  Step 3. Calculate $S^{(k+1)}$ on GPU
    Invoke kernel function `PSum_kernel` to compute $P_{sum} = P + \mu(V_1^{(k)} + D_1^{(k)} + V_2^{(k)} + D_2^{(k)})$
    Invoke `culaDegmm` to compute $S_\Lambda^{(k+1)} = F \times P_{sum}$
    Invoke kernel function `SEye_kernel` to compute $S_\Lambda^{(k+1)} = I$
  Step 4. Invoke `GaussSeidel_Kernel` to calculate $V_2^{(k+1)}, V_3^{(k+1)}$ on GPU
  Step 5. Invoke `Soft_kernel` to calculate $V_1^{(k+1)}, D^{(k+1)}$ on GPU
  Step 6. Invoke `SumReduce_kernel` to calculate $tol = \frac{\|S^{(k+1)}-S^{(k)}\|_F^2}{|S^{(k)}\|_F^2}$ on GPU
  Step 7. Copy tol from device to host
  Step 8. $k \leftarrow k+1$
While $tol > \varepsilon^2$
Step 9. Copy $S^{(k)}$ from device to host
Step 10. Calculate $Class(\boldsymbol{x}) = \min_{k=1,\ldots,K} \|\boldsymbol{x} - \boldsymbol{A}^k \hat{\boldsymbol{s}}^k\|_2$ ($\forall x, x$ is a column of $\boldsymbol{X}$) on CPU
Output: Class labels of $\boldsymbol{X}$.
End

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

Scenes	Indian Pines region of Northwestern Indiana, USA	Urban area of Pavia University, Italy
Instruments	AVIRIS (airborne visible/infrared imaging spectrometer)	ROSIS (reflective optics system imaging spectrometer)
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.10(b)]	9 [Details are shown in Fig.11(b)]
Labeled pixels	10 366	43 923

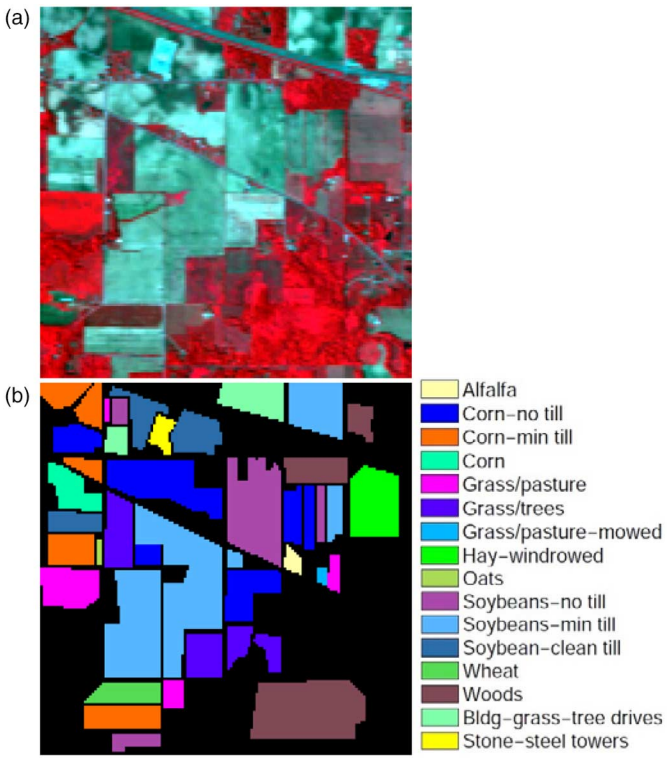


Fig. 10. AVIRIS Indian Pines hyperspectral image. (a) False color composition. (b) Ground truth as a collection of mutually exclusive classes.

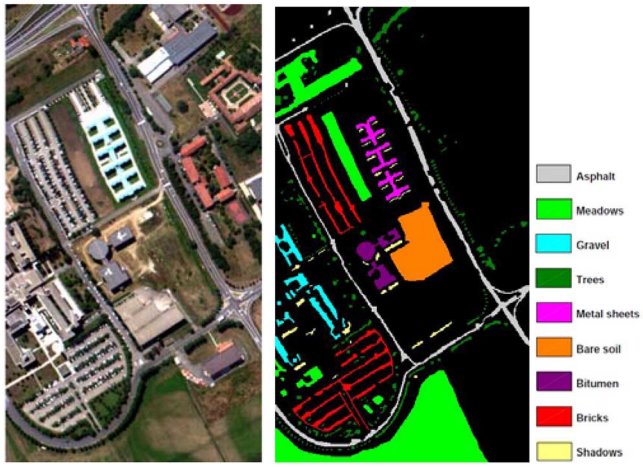


Fig. 11. ROSIS Pavia University hyperspectral image. (a) False color composition. (b) Ground truth as a collection of mutually exclusive classes.

IV. EXPERIMENTAL RESULTS

A. Experimental Platform and Hyperspectral Images

The proposed classification algorithm is assessed using two widely used real hyperspectral images collected by two different imaging spectrometers (AVIRIS Indian Pines and ROSIS Pavia University). The details of these images are given in Table II and Figs. 10 and 11.

The experimental platform used to evaluate the proposed algorithm is the NVIDIA Tesla C2075, which features 448 processor cores operating at clock frequency of 1150 MHz, with single precision floating point performance of 1030 Gflops, double precision floating point performance of 515 Gflops, total dedicated memory of 6 GB, 1500 MHz memory (with 384-bit GDDR5 interface), and memory bandwidth of 144 GB/s. The GPU is connected to two Intel Xeon E5-2609CPUs at 2.4 GHz with 4 cores (8 cores in total) with 32-GB RAM. All the serial and parallel versions of the SRCSC algorithm are implemented using the $C++$ programming language on Microsoft Visual Studio 2010 integrated development environment, and the compiler is Visual $C++$ 2010, which supports the OpenMP 2.0 standard. CUDA 5.5, CULA R17, and Intel MKL 11.2 are used in the corresponding implementations. The tests were performed using the 64b Microsoft Windows 7 operating system.

B. Classification Accuracy Assessment

To illustrate their classification accuracy, both SRCSC_S and SRCSC_P were executed on the two considered images. According to [20] and our repeated experiments, the parameters were empirically set to be $\lambda_1 = 10^{-3}$, $\lambda_2 = 2$, $\varepsilon = 0.0005$, and $\mu = 1$. The classification results were then compared visually and quantitatively to those obtained by the SVM [42], SVM based on composite kernels (SVM-CK) [12], and l_1 sparse representation classification (SRC- l_1) [19] also solved by ADMM [37]. The SVM-CK is a kind of spectral–spatial joint classification method using mean filter and polynomial kernel, which has been proved to be a powerful tool to solve supervised classification problems, generally exhibiting better performance than spectral-based SVMs. Both SVM and SVM-CK are implemented using the LIBSVM library (available online: http://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html) [43], using the radial basis function (RBF) kernel $K(x_i, x_j) = \exp(-\|x_i - x_j\|_2^2/2\sigma^2)$ and one-against-one classification strategy. The kernel parameters and regularization parameters are determined by fivefold cross-validation, according to [12] and [43]. All the quantitative measurements reported in our experiments correspond to the average of results after 10 Monte Carlo runs.

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

Class	Training samples	Test samples	SVM	SVM-CK	SVM-l_1	SRCSC_S	SRCSC_P
Alfalfa	6	48	50.00	90.00	29.58	**96.25**	**96.25**
Corn-notill	144	1290	75.05	93.26	59.94	**99.01**	**99.01**
Corn-min	84	750	65.36	96.29	48.05	**96.94**	**96.94**
Corn	24	210	51.24	93.43	35.81	**98.95**	**98.95**
Grass/Pasture	50	447	91.41	96.74	84.25	**98.30**	**98.30**
Grass/Trees	75	672	93.93	98.21	94.49	**99.79**	**99.79**
Grass/Pasture-mowed	3	23	81.74	**87.82**	71.30	86.09	86.09
Hay-windrowed	49	440	96.64	99.37	98.05	**100.00**	**100.00**
Oats	2	18	38.89	**88.89**	25.56	55.56	55.56
Soybeans-notill	97	871	71.25	89.83	62.99	**95.78**	**95.78**
Soybeans-min	247	2221	82.91	96.21	83.18	**99.73**	**99.73**
Soybean-clean	62	552	71.38	94.75	40.94	**97.93**	**97.93**
Wheat	22	190	95.79	**98.84**	94.74	98.53	98.53
Woods	130	1164	96.79	99.05	97.23	**99.95**	**99.95**
Building-Grass-Trees-Drives	38	342	52.81	92.63	37.43	98.19	**98.19**
Stone-steel Towers	10	85	87.53	97.41	93.41	**99.29**	**99.29**
OA (%)	1043	9323	80.45	95.60	73.24	**98.67**	**98.67**
AA (%)			75.17	94.55	66.05	**95.01**	**95.01**
κ			0.7765	0.9499	0.6915	**0.9848**	**0.9848**

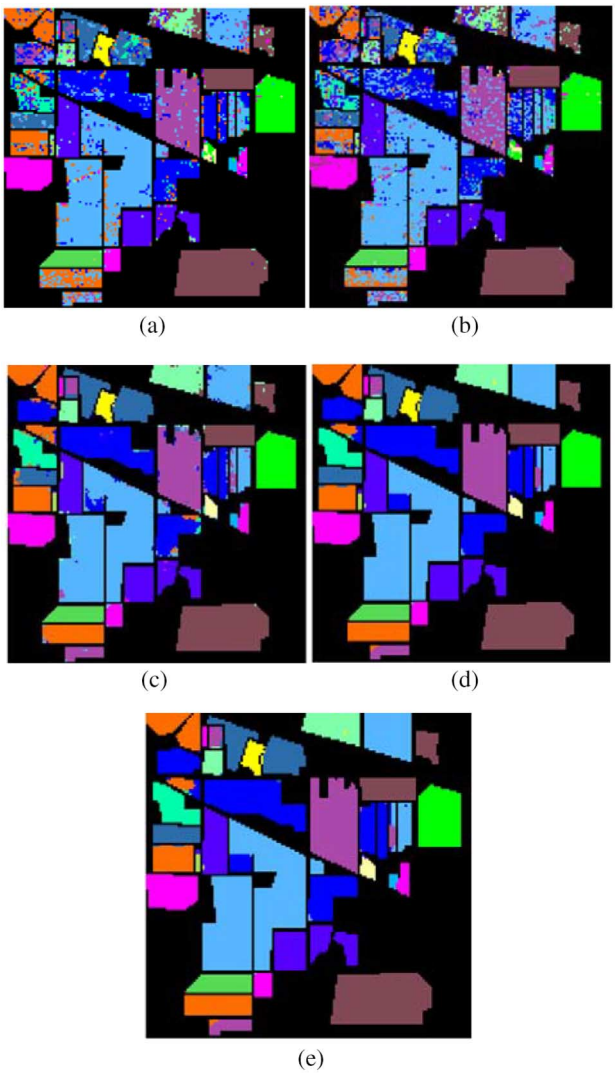


Fig. 12. Some of the classification maps obtained after 10 Monte Carlo runs for the AVIRIS Indian Pines dataset. (a) SVM. (b) SRC-l_1. (c) SVM-CK. (d) SRCSC_S. (e) SRCSC_P.

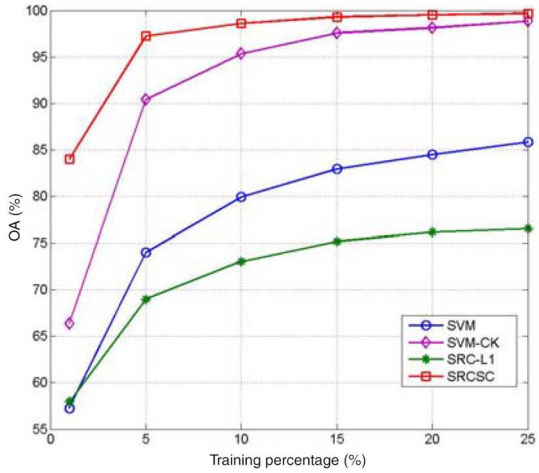


Fig. 13. OA obtained using different training percentages on the AVIRIS Indian Pines dataset.

Before reporting the obtained classification accuracy results, we first describe the metrics used for quantitative comparison in our experiments. To define the metrics in mathematical terms, let us assume that a confusion matrix with K classes is denoted by $\boldsymbol{M}$, in which the matrix element M_{ij} denotes the sample amount of the jth class that is classified as the ith class. Then, the overall accuracy (OA), average accuracy (AA), and the kappa (κ) statistic [44] can be used to measure the classification accuracy. The OA is computed by the ratio between correctly classified test samples and the total number of test samples

$$\mathrm{OA} = \sum_{i=1}^{K} M_{ii}/N.$$

The AA is the mean of all class accuracies

$$\mathrm{AA} = \frac{1}{K}\sum_{i=1}^{K}(M_{ii}/N_i).$$

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

Class	Training samples	Test samples	SVM	SVM-CK	SVM-l_1	SRCSC_S	SRCSC_P
Asphalt	40	6812	77.60	**84.50**	77.03	70.80	70.80
Meadows	40	18646	74.80	90.75	65.13	**94.73**	**94.73**
Gravel	40	2167	62.21	84.68	66.22	**99.63**	**99.63**
Trees	40	3396	92.61	94.70	94.61	**98.94**	**98.94**
Metal sheets	40	1338	99.40	**100.00**	99.70	**100.00**	**100.00**
Bare soil	40	5064	80.49	**97.22**	73.95	95.12	95.12
Bitumen	40	1316	92.10	89.29	88.91	**99.85**	**99.85**
Bricks	40	3838	90.05	85.02	67.95	**91.35**	**91.35**
Shadows	40	986	**99.29**	94.42	94.22	78.60	78.60
OA (%)	360	43563	79.84	90.35	73.06	**91.26**	**91.26**
AA (%)			85.39	91.18	80.86	92.11	92.11
κ			0.7449	0.8752	0.6642	**0.8866**	**0.8866**

The κ statistic is 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 of the degree of agreement

$$\kappa = \left(N \left(\sum_{i=1}^{K} M_{ii} \right) - \sum_{i=1}^{K} \left(\sum_{j=1}^{K} M_{ij} \sum_{j=1}^{K} M_{ji} \right) \right) \Bigg/ \left(N^2 - \sum_{i=1}^{K} \left(\sum_{j=1}^{K} M_{ij} \sum_{j=1}^{K} M_{ji} \right) \right).$$

Our first experiment was carried out using the AVIRIS Indian Pines dataset. Nearly 10% of the labeled pixels of each class (1043 pixels in total) were randomly chosen as training samples, and the remaining labeled pixels were used as test samples, as shown in Table III. The classification accuracies (OA, AA, κ) are quantitatively shown in Table III, and graphically illustrated in Fig. 12. Table III indicates that the proposed SRCSC_S and SRCSC_P obtain exactly the same classification accuracy and outperform the results obtained by other methods, leading to the smoothest classification maps as depicted in Fig. 12.

In a second experiment, we assess the impact of the number of training samples on the classification accuracy. For this purpose, we consider different numbers of training samples, ranging from 1% to 25% of each class. As shown in Fig. 13, the classification accuracy can be greatly increased by incorporating the spatial correlation, and the SRCSC method (both in serial and parallel versions) outperforms SVM-CK at most instances.

A third experiment is conducted on the ROSIS Pavia University dataset. In this experiment, 40 labeled pixels were randomly chosen from each class as training samples, and the remaining labeled pixels are used as test samples. Table IV shows the obtained classification accuracies, while Fig. 14 shows some of the obtained classification maps after the 10 Monte Carlo runs. The results indicate that the SRCSC methods obtain very competitive results. Here, it is also worth mentioning that the serial (SRCSC-S) and the parallel GPU (SRCSC_P) versions obtain exactly the same results in terms of classification accuracy. The two versions can be, therefore, considered exactly equivalent in terms of classification accuracy and only different in terms of computational performance. In the following section, we analyze the computational performance of the two versions by particularly focusing on the improvements of SRCSC_P with regards to SRCSC_S.

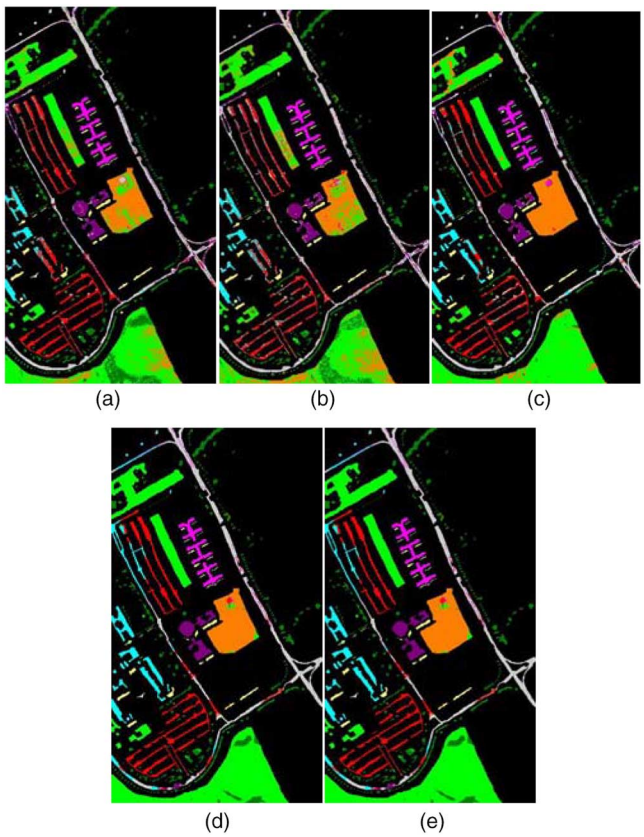


Fig. 14. Some of the classification maps obtained after 10 Monte Carlo runs for ROSIS Pavia University dataset. (a) SVM. (b) SRC-l_1. (c) SVM-CK. (d) SRCSC_S. (e) SRCSC_P.

C. Parallel Performance Assessment

An experimental evaluation of the computational performance of our proposed parallel optimization is reported in this section, using the platform described in Section IV-A. In order to show the performance improvements between the

TABLE V
EXECUTION TIME OF THE SERIAL, MULTICORE AND GPU VERSIONS WITH THE AVIRIS INDIAN PINES IMAGE

Dataset	SRCSC_S	SRCSC_S_MKL		SRCSC_M		SRCSC_P					
						Total		Data transfer from host to device		Data transfer from device to host	
	Time (s)	Time (s)	Speedup (X)	Time (s)	Speedup (X)	Time (s)	Speedup (X)	Time (s)	Percentage (%)	Time (s)	Percentage (%)
IndianPines (1% training)	157.93	74.76	2.11	37.95	4.16	9.86	16.02	0.016	0.16	0.011	0.11
IndianPines (5% training)	1632.47	352.38	4.63	123.17	13.25	20.21	80.76	0.018	0.09	0.051	0.25
IndianPines (10% training)	5186.01	800.49	6.48	226.19	22.93	36.38	142.54	0.018	0.05	0.097	0.27
IndianPines (15% training)	10763.05	1339.67	8.03	333.39	32.28	57.38	187.57	0.020	0.03	0.147	0.26
IndianPines (20% training)	17399.70	2001.56	8.69	448.79	38.77	83.66	207.97	0.020	0.02	0.217	0.26
IndianPines (25% training)	27203.50	2750.97	9.89	562.77	48.34	114.83	236.91	0.013	0.01	0.208	0.18
PaviaUniversity	28962.30	6549.36	4.42	2769.27	10.46	427.79	67.70	0.071	0.02	0.525	0.12

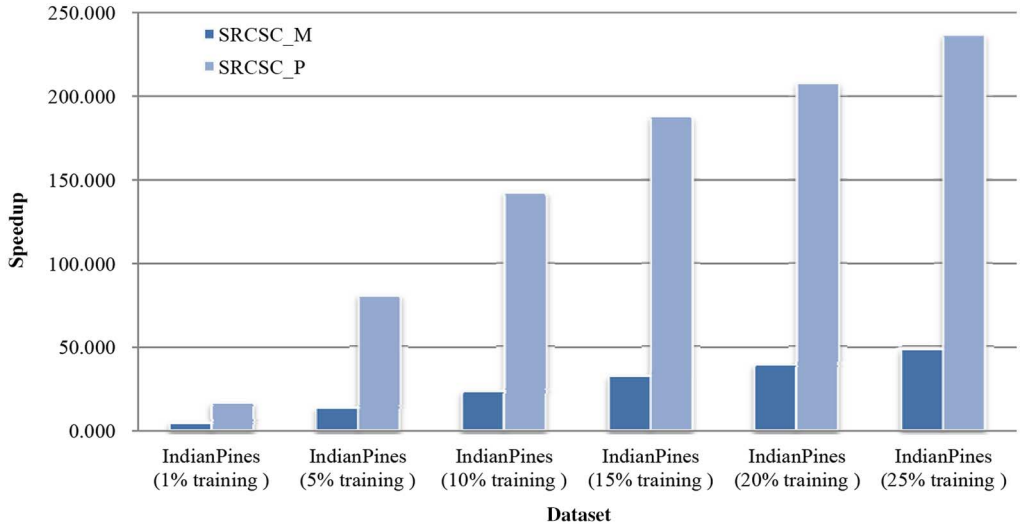


Fig. 15. Speedup comparison between SRCSC_P and SRCSC_M using different number of training samples on the AVIRIS Indian Pines image.

parallel optimizations on multicore CPU platform and our considered GPU platform, a multicore implementation of SRCSC (SRCSC_M) has been carried out following the design principles in [45] and using OpenMP API and the MKL (Intel Math Kernel Library, a library of optimized math routines provided by Intel) [46]. First, OpenMP is adopted to explicitly address multithreaded and shared-memory parallelism, in which we specify the regions (in this case, mainly the codes corresponding to the kernels in GPU parallel version) suitable for parallel implementation. Second, for the time-consuming operations of the big matrix calculations, such as multiplication and inversion, two levels of parallelism are used combining OpenMP with BLAS. The API OpenMP is used to set the first parallelism level, and the routines of `dgetrf`, `dgetri`, and `cblas_dgemm` (multithreading implementations of BLAS in MKL library) are invoked to realize the calculations of matrix inversion and multiplication for the second parallelism level. Last but not the least, the `parallel for` directives are used to instruct the compiler to execute the structured block of code in parallel on multiple cores. In addition, in order to distinguish the effects of MKL and OpenMP respectively, another version (SRCSC_S_MKL) is implemented by simply using the routines of `dgetrf`, `dgetri`, and `cblas_dgemm` in MKL library.

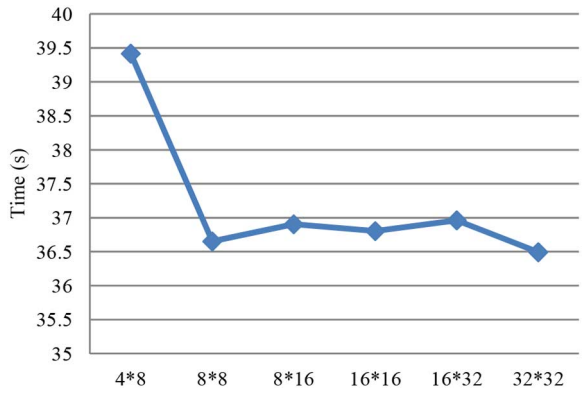


Fig. 16. Comparison of different `THREAD_SIZE` configurations.

The corresponding serial versions are executed on one core of the Intel Xeon E5-2609 CPU, and the multicore version is run on eight cores of the two Intel Xeon E5-2609. For each test,

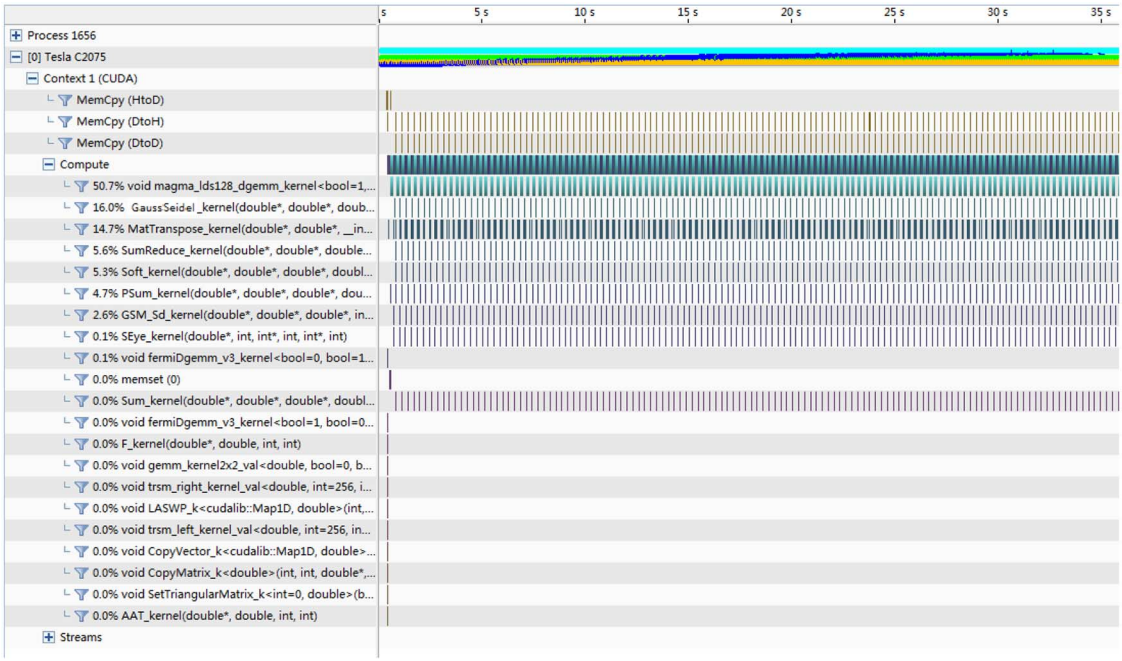


Fig. 17. Percentage of time consumed by different operations in our parallel implementation (SRCSC_P).

TABLE VI
PERFORMANCE TESTS OF DIFFERENT OPERATIONS ON AVIRIS INDIAN PINES DATASET WITH 1043 TRAINING SAMPLES

Kernel	Invocations	Importance (%)	Occupancy (%)		Global memory bandwidth (GB/s)		Shared memory bandwidth (GB/s)	
			Achieved	Theoretical	Reads	Writes	Loads	Stores
magma_lds128_dgemm_kernel	133	50.7	33.3	33.3	39.9	2.69	604.82	56.68
GaussSeidel_kernel	133	16	33.3	33.3	25.04	5.23	0	0
MatTranspose_kernel	533	14.7	66.5	66.7	36.73	25.69	286.74	18.15
SumReduce_kernel	133	5.6	66.6	66.7	29.81	59.81	52.1	48.55
Soft_kernel	133	5.3	66.6	66.7	69.16	42.35	0	0
Psum_kernel	133	4.7	66.6	66.7	100.39	16.41	0	0
GSM_Sd_kernel	133	2.6	66.6	66.7	83.05	31.17	0	0
SEye_kernel	133	0.1	66.6	66.7	51.51	41.09	0	0
fermiDgemm_v3_kernel	3	0.1	33.2	33.3	40.08	12.77	576.33	74.95
Others	–	0.2	–	–	–	–	–	–

10 Monte Carlo runs were performed and the mean values are reported in our experiments.

At this point, it is important to emphasize that both the GPU and multicore parallel versions obtain exactly the same results as the serial implementation. This means that the three versions obtain exactly the same classification accuracy with the considered scenes and the only difference among them is the computing performance. Table V reports the obtained results in terms of computation times and speedups measured after comparing the parallel implementations of SRCSC (SRCSC_M and SRCSC_P) with the equivalent serial version for the two considered hyperspectral images. A speedup comparison between SRCSC_P and SRCSC_M when using the AVIRIS Indian Pines dataset with different numbers of training samples is also graphically illustrated in Fig. 15.

It can be concluded from Figs. 13 and 15 that the classification accuracy improved significantly as the number of training samples is increased, but so does the computational efficiency of the parallel implementations. It is, therefore, crucial to find a balance between accuracy and computing performance. Fortunately, the speedup of SRCSC_P increases significantly with number of training samples. Thus, more accurate results can be generally obtained with the same computation time.

On the other hand, the impact of I/O communications between the host and the device is also an important issue, as this often becomes the main bottleneck of parallel systems. Our experiments indicate that the time required for data transfers between the host and the device is relatively low. Specifically, it represents less than 0.35% of the total execution time of the GPU parallel implementation in all cases.

In order to verify the THREAD_SIZE configuration of our GPU parallel implementation, several THREAD_SIZE configurations were tested, namely $4 \times 8, 8 \times 8, 8 \times 16, 16 \times 16, 16 \times 32,$ and 32×32 (all of them are integral multiple of

the size of warp) on the AVIRIS Indian Pines dataset with 10% training set, and the execution times are comparatively reported in Fig. 16. As can be seen from Fig. 16, the `THREAD_SIZE` of 32×32 leads to the best performance in our GPU version.

To further analyze the bottleneck of the GPU parallel algorithm, nvprof and visual profiler [53] are utilized to explore the execution performance of different operations in the proposed algorithm with the AVIRIS Indian Pines dataset of 1043 training samples. As it can be seen from Fig. 17 and Table VI, the proposed parallel algorithm is compute-bound, and the most significant portion of the time taken by our parallel implementation is the pure computation steps, which is ideal in terms of parallel efficiency. Since `magma_lds128_dgemm_kernel` (the kernel in `culaDeviceDgemm` function) takes more than half of the execution time, it can be concluded that the bottleneck of the GPU parallel implementation mainly lies in the operation of matrix multiplication.

To conclude this section, we emphasize that (as shown in Table V and Fig. 15) the average speedup of the GPU parallel version (SRCSC_P) is more than $130\times$ with regard to the serial version SRCSC_S, while the multicore version SRCSC_M gets an average speedup of $24\times$. For the AVIRIS Indian Pines scene with less than 15% training samples, the classification task can be completed in about 1 min. These are important advantages offered by GPU platforms in terms of being able to adapt computationally expensive classification problems such as those involved in SRC methods to time-critical hyperspectral image analysis scenarios.

V. Conclusion and Future Lines

Sparse representation has been shown to be an extremely powerful tool for hyperspectral image classification. However, it is hard to be adapted to time-critical scenarios due to its high computational complexity. In this paper, we propose a novel parallel sparse representation classification method on GPUs. First, an optimized sparse representation model is introduced based on spatial correlation regularization and a spectral fidelity term, which increases classification accuracy by spreading the label information from each training sample to its neighbors until achieving a global stable state on the whole image. Further, we develop a parallel implementation of the SRC approach for GPUs using the NVidia CUDA. To the best of our knowledge, this is the first GPU-based optimization algorithm for sparse representation classification reported in the literature in the context of remotely sensed hyperspectral imaging. Experimental results on real hyperspectral datasets, which include comparisons to equivalent serial and multicore versions, demonstrate the efficiency of the proposed parallel method, which is shown to be able to provide accurate results in high speed, and is appealing for efficient hyperspectral data processing. Although the reported processing times of the algorithm are still far from real-time performance, the method could be significantly improved by resorting to more recent GPU platforms including multi-GPU environments. Our future work will focus on adapting the presented algorithm to multi-GPU platforms, which are still subject to I/O overheads due to the fast performance of the GPU as compared to the communication network. We will also conduct a detailed investigation of the energy consumption of GPU platforms when executing our newly developed algorithms, as it is an important parameter for onboard processing.

Acknowledgment

The authors gratefully thank Prof. P. Gamba and Prof. D. Landgrebe for providing the ROSIS and AVIRIS data sets used in this study for evaluation purposes. The authors also gratefully acknowledge the Associate Editor and the anonymous reviewers for their detailed comments and suggestions, which greatly helped to improve the technical quality and presentation of this paper.

References

[1] 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.

[2] A. Plaza *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, no. Suppl. 1, pp. S110–S122, Sep. 2009.

[3] S. Li *et al.*, "Hyperspectral imagery clustering with neighborhood constraints," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 3, pp. 588–592, Mar. 2013.

[4] 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.

[5] Y. Gu *et al.*, "Representative multiple kernel learning for classification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 7, pp. 2852–2865, Jul. 2012.

[6] L. Gomez-Chova, G. Camps-Valls, L. Bruzzone, and J. Calpe-Maravilla, "Mean map kernel methods for semisupervised cloud classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 48, no. 1, pp. 207–220, Jan. 2010.

[7] Y. Zhong and L. Zhang, "An adaptive artificial immune network for supervised classification of multi-/hyperspectral remote sensing imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 3, pp. 894–909, Mar. 2012.

[8] 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.

[9] Y. Chen, N. M. Nasrabadi, and T. D. Tran, "Hyperspectral image classification via kernel sparse representation," *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 1, pp. 217–231, Jan. 2013.

[10] 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.

[11] G. Camps-Valls, L. Gomez-Chova, J. Munoz-Mari, J. Vila-Frances, and J. Calpe-Maravilla, "Composite kernels for hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 3, no. 1, pp. 93–97, Jan. 2006.

[12] M. Marconcini, G. Camps-Valls, and L. Bruzzone, "A composite semisupervised svm for classification of hyperspectral images," *IEEE Geosci. Remote Sens.*, vol. 6, no. 2, pp. 234–238, Feb. 2009.

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

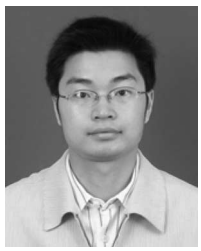
[14] Y. Tarabalka, M. Fauvel, J. Chanussot, and J. A. Benediktsson, "SVM-and MRF-based method for accurate classification of hyperspectral images," *IEEE Geosci. Remote Sens. Lett.*, vol. 7, no. 4, pp. 736–740, Apr. 2010.

[15] 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.

[16] J. Wright *et al.*, "Sparse representation for computer vision and pattern recognition," *Proc. IEEE*, vol. 98, no. 6, pp. 1031–1044, Jun. 2010.

[17] E. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 489–509, Feb. 2006.

[18] D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.
[19] Y. Chen, N. M. Nasrabadi, and T. D. Tran, "Hyperspectral image classification using dictionary-based sparse representation," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 10, pp. 3973–3985, Oct. 2011.
[20] 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.
[21] J. Liu, Z. Wu, Z. Wei, L. Xiao, and L. Sun, "Spatial correlation constrained sparse representation for hyperspectral image classification," *J. Elect. Inf. Technol.*, vol. 34, no. 11, pp. 2666–2671, Oct. 2012.
[22] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb. 2009.
[23] 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.
[24] 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.
[25] A. Plaza, "Special issue on architectures and techniques for real-time processing of remotely sensed images," *J. Real Time Image Process.*, vol. 4, pp. 191–193, 2009.
[26] C. A. Lee *et al.*, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.
[27] 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.
[28] S. Bernabe *et al.*, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, Mar. 2013.
[29] S. Bernabe, S. Lopez, 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] X. Wu, B. Huang, A. Plaza, Y. Li, and C. Wu, "Real-time implementation of the pixel purity index algorithm for endmember identification on GPUs," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 5, pp. 955–959, May 2013.
[31] 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.
[32] 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, Apr. 2010.
[33] X. Kang, S. Li, and J. Benediktsson, "Spectral-spatial hyperspectral image classification with edge-preserving filtering," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 5, pp. 2666–2677, May 2014.
[34] 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.
[35] R. Ji *et al.*, "Spectral-spatial constraint hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 3, pp. 1811–1824, Mar. 2014.
[36] Y. W. Wen, M. K. Ng, and W. K. Ching, "Iterative algorithms based on decoupling of deblurring and denoising for image restoration," *SIAM J. Sci. Comput.*, vol. 30, no. 5, pp. 2655–2674, May 2009.
[37] J. M. Bioucas-Dias and M. A. T. Figueiredo, "Alternating direction algorithms for constrained sparse regression: Application to hyperspectral unmixing," in *Proc. 2nd Workshop WHISPERS*, 2010, pp. 1–4.
[38] W. Deng, W. Yin, and Y. Zhang, "Group sparse optimization by alternating direction method," Dept. Comput. Appl. Math., Rice Univ., Houston, TX, Tech. Rep. TR11-06, 2011.
[39] 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, Oct. 2012.
[40] T. Goldstein and S. Osher, "The split Bregman method for L1-regularized problems," *SIAM J. Imag. Sci.*, vol. 2, no. 2, pp. 323–343, Feb. 2009.
[41] EM Photonics. (2014, Jun.). *CULA Programmer's Guide* [Online]. Available: http://www.culatools.com/cula_dense_programmers_guide/
[42] 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.
[43] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intel. Syst. Technol.*, vol. 2, no. 3, pp. 2701–2727, Mar. 2011.
[44] 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.
[45] S. Bernabe *et al.*, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, Mar. 2013.
[46] Intel Developer Zone. (2014, Nov.). *Reference Manual for Intel® Math Kernel Library 11.2* [Online]. Available: https://software.intel.com/en-us/mkl_11.2_ref
[47] Y. Tarabalka, J. A. Benediktsson, and J. Chanussot. "Spectra-spatial classification of hyperspectral imagery based on partitional clustering techniques," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 8, pp. 2973–2987, Aug. 2009.
[48] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Berlin, Germany: Springer-Verlag, 2006.
[49] L. Zhang *et al.*, "Kernel sparse representation based classifier," *IEEE Trans. Signal Process.*, vol. 60, no. 4, pp. 1684–1695, Apr. 2012.
[50] Intel Developer Zone. (2014, Nov.). *Intel® VTune™ Amplifier 2015* [Online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe
[51] NVIDIA Developer Zone. (2014, Jun.). *cuBLAS User Guide* [Online]. Available: http://docs.nvidia.com/cuda/cublas/index.html
[52] V. Volkov. (2014, Nov.). *Better Performance at Lower Occupancy* [Online]. Available: http://www.nvidia.com/content/GTC-2010/pdfs/2238_GTC2010.pdf
[53] NVIDIA Developer Zone. (2014, Nov.). *Profiler User's Guide* [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/#axzz3K7S7Wk7G



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

He is currently an Associate Professor with the School of Computer Science and Engineering, NUST, and also a Visiting Scholar at the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Caceres, Spain. His research interests include hyperspectral image processing, high-performance computing, and computer simulation.

Qicong Wang was born in Henan, China, in 1988. He received the B.Sc. degree from the School of Computer, Henan University of Science and Technology, Luoyang, China, in 2012. He is currently pursuing the M.S. degree at 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.

Antonio Plaza (M'05–SM'07–F'15) was born in Caceres, Spain, in 1975.

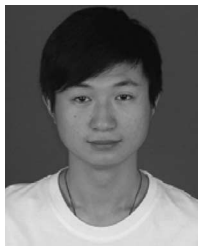
He is an Associate Professor (with accreditation for Full Professor) with the Department of Technology of Computers and Communications, University of Extremadura, Badajoz, Spain, where he is the Head of the Hyperspectral Computing Laboratory (HyperComp). He has been the advisor of 12 Ph.D. dissertations and more than 30 M.Sc. dissertations. He was the Coordinator of the Hyperspectral Imaging Network, a European project with total funding of 2.8 million Euro. He has authored more than 400 publications, including 130 JCR journal papers (82 in IEEE journals), 20 book chapters, and over 240 peer-reviewed conference proceeding papers (94 in IEEE conferences). He has edited a book on High-Performance Computing in Remote Sensing (CRC Press/Taylor & Francis) (the first book on this topic in the published literature) and guest edited eight special issues on hyperspectral remote sensing for different journals. His research interests include remotely sensed hyperspectral image analysis and efficient implementations of large-scale scientific problems on high-performance computing architectures.

Dr. Plaza served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS) in 2011–2012, and is currently serving as President of the Spanish Chapter of IEEE GRSS (since November 2012). He has served as a proposal evaluator for the European Commission (Marie Curie Actions, Engineering Panel), the European Space Agency, the Belgium Science Policy, the Israel Science Foundation, and the Spanish Ministry of Science and Innovation. He has participated in the Tenure Track Selection Committee of different Universities in Italy, Spain and Australia. He has reviewed more than 500 articles for over 50 different journals. He is also currently serving as the Editor-in-Chief of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING. He is a recipient of the recognition of Best Reviewers of the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS (in 2009) and the recognition of Best Reviewers of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING (in 2010), a journal for which he served as Associate Editor in 2007–2012. He is also an Associate Editor for IEEE ACCESS, and was a member of the Editorial Board of the IEEE GEOSCIENCE AND REMOTE SENSING NEWSLETTER (2011–2012) and the IEEE GEOSCIENCE AND REMOTE SENSING MAGAZINE (2013). He was also a member of the steering committee of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He is currently an Associate Editor for the *Journal of Real-Time Image Processing*. He is a recipient of the 2013 Best Paper Award of the JSTARS journal, and a recipient of the most highly cited paper (2005–2010) in the *Journal of Parallel and Distributed Computing*. He is a co-author of the 2011 Best Student Paper at the IEEE International Conference on Space Technology, and a recipient of the 2008 Best Paper award at the IEEE Symposium on Signal Processing and Information Technology. He is a recipient of the Best Ph.D. Dissertation award at the University of Extremadura in 2002.

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 (SIGNAL). 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. Her research interests include hyperspectral image classification and segmentation, spectral unmixing, signal processing, and remote sensing.

Dr. Li 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. She is the recipient of the 2012 Best Reviewer Award of the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.



Jianjun Liu was born in Jiangsu, China, in 1987. He received the B.Sc. degree in applied mathematics from Nanjing University of Science and Technology (NUST), Nanjing, China, in 2009, where he is currently pursuing the Ph.D. degree in computer science.

His research interests include spectral unmixing, hyperspectral image 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 from South East University, Nanjing, China, in 1983, 1986, and 2003, respectively.

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