Multi-GPU Implementation of Nearest-Regularized Subspace Classifier for Hyperspectral Image Classification

Zhixin Li, Jun Ni, *Student Member, IEEE*, Fan Zhang, *Senior Member, IEEE*, Wei Li, *Senior Member, IEEE*, and Yongsheng Zhou, *Member, IEEE*

***Abstract*—The classification of hyperspectral imagery (HSI) is an important part of HSI applications. The nearest-regularized subspace (NRS) is an effective method to classify HSI as one of the sparse representation methods. However, its high computational complexity confines usage in a time-critical scene. In order to enhance the computation efficiency of the NRS classifier, this article proposed a new parallel implementation on the graphics processing unit (GPU). First of all, an optimized single-GPU algorithm is designed for parallel computing, and then the multi-GPU version is developed to improve the efficiency of parallel computing. In addition, optimal parameters for the data stream and memory strategy are put forward to adapt a parallel environment. In order to verify the algorithm's effectiveness, the serial algorithm based on central processing unit is used for a comparative experiment. The performance of the multi-GPU approach is tested by two hyperspectral image datasets. Compared with the serial algorithm, the multi-GPU method with four GPUs can achieve up to 360× acceleration.**

***Index Terms*—Graphics processing unit (GPU), high-performance computing (HPC), hyperspectral imagery (HSI), image classification, nearest-regularized subspace (NRS).**

I. Introduction

OVER the last decade, hyperspectral imagery (HSI) has been garnering growing attention in the remote sensing field [1]–[3]. Hyperspectral imaging system acquires hundreds of images at corresponding continuous narrow spectral channels [4]–[6], which provide rich discriminative spectral information. HSI has been widely applied in many scene interpretation and sensing communities, such as agricultural monitoring [7], target recognition [8], large-scale urban or agriculture mapping [9], etc. Compared with the low-dimensional remote sensing imagery (i.e., traditional optical imagery and synthetic aperture radar imagery [10]), the high-dimensional nature of HSI hinders the transfer of traditional imagery processing algorithms to HSI classification tasks. As a result, the classification of HSI has become a hot topic in remote sensing imagery interpretation field [11].

Recently, some representation-based classifiers achieve excellent results in land cover images [12], [13]. The pixel label can be determined by the category of the closest representation based on the principle that a pixel can be represented as a linear combination of the labeled samples. Compared with existing deep learning methods [14]–[18], the representation-based classifiers do not require a large number of training samples to obtain better classification results, and complicated parameter tuning. The essence of the sparse representation-based classification (SRC) method is to represent a known pixel as the linear combination of labeled samples via l_1-based sparsity-inducing regularization [19]. Although achieving satisfied classification results, SRC encounters the problems of high computational cost and higher accuracy improvement. Therefore, a collaborative-representation classifier is developed by replacing l_1-norm-based regularization with l_2-norm to avoid the algorithmic complexity [20], [21]. Considering the drawback of representation-based classifiers (i.e., SRC, and CRC) in dealing with spectral diversity of intraclass samples, the nearest-regularized-subspace (NRS) classifier is proposed by coupling nearest-subspace learning with the distance-weighted Tikhonov regularization [22], [23]. Through the distance-weighted Tikhonov regularization, the NRS classifier can dynamically learn the similarity between the test sample and each training sample in one dictionary to obtain a more accurate prediction vector. When applied in high-dimensional signal classification, NRS tends to be chosen as the optimal classifier as it can achieve tradeoff performance in classification accuracy and computation efficiency, compared with SRC and CRC, as well as support vector machine (SVM) [22], [24], [25]. However, introducing NRS to HSI classification is still limited because HSI usually boasts of a large number of pixels in real scenarios. Matrix computation and linear solutions of NRS for HSI classification on CPU are time-consuming and storage-cost.

With the rapid development of hyperspectral imaging techniques, spatial and spectral resolutions of HSI are getting more and more finer. The finer the resolution is, the large the data

Manuscript received March 26, 2020; revised June 6, 2020; accepted June 15, 2020. Date of publication June 22, 2020; date of current version July 2, 2020. This work was supported by the National Natural Science Foundation of China under Grant 61871413 and Grant 61801015. *(Zhixin Li and Jun Ni contributed equally to this work.) (Corresponding author: Fan Zhang.)*

Zhixin Li, Jun Ni, Fan Zhang, and Yongsheng Zhou are with the College of Information Science & Technology, Beijing University of Chemical Technology, Beijing 100029, China (e-mail: 88371447@qq.com; nijunbuct@163.com; zhangf@mail.buct.edu.cn; zhyosh@mail.buct.edu.cn).

Wei Li is with the School of Information and Electronics, Beijing Institute Technology, Beijing 100081, China (e-mail: liwei089@ieee.org).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/JSTARS.2020.3004064

volume is. Hence, it is necessary to develop real-time classification methods. Although cloud computing can speed up the algorithms' running time [26], [27], it heavily relies on computer hardware resources, resulting in increased hardware costs and a waste of resources. In terms of one device, the performance of algorithms in the hyperspectral image meets more challenges because of the high computational complexity of the classification, as mentioned in earlier models. All of these methods are time-consuming and limited their application in real-time scenarios.

Benefiting from the improvement of computer performance, different complex computing methods can be optimized by a variety of parallel computing approaches. Compared with the algorithm-level acceleration through precision-for-efficiency, high-performance computing (HPC) technologies are capable of improving efficiency while maintaining accuracy. In terms of parallel computing devices, the HPC technologies can be divided into two categories, namely the central processing unit (CPU) parallel and graphics processing unit (GPU) parallel. In the early remote sensing processing oriented parallel accelerations, the CPU parallel technologies dominated the primary studies, e.g., open multiple processing (OpenMP) with multicores, message passing interface (MPI) with multi-CPUs, and grid computing with multi-computers [28], [29]. With the development of GPU hardware, the general-purpose computation on GPU (GPGPU) technology has drawn increasing attention in remote sensing [30]–[35]. GPU has become a universal computing platform for many data-intensive and computing-intensive scenarios. Compared to traditional CPU parallelism, it can achieve a dozen to several hundred times of acceleration.

In this regard, GPU has been increasingly used as a commodity platform for many compute-intensive, massively parallel, and data-intensive computations [36], [37]. GPU-based parallel computing offers a tremendous potential to bridge the gap toward a real-time analysis of hyperspectral images [38]. In recent developments, a few parallel implementations of the HSI classification based on GPUs have been introduced to enhance calculation efficiency. For instance, SVM with composite kernel [39], SRC [38], the combination of SRC and Markov random fields (MRF) [40], and spatiality adaptive MRFs [41]. Although their calculation efficiency has been significantly improved, the classification accuracy of these methods is inferior to NRS [22]. Despite the importance of the NRS algorithm in HSI classification, there are no available GPU implementations for it in the public literature.

In this article, a novel parallel NRS algorithm is proposed for hyperspectral image classification on multi-GPUs. First, an optimized NRS serial method is proposed, which uses memory to store matrix multiplication results to avoid the repeated large matrix multiplication task. Subsequently, based on this optimized serial algorithm, a single-GPU method is proposed to speed up the calculation of complex matrices. Although the single GPU can optimize the computation of large matrix in the NRS algorithm, it cannot simultaneously solve the processing problem of batch samples. An optimized multi-GPU method is designed for batch processing of samples after comparing different task allocation and scheduling strategies. By using NVIDIA's Compute Device Unified Architecture (CUDA) and NVIDIAs Tesla K 10 GPU, our methods can reach up to $50\times$ speedup in Pavia University dataset and $360\times$ speedup in Pavia Centre dataset. The main contributions of our work are as follows.

1) Based on the serial NRS model, an optimized NRS model is introduced. By using this model, a single-GPU algorithm is proposed to realize the initial parallel acceleration.
2) The single-GPU method is extended to a multi-GPU method by considering the multiple task partition and scheduling.
3) An optimal multi-GPU approach is proposed by comparing different strategies of task partition and scheduling.

The rest of this article are organized as follows. Section II presents the basic procedure and sequential analysis of the NRS algorithm. Section III describes the single-GPU and multi-GPU parallel classification algorithms. Section IV conducts some experiments on processing performance and result accuracy that use real datasets. Eventually, Section V draws conclusions.

II. Related Work

A. NRS Classifier

By integrating nearest-subspace learning with Euclidean distance-weighted Tikhonov regularization, NRS predicts the label of the target test pixel. An approximation for each test sample is generated through a linear combination of all training samples of each class. In this way, the approximation of each test sample is created separately from the training samples of each class. Then, the closest class representation will label the corresponding test data. NRS and SRC both get the approximation of the test sample via linear combinations of training samples. But NRS not only does this by using a noncollaborative approach to the approximation but also makes use of nonuniform regularization. In addition, NRS uses Tikhonov regularization to generate $\tilde{y}_l$, but NRS decides bias atoms of X_l by their Euclidean distance from the test sample y.

The symbols used are as follows:

1) C: number of classes;
2) l: a class label, $l \in \{1, 2, \ldots, C\}$;
3) N_t: spectrum number of hyperspectral image;
4) y: a test sample (vector);
5) X_l: the training samples of class l;
6) P_l: the quantity of training sample in X_l;
7) $x_{l,p}$: a test sample (vector) of class l, and p is the number of training sample X_l, $p \in \{1, 2, \ldots, P_l\}$;
8) α_l: the coefficients of class l;
9) $\tilde{y}_l$: class-specific approximation for a text sample y in class l;
10) $\|\cdot\|_2$: the l_2 Euclidean distance.

In the NRS method, $\tilde{y}_l$ represents the class-specific approximation between the dictionary X_l of lth category and the test sample y, which is calculated via a linear combination of available training samples in each class, X_l and α_l represents the weighting factor. The per-class coefficients are first calculated according to the l_2-norm regularization

$$\alpha_l = \arg\min_{\alpha_l^*} \|y - X_l\alpha_l^*\|_2^2 + \lambda\|\Gamma_{l,y}\alpha_l^*\|_2^2 \tag{1}$$

where λ is a global regularization parameter to balance the minimization between the regularization terms and residual, $\Gamma_{l,y}$ is the biasing Tikhonov diagonal matrix to each training sample $x_{l,p}$ of class l and test sample y

$$\Gamma_{l,y} = \begin{bmatrix} ||y - x_{l,1}||_2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & ||y - x_{l,P_l}||_2 \end{bmatrix}. \quad (2)$$

A closed-form solution of the weighting vector α_l can be directly calculated as

$$\alpha_l = (X_l^T X_l + \lambda \Gamma_{l,y}^T \Gamma_{l,y})^{-1} X_l^T y. \quad (3)$$

After the weighting vector is obtained, the label of the test pixel is determined according to the category that minimizes the Euclidean distance between y and $\tilde{y_l}$

$$r_l(y) = \|\tilde{y_l} - y\|_2^2 = \|X_l \alpha_l - y\|_2^2. \quad (4)$$

According to the aforementioned formulas, the approximate $\tilde{y_l}$ of each test sample y can be solved as

$$\tilde{y_l} = X_l (X_l^T X_l + \lambda \Gamma_{l,y}^T \Gamma_{l,y})^{-1} X_l^T y. \quad (5)$$

Then, the label of test sample y is calculated by

$$\text{class}(y) = \arg \min_{l=1 \ldots C} r_l(y). \quad (6)$$

Although the CRC is similar to the NRS classifier, the latter uses the regularization term instead of an identity matrix I of CRC, and the experimental results have been illustrated that NRS outperforms CRC in hyperspectral image classification [22]. By using a biasing Tikhonov matrix, the samples X_l, which is different from y in the Euclidean distance, should contribute less to the linear combination. As a result, the more relevant weighting vector α_l has better similarity with the test sample, which results in better classification performance than CRC.

B. Serial Algorithm Analysis

Although the bias of NRS provides enormous benefits and makes it receive extensive attention in HSI classification, its computationally expensive implementations limit the practical application.

Supposed that Λ_Range is the range of regularization parameter λ, $Testdata$ is the test dataset, and X_l is the training sample set of lth class. According to the analysis in Section II-A, the serial method is shown in Algorithm 1.

In Algorithm 1, $\Gamma_{l,y}$ is evaluated by function $normKernel(\cdot)$, corresponding to (2), $geagKernel(\cdot)$ is used to calculate $X_l^T X_l + \lambda \Gamma_{l,y}^T \Gamma_{l,y}$ in (3), and $norm1Kernel(\cdot)$ is preformed to calculate the residual between the predicted vector $\tilde{y_l}$ and the test sample y, expressed by Euclidean distance in (4). $minKernel(\cdot)$ is implemented to realize classification in (6). Therefore, the NRS algorithm includes the following five steps in common.

1) Input regularization parameter λ, one test sample $y_{[1\times N_t]}$ and one labeled training samples $X_{l[S_l \times N_t]}$ of class l.
2) Calculate the distance $D_{n[S_l \times S_l]}$ of $y_{[1\times N_t]}$ to each training sample $x_{[1\times N_t]}$.

Algorithm 1: NRS Classifier: Serial Version.

Require: DataTest: Test samples, Λ_Range: range of λ, Training dataset
Ensure: Result: classified result

```
for each λ ∈ Λ_Range do
  for each y ∈ Testdata do
    for each l ∈ [1, C] do
      get X_l the training samples of l-th category;
      norms = normKernel(y, X_l);
      X_sq = X_l^T * X_l;
      Temp = geagKernel(X_sq, norms, λ);
      α_l = Temp^{-1} * X_l^T · y;
      ỹ_l = X_l * α_l;
      res(l) = norm1Kernel(ỹ_l, y);
    end for
    Result = minKernel(res);
  end for
end for
```

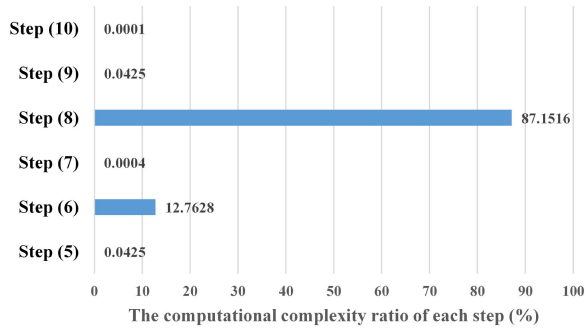


Fig. 1. Histogram of computational complexity for the six steps.

3) Generate the diagonal matrix $G_{[S_l \times S_l]}$ according to λ and diagonal of D_n.
4) Obtain weight $W_{[S_l \times S_l]}$ by solving matrix.
5) Go back to Step (1), and do the iterations.

Generally, λ is considered to be a variable to find an ideal regularization parameter, despite that it is a constant in an actual classification task. Therefore, there are three for-loops in the NRS algorithm: each λ, each test sample, and each training dictionary of different categories. Supposed L is the number of regularization parameter λ, T is the number of test samples, C is the number of categories, and S_l is the number of training samples X_l of lth category. Because there are matrix multiplication and inversion in (5) corresponding to the Step (4), the matrix complexity is $M = O(\max(S_l, N_t)^3)$. The complexity of whole serial algorithm is $O(L, T, C, M)$.

In detail, under a given regularization parameter λ, the core process of a test sample in lth dictionary can be divided into six steps: Step (5), Step (6), Step (7), Step (8), Step (9), and Step (10). Suppose that the number of training samples in the lth dictionary is 300, and each sample has 103 features. The computational complexity of each step is shown in Fig. 1. From Fig. 1, the computational resource is mainly consumed in Step (8), which is the linear solution and multiplication of the matrix,

Algorithm 2: NRS Classifier: The Optimized Serial Version.

Require: DataTest: Test samples, Λ_Range: range of λ, Training dataset
Ensure: Result: classified result
1: **for** each $l \in [1, C]$ **do**
2: $X_{sq}(l) = {X_l}^T * X_l$;
3: **end for**
4: **for** each $y \in Testdata$ **do**
5: **for** each $l \in [1, C]$ **do**
6: get X_l the training samples of l-th category;
7: $norms = normKernel(y, X_l)$;
8: $XY = {X_l}^T * y$;
9: **for** each $\lambda \in \Lambda\_Range$ **do**
10: $Temp = geagKernel(X_{sq}(l), norms, \lambda)$;
11: $\alpha_l = Temp^{-1} * XY$;
12: $\tilde{y_l} = X_l * \alpha_l$;
13: $res(y, l, \lambda) = norm1Kernel(\tilde{y_l}, y)$;
14: **end for**
15: **end for**
16: **end for**
17: $Result(y, \lambda) = minKernel(res)$;

accounting for about 87.15% of the computational resources. The matrix multiplication in Step (5) accounts for 12.76% of the computing resources. Other processes take up about 0.09% of the computing resources.

The relationship between different for-loops is independent between each test sample and λ when the matrix multiplication is performed in the equation $X_{\text{sq}} = {X_l}^T * X_l$ in Algorithm 1, so it can be calculated before the classification is performed. Moreover, if the formula res of each λ and each category is stored in memory instead of executing $Result = minKernel(res)$ immediately in Algorithm 1, the relationships between three for-loops will be independent, so that Algorithm 1 can be optimized to Algorithm 2. In Algorithm 2, memory is used to save calculation resources in Step (2), Step (8), and Step (17). Equation (3) is regarded as $\alpha_l = (X_{sq}(l) + \lambda * norms)^{-1} * XY$, and res is considered as a 3-D matrix. Besides, to speed up the efficiency of data-parallel computing, $XY = X_l^T \cdot y$ is first calculated in the algorithm and saved in memory XY. Therefore, the complexity of the whole serial algorithm is realized without the three for-loops, which will help implement parallel optimization algorithms, e.g., OMP, MPI, and GPU.

III. Multi-GPU Parallel Classification Algorithm

In this section, the aforementioned NRS flow is adapted to fit the hardware architecture and memory layout of GPU. NVIDIA CUDA, based on GPU, is used as the parallel platform to achieve the NRS algorithm. A single-GPU method will be discussed first because it is the foundation to fulfill the multi-GPU algorithm.

Matrix calculation is the most time-consuming operation in NRS. First, some functions of matrix operation in our program are interpreted, and a third party library is introduced to accelerate these operations. In the single-GPU method, some GPU kernel functions are implemented to deal with some special processes of the matrix. Then, this algorithm is extended from single-GPU to multi-GPU, namely to divide tasks into some parts for each GPU, and the task partition and scheduling are performed in multi-GPU. Finally, some detailed strategies are designed to enhance the efficiency of memory reading and copying efficiency.

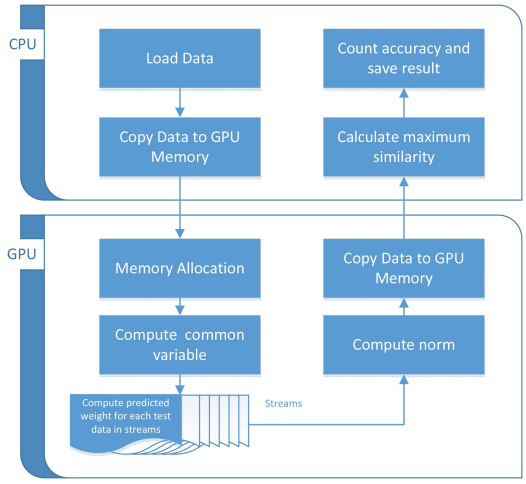


Fig. 2. Overall flowchart of GPU-based NRS classification.

A. CUDA-Based GPGPU Application

NVIDIA CUDA technology has powerful parallel processing capability because of its massive multi-processors and easy-to-control instruction architecture, i.e., single-instruction multiple-data, which allows us to execute same operation on different data. Each multiprocessor maintains its local variables and shares memory with other multiprocessors. When GPU dealing with a parallel task, data should copy from host memory to device memory since GPU could only access display memory via a allocated device memory pointer.

The architecture of GPU is suited to data-intensive linear algebra calculations. The GPU-based NRS classifier runs a great number of threads in one GPU for parallel computation. In order to utilize each parallel thread and grid, the algorithm needs to consider data independence in each grid, because data exchange in different grids cannot be exchanged. Therefore, the NRS-GPUs flows can be designed as Fig. 2.

In Fig. 2, the HSI is first converted to binary files to compress their size and save the loading time. Second, the training data and test data will be copied to device memory from host memory. And then, computationally intensive tasks (i.e., memory allocation, common variable computation, weights computation, norm computation, and data copy) will be performed on the GPU, such as Step (4) to Step (15) in Algorithm 2. After the GPU calculation, the results will be copied from device memory on the GPU to host memory. The classification results will be calculated by CPU on the host. Of course, GPUs information and initiate CUDA devices should be obtained before a new GPU function is designed, so as to maximize the usage of GPU computing resources.

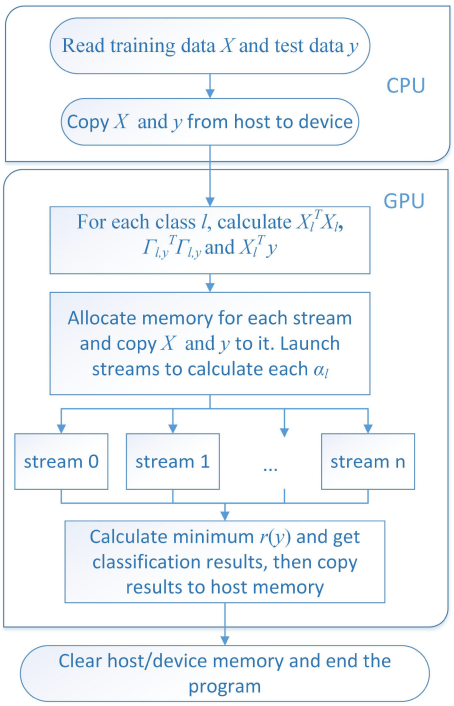


Fig. 3. Specific implementation of GPU-based NRS classification.

Algorithm 3: NRS Classifier: Single-GPU Version.

Require: DataTest: Test samples, Λ_Range: range of λ, Training dataset
Ensure: Result: classified result

```
Copy data from the host memory to device memory;
for each l ∈ [1, C] do
  get X_l the training samples of l-th category;
  normsAll = normsGPU(Testdata, X_l);
  XY = X_l * DataTest;
  X_sq = X_l^T · X_l;
  for each λ ∈ [0, Λ_Range] do
    for each i ∈ [1, T] do
      calculate y in Testdata, i is the index, stream
      is used for each test sample y;
      Temp = geagGPU(X_sq, normsAll[i], λ);
      α_l[i] = posv(Temp, XY[i]);
    end for
    ỹ_l = X_l * α_l;
    res = norm1GPU(ỹ_l, Testdata);
  end for
end for
Result(y, λ) = minKernel(res);
```

B. Single-GPU-Based NRS Algorithm

GPU has powerful computing capabilities, so it should be assigned as many computing tasks as possible when processing computation-intensive tasks. However, it is hard to provide data for GPUs computation since that data need to be loaded and copied not only from host to the device but also from device memory to GPU. The key strategy is to reduce memory access and memory usage. The order of memory access speed is GPU register, GPU shared memory, GPU constant memory, GPU display memory, and from any host memory.

According to the previous description, the overall flowchart can be designed and shown in Fig. 3. The GPU-NRS algorithm is divided into two parts: (1) parallel task partition and scheduling, and (2) specific matrix operation designing.

1) Parallel Task Partition and Scheduling: As the mentioned sequential algorithm in Algorithm 2, the calculation in different test samples can be generated to one matrix operation, such as, $XY = {X_l}^T * y$ with a size of $P_l \times 1$ is replaced by $XY = {X_l}^T * Testdata$ with a size of $P_l \times T$, and $normsAll = normsKernel(y, X_l)$ is replaced by $normsAll = normsGPU(Testdata, X_l)$. As a result, more kernels of GPU and more device memory can be used. Therefore, the NRS algorithm can be implemented with one GPU, as shown in Algorithm 3.

Especially, in some devices of computing capability $2\times$ and higher, $stream$ allows devices perform an asynchronous memory copy to or from the GPU concurrently with kernel execution. In Algorithm 3, $stream$ is used to resolve the parallel time consumption of data computation algorithm and data copy at Step (8) to Step (12).

2) Matrix Operation in GPU: Based on the NRS method, some GPU functions are designed to resolve a series of matrix operation problems, and they are defined as follows.

1) $normsGPU()$: Calculate the biasing Tikhonov matrix in (2) for test samples by GPU.
2) $gemm()$: Compute a matrix–matrix product with general matrices.
3) $geagGPU()$: Calculate the results about $X_l^T X_l + \lambda \Gamma_{l,y}^T \Gamma_{l,y}$ in (3) for ith test sample by GPU.
4) $posv()$: Compute the solution to the system of linear equations $A * X = B$, where A is an symmetric positive-definite coefficient matrix, and B is used to store solution matrix X.
5) $norms1GPU()$: Computes the residuals between the predicted vectors and all test samples by GPU.

Those five global kernel functions are designed with CUDA, and they are defined with the prefix "__global__". In addition, NVIDIA supplies some basic libraries in CUDA such as BLAS (i.e., Basic Linear Algebra Subprograms). BLAS is a library that contains basic linear algebra functions. In order to enhance the efficiency of such matrix functions, a third-party library, namely CULA, is used in our implementations [42]. CULA is a set of GPU-accelerated linear algebra libraries utilizing the NVIDIA CUDA parallel computing architecture to dramatically improve the computation speed of sophisticated mathematics. It provides not only BLAS functions, but also LAPACK (Linear Algebra PACKage) functions such as general matrix multiplication $gemm()$ and positive matrix equation solution $posv()$.

C. NRS Algorithm Based on Multi-GPU

Although the single-GPU method has been developed in Section III-B, it is just used to deal with some matrix operations

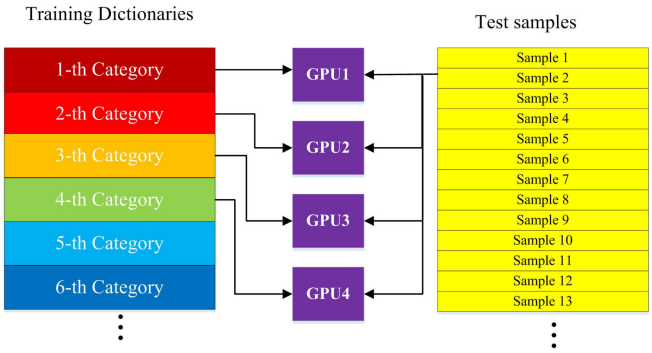


Fig. 4. Parallel task scheduling of Multi-GPU v1.

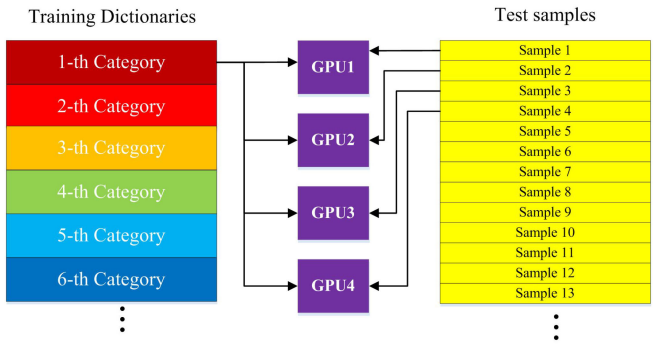


Fig. 5. Parallel task scheduling of Multi-GPU v2.

and limited parallel task scheduling. As for the three for-loops mentioned in Section II-B, a lot of parallel limitations have not been resolved, in spite of some computations are implemented in parallel, e.g., $normsGPU()$, $geagGPU()$, $norms1GPU()$, and $stream$. The reason is that most of the resources are used to accelerate matrix operations, so that the single GPU cannot offer more resources for these for-loops. In addition, these for-loops are nested, and single GPU cannot accelerate two or more nested parallel problems at the same time. Therefore, it is necessary to develop a multi-GPU version of the NRS algorithm, so as to achieve real-time processing of a large number of test samples.

Based on single GPU-NRS, a multi-GPU version of the NRS method, namely MGPU-NRS, is used to deal with one of the for-loops. For controlling each GPU at the same time, multithread is a common technique. In MGPU-NRS, some macros are inserted into the single-GPU version, and OpenMP is used to control GPUs. CUDA toolkit offers a function to bound a thread with a device. At first, the number of created threads is the same as the number of GPUs. Then, the same memory allocation will be done. Data will be copy from host memory according to thread ID (start from 0). As a result, different GPUs are used to deal with a classification task in one of the for-loops. Finally, two versions of MGPU-NRS methods are developed according to different needs, named Multi-GPU v1 and Multi-GPU v2, respectively.

1) Multi-GPU v1: In the first task division method, multiple GPUs are used to accelerate the for-loop about the training dictionary of each category. The training dictionary of each category is assigned to a different GPU to calculate residual for one test sample, and the residuals of all categories will be copied to the host. The advantage of this version is that the category attributes of a test sample can be calculated immediately. The parallel task scheduling is shown in Fig. 4.

As shown in Fig. 4, four GPUs are used to accelerate the NRS algorithm. The training dictionaries are divided into several parts according to their labels and assigned to the specified GPU, respectively. Test samples are copied to each GPU to calculate residuals.

2) Multi-GPU v2: In case that the number of each category of training sample may be uneven, the corresponding GPU of the category with many training samples will slow down the computing speed of other GPUs (waits for the slower GPU), thus affecting the classification speed. Since Multi-GPU v1 is not conducive to batch processing of a large number of test samples, Multi-GPU v2 is designed, and the parallel task scheduling is shown in Fig. 5.

According to Fig. 5, the test samples are divided into several parts and assigned to the specified GPU, respectively. Training samples are copied to each GPU to calculate residuals. Although the classification speed of one sample is slower than Multi-GPU v1, it is more suitable for batch processing of test samples. And the algorithm is described in Algorithm 4.

In Algorithm 4, the test samples are divided according to the number of GPUs (gpu_count). As a comparison, the training samples are divided based on gpu_count in the first multi-GPU

Algorithm 4: NRS Classifier: Multi-GPU v2.

Require: DataTest: Test samples, Λ_Range: range of λ, Training dataset
Ensure: Result: classified result

$thread\_id = omp\_get\_thread\_num()$;
$gpu\_id = cudaGetDeviceCount()$;
for each $l \in [1, C]$ **do**
  get X_l the training samples of lth category;
  $normsAll = normsGPU(Testdata, X_l)$;
  $XY = X_l * DataTest$;
  $X_{sq} = {X_l}^T \cdot X_l$;
  for each $\lambda \in [0, \Lambda\_Range]$ **do**
    $XY_t = XY$;
    for $testNum \in [1, T]$ **do**
      calculate $Testdata$ in each stream
      if $testNum\%\ gpu\_count ==$ thread_id **then**
        $i = testNum$;
      end if
      $streamBuf[i] = X_{sq}$;
      $Temp = geagGPU(X_{sq}, normsAll[i], \lambda)$;
      $\alpha_l[i] = posv(Temp, XY[i])$;
    end for
    $\tilde{y}_l = X_l * \alpha_l$;
    $res = norm1GPU(\tilde{y}_l, Testdata)$;
  end for
end for
$Result(y, \lambda) = minKernel(res)$;

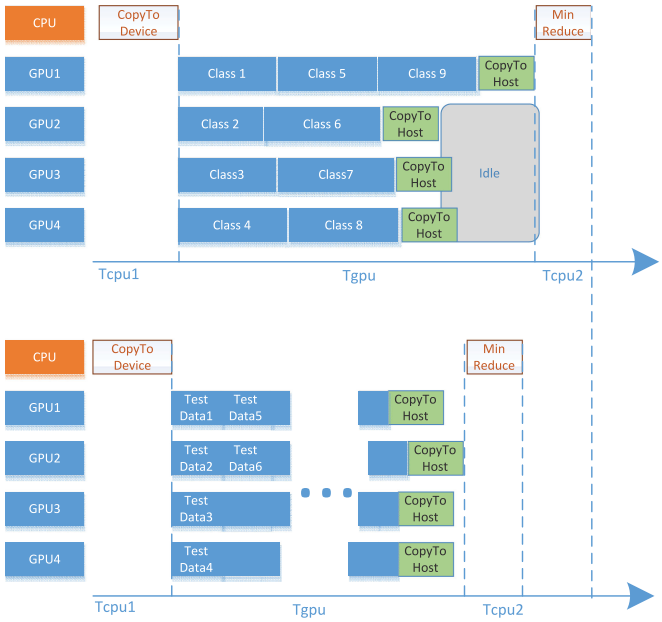


Fig. 6. Comparison of task scheduling: Multi-GPU v1 and Multi-GPU v2.

TABLE I
EXPERIMENT ENVIRONMENT SETUP

Device	Model	Memory	Cores (CUDA SP)
CPU	Intel Xeon E5-2630	64GB	12*2
GPU1	Tesla K10	4GB	1536
GPU2	Tesla K10	4GB	1536
GPU3	Tesla K10	4GB	1536
GPU4	Tesla K10	4GB	1536

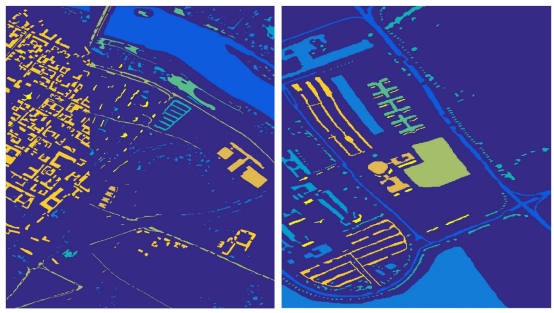


Fig. 7. Ground truths of two hyperspectral images: $PaviaCentre$ (left), $PaviaUniversity$ (right).

TABLE II
DETAILS OF DATASETS USED AS EXPERIMENTS

Types		Pavia Centre		Pavia University	
	No.	Name	Number	Name	Number
Classes	1	Water	824	Asphalt	6631
	2	Trees	820	Meadows	18649
	3	Asphalt	816	Gravel	2099
	4	Bricks	808	Trees	3064
	5	Bitumen	808	Metal sheets	1345
	6	Tiles	1260	Bare Soil	5029
	7	Shadows	476	Bitumen	1330
	8	Meadows	824	Bricks	3682
	9	Bare Soil	820	Shadows	947
	Total	-	42776	-	148152
Bands		102		103	
Size		1096×715		610×340	

version. In detail, the comparison of two parallel task scheduling is shown in Fig. 6. In the first task division method, each GPU only deals with some training samples and training data of the specified category and all test data will be copied from host memory to device memory. The average device memory usage of each GPU is $sizeof(TrainData/gpu\_count) + sizeof(TestData)$. In the second task partition, whole training data and $TestDate/gpu\_count$ of test data will be copied form host memory to device memory. Therefore, the total device memory usage of each GPU is $sizeof(TrainData) + sizeof(TestData/gpu\_count)$.

IV. EXPERIMENTAL RESULTS

In this section, a series of experiments are performed for evaluating the efficiency of single- and multi-GPU implementations. First, the experiment setup is introduced, including computing environment and HSI. Next, the classification accuracy of the NRS algorithm is analyzed. Then, the process of the NRS algorithm is divided into six steps. The computational efficiency comparison for each step can be analyzed to demonstrate the effectiveness of our methods based on both single and multi-GPU implementations. Finally, the results of the two experiments are exhibited to discuss our methods.

A. Experiment Setup

1) Computing Environment Specification: The experimental environment consists of one Xeon with a 24-core processor and four NVIDIA Tesla K10 GPUs, as shown in Table I. Each Tesla K10 has 448 cores with 8 multiprocessors, single-precision floating-point performance of 500 GFlops, double-precision floating-point performance of 200 GFlops, total dedicated memory of 3.5 GB (3583 MB), and power consumption of 300 W. All devices are installed in one server so that we could neglect the influence of network setup.

2) Hyperspectral Image Dataset: In order to verify the effectiveness of our methods, two hyperspectral image datasets are used for comparative experiments, i.e., $PaviaCentre$ with 102 bands and $PaviaUniversity$ with 103 bands. The two datasets are acquired by the ROSIS sensor during a flight campaign over Pavia of northern Italy. The geometric resolution is 1.3 m. The image ground truths differentiate nine classes, and their ground truths are shown in Fig. 7. In the first dataset, the image size is 1096×715 after ignoring some strip samples, and 42 776 pixels are labeled. In the second dataset, the image size is 610×340, with 148 152 labeled pixels. The detailed information of two datasets is shown in Table II.

TABLE III
SIZE OF DATASET

Name	Pavia Centre	Pavia University
Test Samples	100× 9	80× 9
Training Samples	20× 9	15× 9
	100×9	75×9
	200×9	150×9
	300×9	225×9
	500×9	375×9
	1000×9	750×9

TABLE IV
CLASSIFICATION ACCURACIES OBTAINED FOR PAVIA CENTRE BY DIFFERENT λ

λ	The number of training samples for each category					
	20	100	200	300	500	1000
0.10	92.56%	95.00%	96.22%	95.89%	94.22%	94.00%
0.30	92.11%	95.67%	96.44%	96.22%	95.89%	95.89%
0.50	91.56%	95.67%	96.22%	96.56%	96.11%	96.22%
0.70	90.67%	95.22%	96.11%	96.67%	96.00%	96.67%
0.90	90.33%	94.78%	95.89%	96.44%	96.22%	96.22%
1.10	89.78%	94.67%	95.78%	96.89%	96.22%	96.33%
1.30	90.00%	94.56%	95.89%	97.22%	96.44%	96.56%
1.50	89.22%	94.44%	95.67%	96.78%	96.56%	96.56%
1.70	89.00%	94.67%	95.33%	96.33%	96.44%	96.67%
1.90	89.00%	94.78%	95.33%	96.33%	96.22%	96.78%
Ave	90.42%	94.95%	95.89%	96.53%	96.03%	96.19%
Max	92.56%	95.67%	96.44%	97.22%	96.56%	96.78%

Based on the datasets, a few training samples and test samples are randomly picked for verification. As shown in Table III , the number of training samples of each category in the two experiments is kept consistent to ensure the equalization of training samples. In the first experiment, six tests are performed with different numbers of training samples (20 × 9, 100 × 9, 200 × 9, 300 × 9, 500 × 9, and 1000 × 9). The classification accuracy and time are evaluated by 900 (100 × 9) test samples. Correspondingly, six tests are also performed with different numbers of training samples (15 × 9, 75 × 9, 150 × 9, 225 × 9, 375 × 9, and 750 × 9) in the second experiment, and the results are evaluated by 720 (80 × 9) test samples. Moreover, the number of training samples in the two experiments is set slightly different, so as to prove the effectiveness of our methods from different aspects. Training samples and test samples are stored line-by-line in float binary format (IEEE754) without the header.

B. Classification Accuracy Analysis

According to the analysis in Section III, the parallel methods must speed up the algorithm operation without reducing the classification accuracy. Therefore, accuracies obtained for Pavia Centre and Pavia University by different regularization parameters are first analyzed, which is shown in Tables IV and V. It is clear that the accuracy is similar to the serial version algorithm. Meanwhile, the result is consistent with Li's conclusion [22].

In both tables, the classification accuracy varies with the number of training samples and regularization parameters. In the first experiment, the average accuracy, namely Ave, is maintained at a high and stable level after the number of training samples in each category exceeds 200. The maximum accuracy, namely Max, is 97.22%, and the number of training samples accounted for 6%. In the second experiment, the maximum accuracy is 97.22%, and the number of training samples accounted for 0.9%. The results show that the classifier can get a good classification result under the limited training samples [22].

TABLE V
CLASSIFICATION ACCURACIES OBTAINED FOR $PaviaUniversity$ BY DIFFERENT λ

λ	The number of training samples for each category					
	15	75	150	225	375	750
0.10	86.78%	90.67%	92.11%	91.56%	90.11%	88.67%
0.30	86.78%	91.89%	93.78%	92.78%	91.56%	90.22%
0.50	86.11%	92.33%	94.33%	93.89%	92.11%	90.44%
0.70	85.89%	92.22%	94.22%	93.44%	92.89%	91.56%
0.90	84.33%	92.11%	94.78%	93.67%	93.22%	92.00%
1.10	84.33%	91.56%	94.78%	93.89%	93.89%	92.44%
1.30	83.89%	91.78%	95.00%	93.67%	93.89%	92.11%
1.50	83.33%	91.44%	94.89%	94.33%	93.89%	93.00%
1.70	83.00%	91.33%	94.67%	94.11%	94.22%	92.56%
1.90	82.78%	91.33%	94.56%	94.00%	94.44%	92.22%
Ave	84.72%	91.67%	94.31%	93.53%	93.02%	91.52%
Max	86.78%	92.33%	95.00%	94.33%	94.44%	93.00%

C. Computational Efficiency Comparison for Two Experiments

In Algorithm 2, a series of parallel versions have been designed. The algorithm (2) is designed as a comparison experiment, in which the third-party C library CBLAS is used for the matrix operation. The classification consumption time based on the two datasets is shown in Tables VI and VII.

In the first experiment, although the CPU-based algorithm is faster than the single-GPU algorithms when the training samples per class is 20, the Multi-GPU v2 algorithm has achieved over $3.39\times$ acceleration. Especially compared with the CPU-based algorithm, the algorithms based on multiple GPUs accelerate by $2.3\times$ at least and $360\times$ at most. In addition, whenever it is a single-GPU algorithm or multi-GPUs algorithm, their speedup increases with the increase of the training sample number, as shown in Fig. 8. At the same time, the computational performance of the Multi-GPU v2 algorithm is better than that of the Multi-GPU v1 algorithm, which is consistent with the description in Section III-C.

In terms of the six tests, the number of training samples of each category in the second experiment is set slightly smaller than that in the first experiment, so as to verify the effectiveness of the algorithms more comprehensively. Because the number of test samples and the training samples are smaller than the first experiment, the computation speed is faster than the first experiment. From Table VII, the algorithms based on single GPU accelerate by $0.28\times$ at least and $12.85\times$ at most, and the

TABLE VI
EFFICIENCY COMPARISON OF SINGLE-GPU- AND MULTI-GPU-BASED NRS ON PAVIA CENTRE DATASET UNDER DIFFERENT SAMPLE PERCENTAGES

Per Class	CPU	Singal GPU				MultiGPU			
	Serial	GPUv1		GPUv2		MGPUv1		MGPUv2	
	Time (s)	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
20	8.82	20.08	0.44	11.61	0.76	3.80	2.32	**2.60**	**3.39**
100	126.26	67.31	1.88	29.28	4.31	9.96	12.68	**7.75**	**16.30**
200	1103.50	272.96	4.04	211.16	5.23	72.12	15.30	**53.89**	**20.48**
300	2476.15	350.57	7.06	248.00	9.98	81.57	30.36	**61.75**	**40.10**
500	10219.19	506.66	20.17	326.58	31.29	109.85	93.03	**82.63**	**123.68**
1000	64034.12	1069.91	59.85	699.29	91.57	234.50	273.07	**177.58**	**360.59**

The boldface values mean the optimal results in this experiment.

TABLE VII
EFFICIENCY COMPARISON OF SINGLE-GPU- AND MULTI-GPU-BASED NRS ON PAVIA UNIVERSITY DATASET UNDER DIFFERENT SAMPLE PERCENTAGES

Per Class	CPU	Singal GPU				MultiGPU			
	Serial	GPUv1		GPUv2		MGPUv1		MGPUv2	
	Time (s)	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
15	3.78	13.39	0.28	9.25	0.41	2.90	1.31	**2.54**	**1.49**
75	21.52	21.06	1.02	18.45	1.17	6.09	3.54	**4.46**	**4.82**
150	89.40	61.54	1.45	34.15	2.62	11.96	7.48	**8.51**	**10.51**
225	581.99	210.63	2.76	174.50	3.34	57.86	10.06	**43.92**	**13.25**
375	1169.55	277.65	4.21	202.65	5.77	68.85	16.99	**51.06**	**22.90**
750	4938.15	510.91	9.67	385.14	12.82	129.95	38.00	**97.64**	**50.57**

The boldface values mean the optimal results in this experiment.

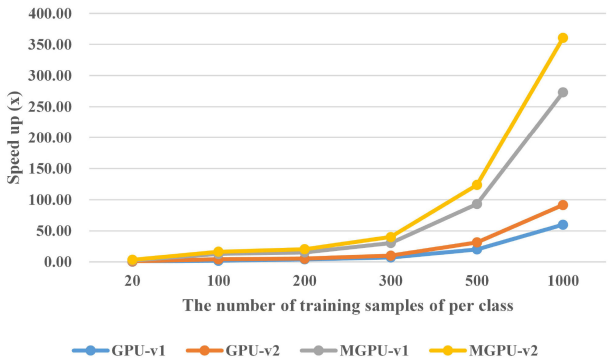


Fig. 8. Speedup of different versions in the first experiment.

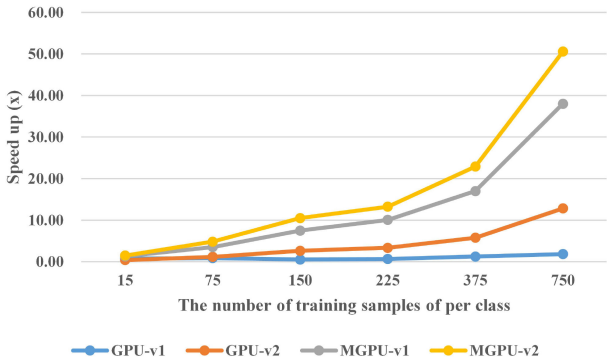


Fig. 9. Speedup of different versions in the second experiment.

algorithms based on multi-GPUs accelerate by $1.3\times$ at least and $50.57\times$ at most. As shown in Fig. 9, GPUs are better at computationally intensive operations than CPUs.

V. CONCLUSION

This article first proposed a series of parallel methods to optimize the NRS classifier by GPU. An optimal serial NRS model is introduced to improve the performance of the data stream by changing the loop order of variables. In addition, a parallel implementation of the NRS algorithm is developed by NVIDIA CUDAs GPU. From the experimental results of actual hyperspectral datasets, including comparisons with equivalent serial versions, the proposed method has a higher speedup and computational efficiency without decreasing accuracy. Furthermore, this method can be significantly improved by adopting an updated GPU platform. Due to the unique advantages of the NRS classifier, our method is meaningful.

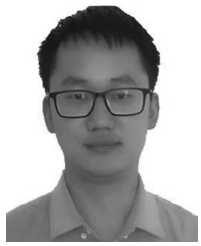
REFERENCES

[1] D. Landgrebe, "Hyperspectral image data analysis," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 17–28, Jan. 2002.
[2] D. Hong, X. Wu, P. Ghamisi, J. Chanussot, N. Yokoya, and X. X. Zhu, "Invariant attribute profiles: A spatial-frequency joint feature extractor for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 6, pp. 3791–3808, Jun. 2020.
[3] L. Gao, B. Zhao, X. Jia, W. Liao, and B. Zhang, "Optimized kernel minimum noise fraction transformation for hyperspectral image classification," *Remote Sens.*, vol. 9, no. 6, 2017, Art. no. 548.
[4] A. Goetz, G. Vane, J. Solomon, and B. Rock, "Imaging spectrometry for earth remote sensing," *Science*, vol. 228, pp. 1147–1153, 1985.

[5] D. Hong, N. Yokoya, and X. X. Zhu, "Learning a robust local manifold representation for hyperspectral dimensionality reduction," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 6, pp. 2960–2975, Jun. 2017.
[6] "Learning to propagate labels on graphs: An iterative multitask regression framework for semi-supervised hyperspectral dimensionality reduction," *ISPRS J. Photogrammetry Remote Sens.*, vol. 158, pp. 35–49, 2019.
[7] P. S. Roy, C. B. S. Dutt, and P. K. Joshi, "Tropical forest resource assessment and monitoring," *J. Indian Soc. Remote Sens.*, vol. 43, no. 1, pp. 21–37, 2002.
[8] X. Wu, D. Hong, P. Ghamisi, W. Li, and R. Tao, "MsRi-CCF: Multi-scale and rotation-insensitive convolutional channel features for geospatial object detection," *Remote Sens.*, vol. 10, no. 12, 2018, Art. no. 1990.
[9] Y. Xie and Q. Weng, "Spatiotemporally enhancing time-series DMSP/OLS nighttime light imagery for assessing large-scale urban dynamics," *ISPRS J. Photogrammetry Remote Sens.*, vol. 128, pp. 1–15, 2017.
[10] Q. Yin, W. Hong, and F. Zhang, "Optimal combination of polarimetric features for vegetation classification in PolSAR image," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 10, pp. 3919–3931, Oct. 2019.
[11] L. Wei, G. Wu, Z. Fan, and D. Qian, "Hyperspectral image classification using deep pixel-pair features," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 2, pp. 844–853, Feb. 2017.
[12] H. Yu *et al.*, "Global spatial and local spectral similarity-based manifold learning group sparse representation for hyperspectral imagery classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 5, pp. 3043–3056, May 2020.
[13] J. Ni, F. Zhang, Q. Yin, and H.-C. Li, "Robust weighting nearest regularized subspace classifier for PolSAR imagery," *IEEE Signal Process. Lett.*, vol. 26, no. 10, pp. 1496–1500, Oct. 2019.
[14] V. Srivastava and B. Biswas, "CNN-based salient features in HSI image semantic target prediction," *Connection Sci.*, vol. 32, pp. 113–131, 2020.
[15] F. Zhou, R. Hang, Q. Liu, and X. Yuan, "Hyperspectral image classification using spectral-spatial LSTMs," *Neurocomputing*, vol. 328, pp. 39–47, 2019.
[16] W. Hu, H. Li, L. Pan, W. Li, R. Tao, and Q. Du, "Spatial-spectral feature extraction via deep ConvLSTM neural networks for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 6, pp. 4237–4250, Jun. 2020.
[17] Y. Xu, L. Zhang, B. Du, and F. Zhang, "Spectral–spatial unified networks for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 10, pp. 5893–5909, Oct. 2018.
[18] W. Xie, X. Jia, Y. Li, and J. Lei, "Hyperspectral image super-resolution using deep feature matrix factorization," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 8, pp. 6055–6067, Aug. 2019.
[19] 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.
[20] L. Zhang, M. Yang, and X. Feng, "Sparse representation or collaborative representation: Which helps face recognition?" in *Proc. IEEE Int. Conf. Comput. Vision*, 2011, pp. 471–478.
[21] W. Li, Q. Du, F. Zhang, and W. Hu, "Hyperspectral image classification by fusing collaborative and sparse representations," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 9, pp. 4178–4187, Sep. 2016.
[22] W. Li, E. W. Tramel, S. Prasad, and J. E. Fowler, "Nearest regularized subspace for hyperspectral classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 1, pp. 477–489, Jan. 2014.
[23] M. Xiong, F. Zhang, Q. Ran, W. Hu, and W. Li, "Representation-based classifications with Markov random field model for hyperspectral urban data," *J. Appl. Remote Sens.*, vol. 8, no. 1, 2014, Art. no. 085097.
[24] W. Li, S. Prasad, J. E. Fowler, and L. M. Bruce, "Locality-preserving dimensionality reduction and classification for hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 4, pp. 1185–1198, Apr. 2012.
[25] H. Yu, L. Gao, W. Liao, B. Zhang, A. Piurica, and W. Philips, "Multiscale superpixel-level subspace-based support vector machines for hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 11, pp. 2142–2146, Nov. 2017.
[26] J. Sun *et al.*, "An efficient and scalable framework for processing remotely sensed big data in cloud computing environments," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 7, pp. 4294–4308, Jul. 2019.
[27] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 6, pp. 2270–2278, Jun. 2016.
[28] F. Zhang, Y. Lin, and W. Hong, "SAR echo distributed simulation based on grid computing," *J. Syst. Simul.*, vol. 20, no. 12, p. 3165–3170, 2008.
[29] F. Zhang, C. Hu, W. Li, W. Hu, P. Wang, and H.-C. Li, "A deep collaborative computing based SAR raw data simulation on multiple CPU/GPU platform," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 2, pp. 387–399, Feb. 2017.
[30] H. Yang, Q. Du, and G. Chen, "Unsupervised hyperspectral band selection using graphics processing units," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 660–668, Sep. 2011.
[31] A. Agathos, J. Li, D. Petcu, and A. Plaza, "Multi-GPU implementation of the minimum volume simplex analysis algorithm for hyperspectral unmixing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2281–2296, Jun. 2014.
[32] F. Zhang, C. Hu, W. Li, W. Hu, and H.-C. Li, "Accelerating time-domain SAR raw data simulation for large areas using multi-GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 9, pp. 3956–3966, Sep. 2014.
[33] S. Sánchez, R. Ramalho, L. Sousa, and A. Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs," *J. Real-Time Image Process.*, vol. 10, no. 3, pp. 469–483, 2015.
[34] F. Zhang, G. Li, W. Li, W. Hu, and Y. Hu, "Accelerating spaceborne SAR imaging using multiple CPU/GPU deep collaborative computing," *Sensors*, vol. 16, no. 4, 2016, Art. no. 494.
[35] F. Zhang, X. Yao, H. Tang, Q. Yin, Y. Hu, and B. Lei, "Multiple mode SAR raw data simulation and parallel acceleration for Gaofen-3 mission," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 6, pp. 2115–2126, Jun. 2018.
[36] W. Li, L. Zhang, and L. Zhang, "GPU parallel implementation of isometric mapping for hyperspectral classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 9, pp. 1532–1536, Sep. 2017.
[37] L. Pan, H.-C. Li, and J. Ni, "GPU-based fast hyperspectral image classification using joint sparse representation with spectral consistency constraint," *J. Real-Time Image Process.*, vol. 15, no. 3, pp. 463–475, 2018.
[38] Z. Wu, Q. Wang, A. Plaza, J. Li, J. Liu, and Z. Wei, "Parallel implementation of sparse representation classifiers for hyperspectral imagery on GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2912–2925, Jun. 2015.
[39] Z. Wu, J. Liu, A. Plaza, J. Li, and Z. Wei, "GPU implementation of composite kernels for hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 9, pp. 1973–1977, Sep. 2015.
[40] Z. Wu, Q. Wang, A. Plaza, J. Li, L. Sun, and Z. Wei, "Parallel spatial–spectral hyperspectral image classification with sparse representation and Markov random fields on GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2926–2938, Jun. 2015.
[41] Z. Wu *et al.*, "GPU parallel implementation of spatially adaptive hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 4, pp. 1131–1143, Apr. 2018.
[42] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, "CULA: Hybrid GPU accelerated linear algebra routines," *Proc. SPIE*, vol. 7705, 2010, Art. no. 770502.

Zhixin Li received the B.E. degree in communication engineering from the Beijing University of Chemical Technology, Beijing, China, in 2015, and the M.A.S. degree in interdisciplinary information studies from the University of Tokyo, Tokyo, Japan, in 2019.

He is currently an Engineer with LINE Corporation, Tokyo. His research interests include high-performance computing, artificial intelligence, and bioinformatics.



Jun Ni (Student Member, IEEE) received the B.E. degree in electronic and information engineering in 2016 from the Beijing University of Chemical Technology, Beijing, China, where he is currently working toward the Ph.D. degree in pattern recognition and intelligent system with the College of Information Science and Technology.

His research interests include PolSAR classification, parallel computing, and imaging processing.

Fan Zhang (Senior Member, IEEE) received the B.E. degree in communication engineering from the Civil Aviation University of China, Tianjin, China, in 2002, the M.S. degree in signal and information processing from Beihang University, Beijing, China, in 2005, and the Ph.D. degree in signal and information processing from the Institute of Electronics, Chinese Academy of Sciences, Beijing, in 2008.

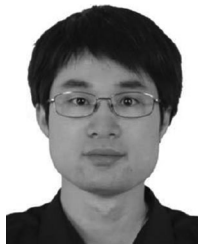
He is currently a Full Professor in electronic and information engineering with the College of Information Science and Technology, Beijing University of Chemical Technology, Beijing, China. His research interests include remote sensing image processing, high-performance computing, and artificial intelligence.

Dr. Zhang is an Associate Editor for IEEE ACCESS and a Reviewer for the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING, the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, etc.

Wei Li (Senior Member, IEEE) received the B.E. degree in telecommunications engineering from Xidian University, Xi'an, China, in 2007, the M.S. degree in information science and technology from Sun Yat-sen University, Guangzhou, China, in 2009, and the Ph.D. degree in electrical and computer engineering from Mississippi State University, Starkville, MS, USA, in 2012.

Subsequently, he spent one year as a Postdoctoral Researcher with the University of California, Davis, CA, USA. He was a Professor with the College of Information Science and Technology, Beijing University of Chemical Technology, from 2013 to 2019. He is currently a Professor with the School of Information and Electronics, Beijing Institute of Technology, Beijing. His research interests include hyperspectral image analysis, pattern recognition, and data compression.

Prof. Li is currently an Associate Editor for the IEEE SIGNAL PROCESSING LETTERS and the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He was a Guest Editor for special issue of the *Journal of Real-Time Image Processing*, *Remote Sensing*, and IEEE JSTARS. He was the recipient of the 2015 Best Reviewer Award from IEEE Geoscience and Remote Sensing Society for his service for IEEE JSTARS and the Outstanding Paper Award at IEEE International Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (Whispers), 2019.



Yongsheng Zhou (Member, IEEE) received the B.E. degree in communication engineering from the Beijing Information Science and Technology University, Beijing, China, in 2005, and the Ph.D. degree in signal and information processing from the Institute of Electronics, Chinese Academy of Sciences, Beijing, in 2010.

He is currently a Professor of Electronic and Information Engineering with the College of Information Science and Technology, Beijing University of Chemical Technology, Beijing. His general research interests include target detection and recognition from microwave remotely sensed image, digital signal, and image processing, etc.