GPU Implementation of Composite Kernels for Hyperspectral Image Classification

Zebin Wu, *Member, IEEE*, Jiafu Liu, Antonio Plaza, *Fellow, IEEE*, Jun Li, *Member, IEEE*, and Zhihui Wei

***Abstract*—In this letter, we present an efficient parallel implementation of composite kernels in support vector machines (SVMs) for hyperspectral image (HSI) classification. Our implementation makes effective use of commodity graphics processing units (GPUs). Specifically, we port the calculation of composite kernels to GPUs, perform intensive computations based on NVidia's compute unified device architecture, and execute the rest of the operations related with control and small data calculations in the CPU. Our experimental results, conducted using real hyperspectral data sets and NVidia GPU platforms, indicate significant improvements in terms of computational effectiveness, achieving near-real-time performance of spatial–spectral HSI classification for the first time in the literature.**

***Index Terms*—Composite kernels, graphics processing units (GPUs), hyperspectral classification, support vector machines (SVMs).**

I. Introduction

HYPERSPECTRAL image (HSI) classification is an active research area in remote sensing [1]. An important challenge for HSI classification is the high dimensionality of the data, together with the limited number of labeled training samples available *a priori*. Various methods have been developed for HSI classification, and these methods are generally categorized into supervised and unsupervised [2]. Among supervised approaches, kernel methods have been proven to be highly effective [1]. Support vector machines (SVMs) [3] represent one of the most successful techniques for HSI classification due to their robust generalization performance, as well as their ability to handle large input spaces efficiently with a relatively low number of labeled training samples (producing sparse solutions and dealing with noisy samples in a robust way) [4]. The SVM with composite kernel (SVMCK) [5] formulation offers the possibility to easily integrate spatial and spectral information to improve classification accuracy. Due to its enhanced classification accuracy and flexibility to balance the spatial and spectral information, the SVMCK is a popular approach that still suffers from high computational complexity, particularly at the training stage.

In time-critical scenarios, such as target detection for military purposes, monitoring of chemical contamination, or wildfire tracking, it is important to accelerate hyperspectral data analysis techniques. Recent advances in high-performance computing technologies [6], [7] have enabled speeding up HSI processing algorithms by using multicore CPUs [8], field-programmable gate arrays [9], and graphics processing units (GPUs) [10], [11]. Particularly, hybrid CPU–GPU implementations are particularly appealing, since massively parallel and data-intensive computations can be performed in the GPU, whereas the logic control ability of CPUs can still be exploited. The effective utilization of both CPUs and GPUs can provide compelling benefits. To our best knowledge, and despite the importance of SVMCK in the hyperspectral imaging, there are no available parallel implementations in the literature for this method.

In this letter, we develop the first parallel implementation of composite kernels in SVMs on heterogeneous CPU–GPU platforms. The proposed implementation accelerates intensive computations and operations involving large data sets on the GPU by utilizing NVidia's compute unified device architecture (CUDA), executing the rest of the operations (mostly related with control) on the CPU. The method ports the calculation of the composite kernel matrix to the GPU, thus significantly accelerating the algorithm while retaining the same classification accuracy achieved in the serial version. The proposed method is evaluated using real hyperspectral data and compared with the serial and multicore versions.

The remainder of the letter is organized as follows. Section II briefly describes the SVMCK method. Section III presents its parallel implementation. Experimental results are reported in Section IV. Conclusions and hints at plausible future research lines are given in Section V.

II. SVMCK

Let us assume that an HSI is denoted by $X \in R^{l \times rows \times cols}$, where l is the number of spectral bands and $rows \times cols$ is the

Manuscript received March 29, 2015; revised May 1, 2015; accepted June 2, 2015. This work was supported in part by the China Scholarship Fund under Grant 201406845012, by the National Natural Science Foundation of China under Grants 61471199, 61101194, and 11431015, by the Research Fund for the Doctoral Program of Higher Education of China under Grant 20113219120024, by the Fundamental Research Funds for the Central Universities under Grant 30915012204, by the Jiangsu Province Six Top Talents project of China under Grant WLW-011, and 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, University of Extremadura, 10003 Cáceres, Spain (e-mail: zebin.wu@gmail.com).

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: liu_jia_fu@163.com; gswei@njust.edu.cn).

A. Plaza is with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, 10003 Cáceres, 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).

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/LGRS.2015.2441631

number of pixels. Le us also assume that $\boldsymbol{T} = (\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_p) \in R^{l\times p}$ is the training set and $\boldsymbol{S} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_q) \in R^{l\times q}$ is the test set. Similarly, $\mathbf{y} = (y_1, y_2, \ldots, y_q) \in R^{l\times q}$, and $y_i \in \{1, -1\}$ denotes the class label of $\boldsymbol{s}_i$. The SVM can be trained by the following model [5]:

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{p}\xi_i \\ \text{subject to} \quad & y_i\left(\boldsymbol{w}^T\phi(\boldsymbol{t}_i) + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0, \qquad \forall\, i = 1, \ldots, p \end{aligned} \tag{1}$$

and its Lagrangian dual is formulated as

$$\min_{\alpha} \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \tag{2}$$

subject to $y^T\boldsymbol{\alpha} = 0,\ 0 \leq \alpha_i \leq C,\ i = 1, \ldots, p$, where $\boldsymbol{w}$ and b define a linear classifier in the feature space; $\phi(\boldsymbol{t}_i)$ is a nonlinear mapping function that projects $\boldsymbol{t}_i$ to enhance its separability; C is a regularization parameter that controls the generalization capabilities of the classifier; $\mathbf{1} = [1, \ldots, 1]^T$ and $\boldsymbol{Q}$ is a $p \times p$ positive semidefinite kernel matrix, with $Q_{ij} = y_i y_j K(\boldsymbol{t}_i, \boldsymbol{t}_j)$, and $K(\boldsymbol{t}_i, \boldsymbol{t}_j) =< \phi(\boldsymbol{t}_i), \phi(\boldsymbol{t}_j) >$ is the kernel function.

A detailed formulation of the SVM optimization problem can be found in [12]. After solving model (2), the decision function for classifying a certain test sample $\boldsymbol{s}$ is defined as

$$f(\boldsymbol{s}) = \operatorname{sgn}\left(\sum_{i=1}^{p} y_i\alpha_i K(\boldsymbol{t}_i, \boldsymbol{s}) + b\right). \tag{3}$$

There are many implementations of SVM available, among which LIBSVM [13] is perhaps the most popular. It offers a variety of classification modes and kernels. In this letter, we use the LIBSVM with C-support vector classification mode to train the SVM model and obtain the final classification results.

On the other hand, any kernel function that fulfills Mercer's condition can be used in the context of SVMs and kernel methods, such as the linear kernel, the polynomial kernel, the radial basis function (RBF), etc. However, spectral information alone is generally not enough to obtain high classification accuracy. In [5], a framework for SVM classification with composite kernels is presented to enhance classification accuracy by combining spatial and spectral features. This formulation has flexibility to balance the spatial and spectral information. There are many different types of composite kernel methods, e.g., stacked features approach, direct summation kernel, weighted summation kernel, or cross-information kernel [5]. In this letter, we adopt the weighted summation kernel as a case study to illustrate the advantages and potential challenges of applying parallel optimization principles on GPU platforms. Moreover, the polynomial kernel is used to model spectral features, and the RBF kernel is utilized to model spatial features, as suggested in [5]. The proposed parallel implementation is valid for any composite kernel formulation. A serial implementation of SVMCK (SVMCK-S) is summarized in Algorithm 1.

Algorithm 1. SVMCK-S.

Input: HSI image $\boldsymbol{X} \in R^{l\times rows\times cols}$, training samples set $\boldsymbol{T} = (\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_p) \in R^{l\times p}$, test samples set $\boldsymbol{S} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_q) \in R^{l\times q}$

Step 1. $\boldsymbol{f}_1 = \boldsymbol{T}^T \bullet \boldsymbol{T}$, $\boldsymbol{f}_2 = \boldsymbol{T}^T \bullet \boldsymbol{S}$

Step 2. Calculate polynomial kernel for spectral features:

$$K_{s1} = (\boldsymbol{f}_1 + \mathbf{1})^d,\ K_{s2} = (\boldsymbol{f}_2 + \mathbf{1})^d$$

Step 3. Calculate RBF kernel for spatial features

Step 3.1. For $k = 1$: p,

Compute the mean values and standard deviations for $\boldsymbol{t}_k$ in local window coordinates:

$$\bar{t_k} = \frac{1}{n}\sum_i\sum_j X_{ij},\ d_{t_k} = \frac{1}{n-1}\left(\sum_i\sum_j (X_{ij} - \bar{t}_k)\right)^{\frac{1}{2}}$$

Step 3.2. $\boldsymbol{T}' = (\boldsymbol{t}_i^w) = (\bar{\boldsymbol{t}}\boldsymbol{d}_t)^T$

Step 3.3. For $k = 1$: q,

Compute the mean values and standard deviations for $\boldsymbol{s}_k$ in local window coordinates:

$$\bar{s}_k = \frac{1}{n}\sum_i\sum_j X_{ij},\ d_{s_k} = \frac{1}{n-1}\left(\sum_i\sum_j (X_{ij} - \bar{s}_k)\right)^{\frac{1}{2}}$$

Step 3.4. $\boldsymbol{S}' = (\mathbf{s}_i^w) = (\bar{\boldsymbol{s}}\boldsymbol{d}_s)^T$

Step 3.5. Calculate RBF kernel for spatial features:

$$K_{w1}\left(\boldsymbol{t}_i^w, \boldsymbol{t}_j^w\right) = \exp\left(-\frac{\left\|\boldsymbol{t}_i^w - \boldsymbol{s}_j^w\right\|^2}{2\sigma^2}\right)$$

$$K_{w2}\left(\boldsymbol{t}_i^w, \boldsymbol{s}_j^w\right) = \exp\left(-\frac{\left\|\boldsymbol{t}_i^w - \boldsymbol{s}_j^w\right\|^2}{2\sigma^2}\right)$$

Step 4. Calculate composite kernel:

$$K_1 = (1-\mu)K_{s1} + \mu K_{w1},\ K_2 = (1-\mu)K_{s2} + \mu K_{w2}$$

Step 5. Train the training set with K_1

Step 6. Use the SVM model to predict:

$$\operatorname{class}(\boldsymbol{s}_{j=1,2,\ldots,q}) = \operatorname{sgn}\left(\sum_{i=1}^{p} y_i\alpha_i K(\boldsymbol{t}_i, \boldsymbol{s}_j) + b\right)$$

Output: the class labels of $\boldsymbol{S}$.

End

III. GPU Implementation

Algorithm 1 is expensive in computational terms, particularly for the calculation of composite kernels, which involves operations with big matrices. Here, we take the AVIRIS Indian Pines hyperspectral data set with 10% training samples (details of the data set will be given in Section IV-A) as an example to experimentally analyze the computational bottleneck of Algorithm 1. Fig. 1 shows the percentage of total CPU time consumed by the different steps of Algorithm 1 (obtained by Microsoft Visual Studio Profiler). Since the calculation of composite kernels takes almost 95% of the total execution time (with SVM training and predicting taking only 1% and 2%, respectively), the key for optimizing Algorithm 1 is accelerating the calculation of composite kernels.

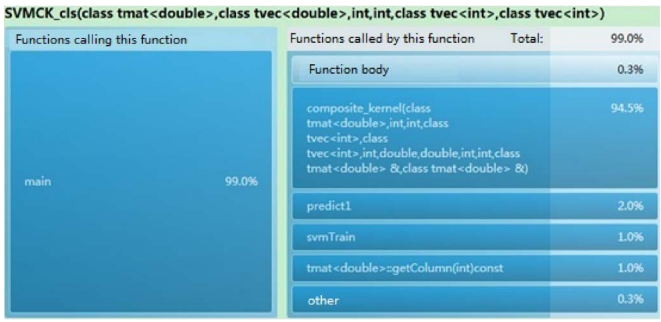

Fig. 1. Profiling the processing of AVIRIS Indian Pines with Algorithm 1.

GPUs are specifically optimized for computational and memory-intensive problems, whereas CPUs devote more resources to caching or control flow operations. Therefore, a CPU–GPU heterogeneous platform is appealing to solve problems such as SVMCK, since GPUs can be used to accelerate the composite kernel-related computations and the host CPU can be used to perform other small data computations and most of the control operations. Taking into consideration that the calculation of composite kernels is a dominant part of Algorithm 1 that consists of operations on big matrices and high-dimensional vectors, we precompute them on the device (GPU), copy them to the host (CPU), and then cache them in the available memory on the host. The rest of the computations (related with control and small data structures) are computed on the CPU to dramatically reduce the data transfer between the device and the host. Details of the parallel implementation are given next.

First, in terms of input/ouput (I/O) transfers between host and device, we allocate memory on the device, copy the original data from the host to the device at the beginning of the algorithm, and then transfer the final results from the device to the host at the end. In the parallel implementation, the data are stored in the GPU memory as much as possible, and the storage space for intermediate variables of the iterative process is allocated in advance, thus minimizing the data transfers between the host and the device.

Second, the calculation of the polynomial kernel for spectral features, i.e., Step 1 of Algorithm 1, is realized by invoking `cublasDgemm` (a highly effective matrix multiplication function included in the NVidia's CUDA basic linear algebra library, cuBLAS [14]). It takes full advantage of registers and shared memories of the GPU to achieve high efficiency in the multiplication of big matrices. Then, a kernel function called `POLYKernel` is defined to carry out the operations $K_{s1} = (\boldsymbol{f}_1 + \mathbf{1})^d$ and $K_{s2} = (\boldsymbol{f}_2 + \mathbf{1})^d$. According to the sizes of the training and test sets, we now start a $p \times p$ thread grid and a $p \times q$ thread grid on the GPU to compute K_{s1} and K_{s2}, respectively, where each thread calculates one matrix element. The number of *threads* per *block* is set to 16×16, according to the computing capabilities of NVidia Tesla C2050 (the GPU platform considered in this letter).

Third, the mean values and standard deviations for $\boldsymbol{t}_k$ and $\boldsymbol{s}_k$ are computed by means of a kernel function `SubMatMeanStdKernel`. A $p \times l$ thread grid is launched for the calculation of $\boldsymbol{t}_k$, and a $q \times l$ thread grid is launched for the calculation of s_k. After that, the obtained results are arranged in the form of two different matrices, i.e., $\boldsymbol{T}'$ and $\boldsymbol{S}'$.

As we can see from Step 3.5 in Algorithm 1, the operations involved in the calculation of the RBF kernel are pixel-wise. According to the GPU parallel architecture and the CUDA programming paradigm, we can arrange batches of threads (organized as blocks of threads in grids) executing together and sharing local memories. In order to efficiently utilize the computation power of GPUs and achieve high execution performance, one has to launch a large number of thread blocks in parallel in the grid. Since it is more efficient to perform big matrix computations than pixel-wise operations in the GPU [15], we modified Step 3.5 in Algorithm 1 and carefully orchestrated the pixel-wise calculation in the form of matrix batch computing to speed up the whole process. Algorithm 2 shows in detail how the matrix batch computing is realized in our case for calculating the RBF kernel. After the modifications conducted, Step 3.5.1 in Algorithm 2 can be efficiently calculated by invoking function `cublasDgemm` in the cuBLAS library, and two kernel functions (i.e., `MatRowsSumKernel` and `RBFKernel`) are implemented to perform the rest of the steps in Algorithm 2. The former takes care of the cumulative summation $\boldsymbol{N}_1 = (\|\boldsymbol{t}_i^w\|_2^2)$ and $\boldsymbol{N}_2 = (\|\boldsymbol{s}_i^w\|_2^2)$, which starts p threads (each performs the cumulative sum of a column of $\boldsymbol{T}'$) to compute $\boldsymbol{N}_1$ and launches q threads (each performs the cumulative sum of a column of $\boldsymbol{S}'$) to compute $\boldsymbol{N}_2$. The latter implements Steps 3.5.3 to Step 3.5.6 in Algorithm 2 successively. A thread grid with the same size as $\boldsymbol{D}_1$ ($p \times p$) is now started to compute $\boldsymbol{D}_1$ and K_{w1}, and then, a $p \times q$ thread grid is launched to calculate $\boldsymbol{D}_2$ and K_{w2} on the GPU.

Algorithm 2. Modifications of Algorithm 1 for Calculating the RBF Kernel in the GPU.

Step 3.5.1.

$$\boldsymbol{g}_1 = \boldsymbol{T}'^T \bullet \boldsymbol{T}', \boldsymbol{g}_2 = \boldsymbol{T}'^T \bullet \boldsymbol{S}'$$

Step 3.5.2.

$$\boldsymbol{N}_1 = \left(\|\boldsymbol{t}_i^w\|_2^2\right), \boldsymbol{N}_2 = \left(\|\boldsymbol{s}_i^w\|_2^2\right)$$

Step 3.5.3.

$$\boldsymbol{D}_1 = \left(\begin{pmatrix}1\\1\\\cdots\\1\end{pmatrix}\boldsymbol{N}_1\right)^T + \begin{pmatrix}1\\1\\\cdots\\1\end{pmatrix}\boldsymbol{N}_1 - 2\boldsymbol{g}_1$$

Step 3.5.4.

$$\boldsymbol{D}_2 = \left(\begin{pmatrix}1\\1\\\cdots\\1\end{pmatrix}\boldsymbol{N}_1\right)^T + \begin{pmatrix}1\\1\\\cdots\\1\end{pmatrix}\boldsymbol{N}_2 - 2\boldsymbol{g}_2$$

Step 3.5.5.

$$K_{w1}\left(\boldsymbol{t}_i^w, \boldsymbol{t}_j^w\right) = \exp\left(-\frac{\boldsymbol{D}_1}{2 * \sigma^2}\right)$$

Step 3.5.6.

$$K_{w2}\left(\boldsymbol{t}_i^w, \boldsymbol{s}_j^w\right) = \exp\left(-\frac{\boldsymbol{D}_2}{2 * \sigma^2}\right)$$

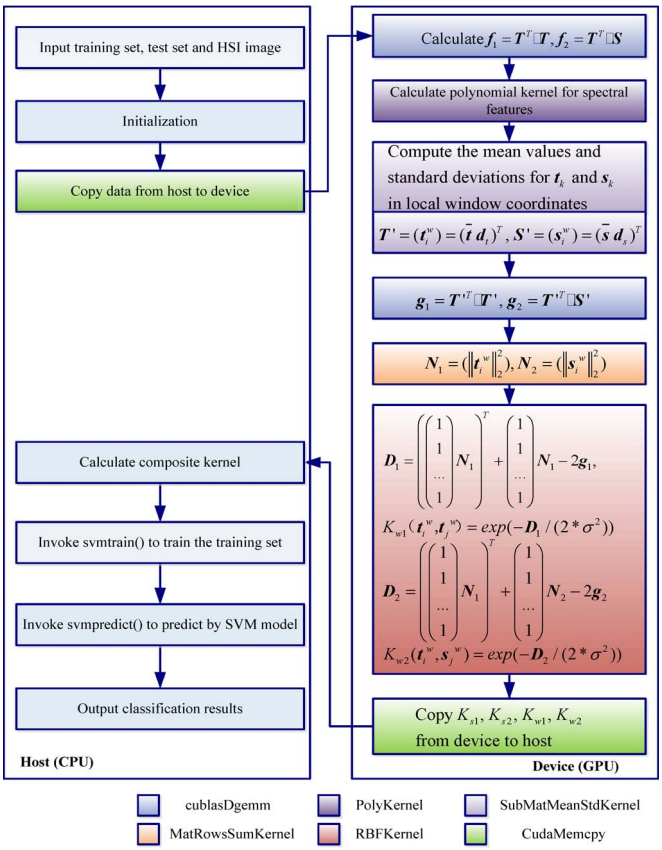

Fig. 2. Summary of the parallel implementation of SVMCK (SVMCK_P).

When $K_{s1}, K_{s2}, K_{w1}, K_{w2}$ have been calculated in the GPU, we pass them to the host and invoke `svmtrain()` (included in LIBSVM) to train the SVM and then invoke `svmpredict()` (also included in LIBSVM) to obtain the final classification results. Fig. 2 summarizes the proposed parallel implementation (SVMCK_P).

IV. Experimental Results

The experimental environment used in our tests is a heterogeneous platform consisting of two CPUs and a GPU. Both CPUs are Intel Xeons E5603 at 1.6 GHz with four cores and 8 GB of main memory. The GPU is an NVidia Tesla C2050, 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 3 GB, 1500 MHz memory (with 384-bit GDDR5 interface), and memory bandwidth of 144 GB/s. The tests were performed using the 64-bit Microsoft Windows 7 operating system. The versions of OpenMP and CUDA are 2.0 and 5.0, respectively.

The classification accuracy and execution performance of SVMCK_P were evaluated on two real HSIs (see Table I) for which ground truth information is available. The classification accuracies are measured by the overall classification accuracy (OA).

A multicore implementation of SVMCK (SVMCK-M) has also been developed using OpenMP [16] to explicitly address multithreaded and shared-memory parallelism. The serial version (SVMCK-S) was executed on one core of the Intel Xeon E5603 CPU, whereas SVMCK-M was run on the eight available cores of the two Intel Xeon E5603 CPUs. These three versions (SVMCK-S, SVMCK-M, and SVMCK-P) were implemented using the C++ programming language and compiled by Visual C++ 2010. All the times reported in the following experiments were measured by the adopted classifiers after ten Monte Carlo runs. In all cases, fivefold cross validation was used to tune the parameters. According to [5], σ, C, μ, and d were varied in the range of $\sigma \in \{10^{-1}, \dots, 10^{3}\}$, $C \in \{10^{-1}, \dots, 10^{7}\}$, $\mu \in \{0, 0.1, \dots, 1.0\}$, and $d \in \{1, \dots, 10\}$, respectively. The window size was set to 9×9 , and a one-against-one multiclassification scheme was adopted.

Our first experiment was carried out using the AVIRIS Indian Pines data set. To assess the impact of the number of training samples on classification accuracy and performance, we tested different numbers of training samples, ranging from 5% to 30% for each class, and used the rest for validation. The classification accuracies are shown in Table II. It can be concluded that SVMCK-S, SVMCK-M, and SVMCK-P obtain very similar classification results. Table III reports the obtained results in terms of processing times and acceleration factors measured after comparing the GPU and multicore parallel implementations of SVMCK with the equivalent serial version. As can be seen from Table III, the acceleration factors achieved by both SVMCK-M and SVMCK-P are stable as the number of training samples is increased. SVMCK-P achieves approximately 11×

TABLE I
Details of the Experimental Hyperspectral Data Sets

	AVIRIS Indian Pines	ROSIS Pavia University
Scenes	Indian Pines region of Northwest Tippecanoe County, 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	9
Labeled pixels	10366	42776

TABLE II
Accuracies (%) for the AVIRIS Indian Pines Data Set

Algorithms	Training Set (Percentage of The Labeled Samples)					
	5%	10%	15%	20%	25%	30%
SVMCK-S	66.39	90.39	95.34	97.56	98.14	98.84
SVMCK-M	66.39	90.39	95.34	97.57	98.14	98.84
SVMCK-P	66.39	90.38	95.34	97.56	98.14	98.84

TABLE III
Performance of Three Versions of SVMCK With AVIRIS Indian Pines

Training Set	SVMCK-S Time (ms)	SVMCK-M		SVMCK-P	
		Time (ms)	Acceleration factor (X)	Time (ms)	Acceleration factor (X)
5%	4902.5	917.2	5.35	449.4	10.91
10%	10768.3	2336.8	4.61	979.2	11.00
15%	18177.8	4143.0	4.39	1765.8	10.29
20%	27116.2	6383.6	4.25	2455.4	11.04
25%	35299.8	8533.4	4.14	3145.0	11.22
30%	44404.0	9868.6	4.50	3843.8	11.55

TABLE IV
I/O TRANSFER TIMES (ms) OF SVMCK-P WITH AVIRIS INDIAN PINES

Training Set	CPUToGPU	GPUToCPU	Total I/O	Total Execution	I/O Percentage
5%	77.969	51.579	129.548	449.4	28.83%
10%	81.991	127.857	209.848	979.2	21.43%
15%	87.222	219.285	306.507	1765.8	17.36%
20%	88.148	314.199	402.347	2455.4	16.39%
25%	94.951	391.798	486.749	3145.0	15.48%
30%	99.348	495.427	594.775	3843.8	15.47%

TABLE V
ACCURACIES (%) FOR THE ROSIS PAVIA UNIVERSITY DATA SET

Algorithms	Training Set (Number of Samples per Class)			
	40 per class	60 per class	80 per class	100 per class
SVMCK-S	91.72	94.19	95.41	95.85
SVMCK-M	91.72	94.19	95.41	95.85
SVMCK-P	91.72	94.19	95.41	95.85

TABLE VI
PERFORMANCE OF THREE VERSIONS OF SVMCK WITH PAVIA UNIVERSITY

Training Set	SVMCK-S Time (ms)	SVMCK-M		SVMCK-I	
		Time (ms)	Acceleration factor (X)	Time (ms)	Acceleration factor (X)
40 per class	16008.7	4851.6	3.30	2525.6	6.34
60 per class	22287.9	6569.2	3.39	3424.3	6.51
80 per class	28629.4	8043.7	3.56	4352.1	6.58
100 per class	35067.5	9787.6	3.58	5327.3	6.58

TABLE VII
I/O TRANSFER TIMES (ms) OF SVMCK-P WITH ROSIS PAVIA UNIVERSITY

Training Set	CPUTo GPU	GPUTo CPU	Total I/O	Total Execution	I/O Percentage
40 per class	226.671	279.482	506.153	2525.6	20.04%
60 per class	248.952	338.483	587.435	3424.3	17.15%
80 per class	233.544	450.798	684.342	4352.1	15.72%
100 per class	232.849	558.010	790.859	5327.3	14.85%

acceleration factor over SVMCK-S on the AVIRIS Indian Pines data set.

Taking into consideration that AVIRIS is a whiskbroom instrument with a scanning rate of 12 Hz, and its cross-track line scan time is as fast as nearly 83 ms to collect one line of 614 spatial samples [17], the whole 10366 samples could be collected in 1.4 s. Therefore, in the case of using 10% of the samples for training, SVMCK-P can classify the image faster than the data are acquired, thus achieving real-time classification performance with accuracy above 90%. Table IV shows the corresponding I/O transfer times.

Tables V–VII show the corresponding results obtained for the ROSIS Pavia University data set. For this experiment, we randomly chose different numbers of training samples, ranging from 40 to 100 samples of each class (the remaining labeled pixels are used as test samples). In this case, the acceleration factors obtained are nearly 6.5×, independently of the size of the training set. ROSIS is a pushbroom instrument with a maximum sampling rate of 62 Hz [18], which means that one line of full-pixel vectors (512 pixels) can be collected in 16 ms, and the whole 42 776 samples could be collected in 1.35 s. When using 40 samples per class, SVMCK-P achieves near-real-time performance.

It can be observed from Tables IV and VII that less than 30% of the total solver time is consumed by data transfers. As a result, SVMCK-P can be defined as compute-bound, with the most significant portion of the time taken by pure computation steps.

V. CONCLUSION AND FUTURE LINES

In this letter, a new parallel implementation of composite kernels in SVMs has been proposed for heterogeneous CPU–GPU platforms. Experimental results on real hyperspectral data sets show remarkable acceleration factors and near-real-time performance, which is appealing for time-critical applications of hyperspectral remote sensing. Future works will explore other parallel implementations on different high-performance computing platforms.

REFERENCES

[1] M. Fauvel, Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton, "Advances in spectral–spatial classification of hyperspectral images," *Proc. IEEE*, vol. 101, no. 3, pp. 652–675, Mar. 2013.

[2] A. Plaza *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, no. S1, pp. S110–S122, Sep. 2009.

[3] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[4] G. Camps-Valls and L. Bruzzone, "Kernel-based methods for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 6, pp. 1351–1362, Jun. 2005.

[5] 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.

[6] C. A. Lee *et al.*, "Recent developments in high performance computing for remote sensing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.

[7] 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.

[8] 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, Jun. 2013.

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

[10] 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.

[11] 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.

[12] B. Schölkopf and A. Smola, *Learning with Kernels—Support Vector Machines, Regularization, Optimization and Beyond*. Cambridge, MA, USA: MIT Press, 2002.

[13] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," 2001. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm

[14] NVIDIA Developer Zone, "CublAS user guide," Jan. 2015. [Online]. Available: http://docs.nvidia.com/cuda/cublas/index.html

[15] J. M. Molero, E. M. Garzón, I. García, E. S. Quintana-Ortí, and A. Plaza, "Efficient implementation of hyperspectral anomaly detection techniques on GPUs and multicore processors" *IEEE J. Sel. Topics Appl. Earth Obs. Remote Sens.*, vol. 7, no. 6, pp. 2256–2266, Jun. 2014.

[16] B. Chapman, G. Jost, and R. Pas, *Using OpenMO: Portable Shared Memory Parallel Programming*. Cambridge, MA, USA: MIT Press, Oct. 2007.

[17] R. Green *et al.*, "Imaging spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Remote Sens. Environ.*, Vol. 65, no. 3, pp. 227–248, Sep. 1998.

[18] S. Holzwarth, A. Müller, and M. Habermeyer, "HySens-DAIS 7915/ROSIS imaging spectrometers at DLR," in *Proc. SPIE 4545, Remote Sensing for Environmental Monitoring, GIS Applications, and Geology*, Jan. 2002, pp. 225–235.