Abstract—Support Vector Machine (SVM) is a classification method that has been widely used in the domain of remote sensing for decades. Although SVM-based classification method achieves good performance for classification accuracy in many studies, it can become very time-consuming in some remote sensing applications such as hyperspectral image classification or large-scale land cover mapping. To improve the efficiency for SVM training and classification in remote sensing applications, we designed and implemented a highly efficient multiclass support vector machine (MMSVM) for × 86-based multicore and many-core architectures such as the Ivy Bridge CPUs and the Intel Xeon Phi coprocessor (MIC) based on our previous MIC-SVM library. Various analysis methods and optimization strategies are employed to fully utilize the multilevel parallelism of our studied architectures. We select several real-world remote sensing datasets to evaluate the performance of our proposed MMSVM. Compared with the widely used serial LIBSVM, our MMSVM achieves 6.3–31.1 × (in training) and 4.9–32.2 × (in classification) speedups on MIC, and 6.9–14.9 × (in training) and 5.5–22.1 × (in classification) speedups on the Ivy Bridge CPUs. We also conduct a performance comparison analysis on different platforms and provide some ideas on how to select the most suitable architecture for specific remote sensing classification problems in order to achieve the best performance.

Index Terms—Classification, multicore and many-core architectures, parallel optimization, performance analysis, support vector machine (SVM).

I. INTRODUCTION

CLASSIFICATION is an important technology with wide attention in the remote sensing domain. Classification of land cover and land use types has been one of the most widely adopted applications in remote sensing. Land cover mapping products play a critical role in various fields including earth system studies, atmospheric models, and ecosystem management, etc. [1]. A large number of classification methods have been developed and applied to land cover mapping, such as maximum likelihood classifier (MLC), support vector machine (SVM), artificial neural network, and random forest (RF), etc. [2]. For instance, in 2013, the finest resolution global land-cover (GLC) maps (30 m) were produced using Landsat TM and ETM+ data based on SVM, RF, J4.8, and MLC classifiers [1]. Following that, a segmentation-based approach was applied to Landsat imagery to downscale coarser-resolution moderate resolution imaging spectroradiometer (MODIS) data (250 m) and other 1-km resolution auxiliary data to the segment scale using SVM and RF classifiers [3].

Hyperspectral image classification is another active research area in remote sensing. Since hyperspectral images generally have hundreds of narrow and continuous spectral bands with detailed spectral signatures of different object materials, classification can be performed more accurately and effectively using hyperspectral images [4]. There are increasing studies that focus on improving hyperspectral image classification algorithms for higher accuracy. Consequently, hyperspectral image classification has been applied to many application domains, such as environmental monitoring, geologic surveys, agricultural production, urban planning, and military reconnaissance, etc. [5].

SVM [6] is a classification method that has been widely used in many fields including text categorization [7], financial analysis [8], bioinformatics [9], and lithological mapping [10], etc. It has also been widely applied to remote sensing domain for a long time, with its good performances for classification accuracy shown in many studies [1], [11]. However, in some cases, the training and classification of SVM can become very
time-consuming. On one hand, for hyperspectral image classification, the large number of features of hyperspectral datasets will limit the efficiency of SVM. On the other hand, in some studies for large-scale land cover mapping studies, both the training and classification processes of SVM can become extremely slow [12]. As large-scale land cover mapping studies often use massive training samples with a complex spatial distribution of their features, a large number of support vectors will arise after training phase. Moreover, since we need to predict the label for billions of pixels in large-scale land cover mapping studies, it is very essential to improve the efficiency of the classification phase of SVM.

In recent years, high-performance computing technologies have achieved great performance in several remote sensing application fields, especially in hyperspectral image processing applications [13], [14]. For example, there have been lots of studies about accelerating spectral unmixing or end-member extraction for hyperspectral image on various high-performance computing platforms including graphics processing unit (GPU) [15], field programmable gate array [16], and distributed system [17], etc. Besides, some studies have applied GPU to hyperspectral image classification algorithms such as SVM with composite kernel [18], sequential minimal optimization (SMO) based SVM [19], sparse representation [20], and sparse multinomial logistic regression [21], etc. However, to our best knowledge, there has been no efficient algorithms in remote sensing field designed for advanced \( \times 86 \)-based multicore and many-core architectures such as the Ivy Bridge CPUs and the Intel Xeon Phi (MIC), even though they have already gained popularity on recent TOP500 list [22].

Meanwhile, in many studies, efficient SVM tools have been designed for various high-performance platforms. Some previous work focused on designing SVM tools for CPUs with relatively few cores and simple memory hierarchies, such as the serial LIBSVM [23] and the SVM with multicore CPU [24]. For GPU platform, a fast binary-class GPUSVM was proposed in [25] that parallelized SVM training and classification on graphics processors. A parallel multiclass SVM on GPUs was proposed in [26]. Although it improves the efficiency of SVM training and classification for datasets mentioned in their paper, it cannot process the large-scale remote sensing datasets used in our research properly. The GPU-tailored approach for training kernelized SVMs [27] proposed a novel sparsity clustering approach to handle sparse datasets cleanly and process different types of datasets effectively, which will be used as a comparison of our proposed algorithm in cross-platform performance evaluation section. For multicore and many-core architectures, an efficient MIC-SVM tool was proposed in our previous work [28] for accelerating the training phase of SVM for binary classification problems, which is the predecessor of our proposed multiclass support vector machine (MMSVM).

In this research, we design and implement MMSVM, a highly efficient parallel SVM for \( \times 86 \)-based multicore and many-core architectures such as the Ivy Bridge CPUs and Intel Knight Corner (KNC) MIC [29]. Various analysis methods and optimization strategies are employed to fully utilize the multilevel parallelism provided by the studied architectures. We select several real-world remote sensing datasets to evaluate the performance of MMSVM. Compared with the widely used serial LIBSVM [23], our MMSVM achieves 6.3–31.1 \( \times \) (in training) and 4.9–32.2 \( \times \) (in classification) speedups on MIC, and 6.9–14.9 \( \times \) (in training) and 5.5–22.1 \( \times \) (in classification) speedups on the Ivy Bridge CPUs. Even compared with an efficient version of GPUSVM [27], MMSVM is still competitive especially for datasets with a large number of features. We also conduct a performance comparison analysis on different platforms to provide insights on how to select the most suitable implementation and architecture for specific remote sensing classification problems.

The rest of this paper is organized as follows. Section II reviews the basic principles of SVM. Section III describes the general flow and several optimization strategies for our MMSVM on the Intel Ivy Bridge CPUs and the Intel Xeon Phi coprocessor (MIC). Section IV provides the experimental results regarding the classification accuracies and speedups of different implementations based on real-word remote sensing datasets. Section V presents some important conclusions of this research.

## II. SVM BACKGROUND

In this section, we review the basic principles of SVM. It consists of two phases: training and classification. We use our training samples to train the SVM and use the model file obtained from the training phase to predict labels for test samples. The details of this process are discussed as follows.

### A. Multiclass SVM Training

The training dataset of SVM can be denoted as a set of data points \( X_i, i \in \{1, 2, \ldots, N\} \) with their labels \( y_i, i \in \{1, 2, \ldots, N\} \), where \( N \) is the total number of training samples. In our research, we group the input data and employ a one-versus-one way to convert a multiclass SVM problem into \( n\text{Class} \times (n\text{Class}−1) / 2 \) binary-class SVM problems, in which \( n\text{Class} \) denotes the number of classes. For each binary-class problem, SVM training can be presented as a quadratic programming problem [shown in (1) and (2)], where \( \alpha_i \) is the Lagrange multiplier, one for each training sample, which will be optimized during the training phase. \( n \) is the number of training samples of a binary-class problem. \( C \) is a regularization constant for balancing the generality and accuracy that can be set by users. \( \text{Kernel}(X_i, X_j) \) denotes the Kernel function. The standard Kernel functions are shown in Table I.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>( \text{Kernel}(X_i, X_j) = X_i^T X_j )</td>
</tr>
<tr>
<td>Polynomial</td>
<td>( \text{Kernel}(X_i, X_j) = (\alpha X_i^T X_j + r)^d )</td>
</tr>
<tr>
<td>Gaussian</td>
<td>( \text{Kernel}(X_i, X_j) = -\gamma |X_i - X_j|_2^2 )</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>( \text{Kernel}(X_i, X_j) = \tanh(\alpha X_i^T X_j + r) )</td>
</tr>
</tbody>
</table>
Algorithm 1: The SMO Algorithm for Multiclass SVM Training.

1: Input the training samples $X_i$ and their labels $y_i$ ($\forall i \in \{1, 2, \ldots, N\}$).
2: Group the training samples and employ a 
One-versus-One way to convert a multiclass SVM problem 
into $n_{\text{class}} \times (n_{\text{class}}-1)/2$ binary-class SVM problems.
3: Initialize $\alpha_i = 0$, $F_i = -y_i(\forall i \in \{1, 2, \ldots, n\})$.
4: Initialize $b_{\text{high}} = -1$, $i_{\text{high}} = \min\{i:y_i = -1\}$, 
$b_{\text{low}} = 1$, $i_{\text{low}} = \max\{i:y_i = 1\}$.
5: Update $\alpha_{\text{high}}$ and $\alpha_{\text{low}}$ according to (4) and (5).
6: If Kernel($X_{\text{high}}, X_i$) is not in memory, then compute 
Kernel($X_{\text{high}}, X_i$) and cache it in memory through 
least-recent-use (LRU) strategy.
7: If Kernel($X_{\text{low}}, X_i$) is not in memory, then compute 
Kernel($X_{\text{low}}, X_i$) and cache it in memory through LRU strategy.
8: Update the $F_i(\forall i \in \{1, 2, \ldots, n\})$ according to (6).
9: Define the index sets $I_{\text{high}}$ and $I_{\text{low}}$ as (7) and (8), and compute 
$i_{\text{high}}, i_{\text{low}}, b_{\text{high}}$ and $b_{\text{low}}$ according to (9) to (12).
10: Update $\alpha_{\text{high}}$, and $\alpha_{\text{low}}$ according to (4) and (5).
11: If iterations meet some certain number, then do shrinking.
12: If $b_{\text{low}} > b_{\text{high}} + 2 \times$ tolerance, then go back to Step 6; else, terminate the iterations.
13: After all binary-class problems finish the iterations, save the final parameters (such as $b$, support vectors, and $\alpha_i$ corresponding to each sample, etc.) of each binary-class problem into the model file.

Maximize:

$$
\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \text{Kernel}(X_i, X_j)
$$

Subject to:

$$
0 \leq \alpha_i \leq C, \forall i \in \{1, 2, \ldots, n\}, \text{and} \sum_{i=1}^{n} \alpha_i y_i = 0.
$$

There are many algorithms for solving the SVM quadratic programming problem. In our research, we implement the popular SMO algorithm [30], [31], which has been used in several SVM tools including LIBSVM [23]. The SMO algorithm can take advantage of the sparse nature of the support vector problem and the simple nature of the constraints in the SVM quadratic programming to minimize the size of the working set in the training phase [25]. Algorithm 1 shows the brief outline of a commonly used SMO algorithm for multiclass SVM training, in which steps 3–12 are conducted for each binary-class SVM problem. The discrepancy between the calculated objective value and the real objective value for each point can be calculated according to (3). The mathematical derivations of the major parameters in Algorithm 1 are shown in (4)–(12).

$$
\begin{align*}
\hat{f}_i &= \sum_{j=1}^{n} \alpha_i y_j \text{Kernel}(X_i, X_j) - y_i \\
\hat{\alpha}_{\text{high}} &= \alpha_{\text{high}} + y_{\text{low}} y_{\text{high}} (\alpha_{\text{low}} - \alpha_{\text{low}}) \\
\hat{\alpha}_{\text{low}} &= \alpha_{\text{low}} + (y_{\text{low}} (b_{\text{low}} - b_{\text{high}})) / (\text{Kernel}(x_{\text{high}}, x_{\text{high}}) \\
&+ \text{Kernel}(x_{\text{low}}, x_{\text{low}}) - 2 \text{Kernel}(x_{\text{high}}, x_{\text{low}}) \\
\hat{f}_i &= \hat{f}_i + (\hat{\alpha}_{\text{high}} - \alpha_{\text{high}}) y_{\text{high}} \text{Kernel}(X_{\text{high}}, X_i) \\
&+ (\hat{\alpha}_{\text{low}} - \alpha_{\text{low}}) y_{\text{low}} \text{Kernel}(X_{\text{low}}, X_i) \\
I_{\text{high}} &= \{i: 0 < \alpha_i < C\} \cup \{i: y_i > 0, \alpha_i = 0\} \\
&\cup \{i: y_i < 0, \alpha_i = C\} \\
I_{\text{low}} &= \{i: 0 < \alpha_i < C\} \cup \{i: y_i > 0, \alpha_i = C\} \\
&\cup \{i: y_i < 0, \alpha_i = 0\} \\
i_{\text{high}} &= \arg\min\{f_i : i \in I_{\text{high}}\} \\
i_{\text{low}} &= \arg\max\{f_i : i \in I_{\text{low}}\} \\
b_{\text{high}} &= \min\{f_i : i \in I_{\text{high}}\} \\
b_{\text{low}} &= \max\{f_i : i \in I_{\text{low}}\}.
\end{align*}
$$

B. Multiclass SVM Classification

For a binary-class SVM classification problem, when we predict the label for a test sample $x$, we need to conduct the kernel evaluation for $x$ with all support vectors. Then we can predict the label for $x$ according to (13). If $g(x) > 0$, the predicted label of test sample $x$ is +1; else, the predicted label of test sample $x$ is $-1$.

$$
g(x) = \sum_{j \in SV} \alpha_j \text{Kernel}(x_j, x) + b.
$$

For a multiclass SVM classification problem, the model file obtained from SVM training phase contains parameters (such as $b$, support vectors, and $\alpha_i$ corresponding to each sample, etc.) of $n_{\text{class}} \times (n_{\text{class}} - 1)/2$ binary-class SVM problems. The brief outline of the multiclass SVM classification is shown in Algorithm 2, in which step 4 is conducted for each binary-class SVM problem.

$$
g(x) = \sum_{j \in SV_{C_1}} \alpha_j \text{Kernel}(x_j, x)
+ \sum_{k \in SV_{C_2}} \alpha_k \text{Kernel}(x_k, x) + b.
$$

III. DESIGN OF MMSVM

In this section, we will describe the general flow and several optimization strategies for MMSVM proposed in this research on the Intel Ivy Bridge CPUs and the Intel Xeon Phi coprocessor (MIC). We employ similar optimization techniques for the Ivy Bridge CPUs and MIC as their architectures share many similarities. Moreover, the code of our MMSVM on MIC is very similar to the code of MMSVM on the Ivy Bridge CPUs. For
MMSVM training phase, we focus on providing various optimization strategies to parallelize and accelerate SMO algorithm based on features of modern multicores and many-core architectures. The general flow of our proposed MMSVM algorithm will be discussed in Section III-A. The optimization details of MMSVM will be discussed from Section III-B to III-D.

A. General Flow of MMSVM Training and Classification

Algorithm 3 shows the general flow for our proposed parallel SMO algorithm in the training phase of MMSVM, where $T$ is the number of hardware threads, $t$ is the thread ID, and $F$, $Y$ are the vector forms of $f_i$ and $y_i$ $(\forall i \in \{1, 2, \ldots, n/T\})$, respectively. Algorithm 4 shows the general flow for parallel classification phase of MMSVM, where $nt$ is the total number of test samples. The optimization strategies of the algorithms (denoted by bold type) are listed in brief and will be described in detail in the following sections.

B. Adaptive Heuristic Supports for Input Datasets

Similar to the datasets used in other application domain (e.g., text classification, handwritten recognition, and image classification in computer vision, etc.), there are different types of datasets in remote sensing domain as well, including the dense datasets and sparse datasets (described in detail in Section IV-A). As mentioned in our previous work [28], many previous SVM tools apply one specific data processing approach for different types of input datasets. However, this strategy may lead to some problems such as limiting the parallelism (applying sparse format, e.g., in LIBSVM [23]) or increasing the memory requirement (applying dense format, e.g., in binary-class GPUSVM [27]).

In order to solve the above-mentioned problems in previous SVM tools, our MMSVM provide adaptive heuristic supports for handling different types of remote sensing datasets. Based on our previous work in MIC-SVM [28], we build a LIBSVM-based classification model from the information of training $N (N \geq 100)$ datasets covering a variety of real-world use-cases. Users have the flexibility to add new training samples to update the classification model. For each dataset, the training time of SVM required by the sparse method is denoted by $t_1$, and the training time of SVM required by the dense method is denoted by $t_2$. The training of each dataset is based on the default parameters of LIBSVM, of which the classification accuracies are close to the accuracies obtained in other articles (the difference is less than 3%). The input features of the...
algorithm include the number of nonzero elements, the number of samples, the number of features, the variance of the number of nonzero elements in each sample, the number of cores, the memory size, and the single instruction multiple data (SIMD) width. We choose these features because they have a great influence on the performance of MMSVM when using dense method or the sparse method. The output of the classifier for one test sample is the predicted label of test sample. If the kernel is vectorized, then do the kernel evaluation in each hardware thread concurrently through vectorization; If the kernel is un-vectorized, then do the kernel evaluation in each hardware thread serially.

5: Calculate $g(x)$ according to (14) for each binary-class problems. If $g(x) > 0$, vote(Class$_i$)++; else, vote(Class$_j$)++. After finishing $g(x)$ calculation for all binary-class problems, the index of the class with the maximum vote is the predicted label of test sample $x$.

6: Conduct Step 4 and Step 5 to predict labels for all test samples $x_i(\forall i \in \{1, 2, \ldots, nt\})$ concurrently through parallel for.

### C. Task Parallelism and Data Parallelism

Our experimental architectures (MIC and Ivy Bridge CPUs) employ two level parallelism: task and data parallelism. For MIC, the data parallelism benefits from on-core vector processing unit (VPU) and SIMD. For both Ivy Bridge and MIC, the task parallelism is achieved by utilizing multiple hardware threads. It is crucial to make full use of the two-level parallelism and improve the occupancy rate of the computing resources in order to get satisfactory performance.

1) Algorithm and Architecture Analysis: As mentioned in MIC-SVM [28], the update of each $f_i$ [see (6)] at each step is the most time consuming phase for the SMO algorithm because it requires to access all the training samples. Fortunately, for any pair of $i, j \in \{1, 2, \ldots, n\}$, the updates of $f_i$ and $f_j$ are independent of each other. Similarly, the kernel evaluation of a given element with $X_{i, j}$ is independent from the kernel evaluation of any other element with $X_{i, j}$. It is appropriate to employ task parallelism for kernel evaluation because of its independent computation, independent memory requirement, and coarse parallel granularity.

In order to optimize the parallel SMO algorithm in MMSVM effectively, we need to analyze the gap between characteristics of algorithm and the underlying architectures. We use the ratio of computation to memory access (RCMA) to describe characteristics of SMO algorithm, which can be defined as (15). The RCMA of SMO in our MMSVM is about 1.5 for processing single precision Gaussian kernels (theoretical lower bound), of which the computation process has been discussed in the MIC-SVM paper [28]

$$RCMA = \frac{\text{number of comp. flops}}{\text{number of memory access bytes}}$$

Similarly, we use the ratio of peak computation to peak memory bandwidth (RCMB) to describe the theoretical architectural bound, which can be defined as

$$RCMB = \frac{\text{theoretical peak performance}}{\text{theoretical bandwidth}}$$

The RCMB values of our experiment architectures used in this research are shown in Table II. We can find that the RCMA of SMO in our MMSVM for the single precision case (1.5) is lower than all the RCMB values of the evaluated architectures. This indicates that the limited memory bandwidth of the evaluated architectures may not match the high computational power required for accelerating the parallel SMO. Therefore, it is necessary to improve data reuse in order to reduce the impact of the bandwidth constraint.

2) Number of Threads: The configuration of the proper thread number is an important technique to improve task parallelism. For both Ivy Bridge CPUs and MIC, each independent hardware device owns a specific amount of physical resources, which provides an inherent task parallelism. In order to take full advantage of this mechanism, a practical way is to set the number of threads according to the number of physical resources in each device. Taking MIC as an example, according to Figs. 1 and 2, our MMSVM shows decent strong scalability for both training and classification. The speedup over LIBSVM denotes the relative performance of the major computation time (the SMO algorithm in parallel for).
training and the kernel evaluation in classification) of MMSVM to the corresponding major computation time of LIBSVM. For both training and classification, the speedup of MMSVM improves greatly when the number of threads increases from 10 to 60, whereas it does not have a significant improvement when the number of threads increases from 60 to 180 and it decreases when the number of threads increases from 180 to 240, due to the limitation of physical resource and the memory contention from multithreading.

3) Other Optimization Strategies: Besides identifying and adopting a proper number of thread, we apply several other optimization strategies to further improve the performance of our MMSVM. First, we use affinity models to facilitate load balancing. The affinity models provide different ways to allocate the virtual threads to the proper cores for better performance and resource utilization. For the real-world remote sensing datasets in our research, which will be described in detail in Section IV, the running efficiency of using balanced mode is slightly better than using compact mode and scatter mode. Therefore, we use the balanced mode for our MMSVM on the Intel Xeon Phi and the Ivy Bridge CPUs, in which tasks can be decomposed into small portions and distributed across all physical devices evenly.

Second, we employ the OpenMP [32] SCHEDULE technique (static type, default chunk size) to distribute all training samples to all hardware threads evenly, as the compute intensity of each training sample in a binary-class problem is the same according to Section II. Along with prefetching, this method not only eliminates the overheads of data communication between different hardware threads through avoiding scatter and gather operation, but also takes full advantage of abundant caches provided by the Ivy Bridge CPUs and the Intel MIC. In this way, we can effectively reduce the gap between the algorithmic RCMA and architectural RCMB through data reuse.

Third, to achieve data parallelism within each hardware thread for the data-wise operations in individual kernel evaluation, we apply SIMD mechanism and VPU for more efficient vectorization. We use the Cilk [33] array notation to achieve explicit data parallelism instead of applying the implicit compiler model, which helps to unleash vectorization by providing context information. We also apply memory alignment so that vectorization can use aligned load instructions.

D. Hierarchical Parallelism for MMSVM Training on MIC

Besides the task and data parallelism, we design and implement a CPU-MIC hierarchical parallelism for multiclass SVM training. Algorithm 5 shows the general flow of this strategy. The definitions of the variables are the same as those in Section II-A. For the outer loop, the number of iterations equals to the number of binary-class problems. We use the OpenMP schedule technique of dynamic type to facilitate load balancing because in remote sensing domain the number of samples in each classes often differs greatly. For the inner loop, the number of iterations equals to the number of samples in the corresponding binary-class problem (denoted by \( n_{\text{Samples}} \)). We use OpenMP schedule technique of static type because the compute intensity of each sample is the same, which is mentioned previously in Section III-C3. The number of threads for the outer loop (denoted by \( n_{\text{threads}_\text{cpu}} \)) and the number of threads for the inner loop (denoted by \( n_{\text{threads}_\text{mic}} \)) are defined by the user.

Algorithm 5: Hierarchical Parallelism for MMSVM Training.

```c
#pragma omp parallel for schedule(dynamic) num_threads(n_threads_cpu)
for i ∈ \{0, \ldots, n_{\text{classes}} \times (n_{\text{classes}} - 1)/2\} do
  Initialize \( \alpha_{\text{high}}, \alpha_{\text{low}} \) and \( F_i \), etc.
  #pragma offload target(mic:1)
  while \( b_{\text{low}} > b_{\text{high}} + 2 \times \text{tolerance} \) do
    #pragma omp parallel for schedule(static) num_threads(n_threads_mic)
    for j ∈ \{1, \ldots, n_{\text{Samples}}\} do
      Compute updated \( F_i \).
    end for
    Compute updated \( \alpha_{\text{high}} \) and \( \alpha_{\text{low}} \).
    Execute other essential operations.
  end while
  Execute other essential operations.
end for
```
Fig. 3 shows the speedups over LIBSVM obtained from this strategy (CPU-MIC parallelism) with the one obtained from only using the openmp parallel for inner loop (MIC parallelism) for each remote sensing dataset. The characteristics of each dataset will be described in detail in Section IV-A. The speedups of each dataset shown in Fig. 3 are the highest speedups obtained from each case. For MIC parallelism, the number of MIC threads $n_{threads}$ for the highest speedup is 120 for Covtype, MODIS, and airborne visible infrared imaging spectrometer (AVIRIS) datasets and 180 for African land cover (ALC), AVIRIS_n8, hyperspectral digital imagery collection experiment (HYDICE), and Palm datasets. For CPU-MIC parallelism, the number of CPU threads and MIC threads (denoted by $n_{threads_cpu}/n_{threads}$) for each dataset are 12/20, 4/60, 4/60, 6/40, 3/80, 3/80, and 6/40 respectively. We can find that the speedups achieved from CPU-MIC parallelism are higher than those achieved from MIC parallelism for all datasets.

There are several reasons for these results. First, as mentioned in Section III-C1, the SVM training on MIC is a memory bound problem. If we only use a great number of MIC threads, sometimes the MIC cores will be idle and waiting for data transfer. Second, there are some operations such as the variable initialization between the inner and the outer loops. If we use the CPU-MIC hierarchical parallelism, the variable initialization operations of one binary-class problem can be executed in parallel with the operations in the inner loop of another binary-class problem. Similarly, the variable initialization operations of different binary-class problems can be executed in parallel as well. Moreover, this might also be related with the Cache mechanism of MIC. Through utilizing the CPU-MIC hierarchical parallelism strategy, the speedups of MMSVM training can be improved by 1.05 to 1.88 times compared with only using MIC parallelism.

IV. Experimental Results and Analysis

A. Experimental Setup and Datasets

The architecture details of the experimental platforms used to evaluate the proposed algorithm can be found in Tables II and III. Our proposed algorithms are evaluated on both MIC (KNC 5110P) and the Ivy Bridge CPUs (Intel Xeon CPU E5-2697) architectures. The Ivy Bridge system used in this study has two CPUs and each CPU comprises 12 cores. It is a two-socket system based on QPI bus. All the 24 cores are in the same node and share the memory. For the purpose of cross-platform performance evaluation, we also compare the performance of our proposed algorithms with a multiclass GPU-SVM [25], which is tested on NVIDIA Tesla K40 GPU.

Several representative real-world remote sensing datasets are selected for evaluating the performance of each architecture, including three datasets for land cover classification (ALC, Covtype, and MODIS), three hyperspectral datasets (AVIRIS, AVIRIS_n8, and HYDICE), and one dataset for object-based classification (Palm). The characteristics of these datasets are shown in Table IV, where $n_{Train}$ is the number of training samples, $n_{Test}$ is the number of test samples, $n_{Classes}$ is the number of classes, $n_{Dim}$ is the number of features of each sample, and density is the ratio of the number of nonzero elements to the total number of elements.

The ALC [34] is a remote sensing dataset for large-scale land cover mapping, of which the training and test samples cover the whole African continent. The samples are collected by human interpretation with reference to Landsat imagery (nominal year 2014), time series of MODIS imagery (2014), high-resolution images (latest) from Google Earth, and temperature/precipitation (2014) for five seasons. Those samples represent eight land cover types (farmland, forest, grassland, shrubland, water, impervious, bareland, and snow/ice) and a cloud type. For each sample, eight features are selected for classification: Landsat spectral reflectance (bands 1–5 and band 7 for Landsat 5/7 TM (Thematic Mapper)/ETM+ (Enhanced TM...
Plus), bands 2–7 for Landsat 8 operational land imager), NDVI (normalized difference vegetation index), and modified normalized difference water index [35].

The Covtype [36] is a sparse dataset for forest cover type classification, which was taken from the UCI machine learning repository. The dataset consists of 5 81 012 samples of 30 m × 30 m patches of forest located in Roosevelt National Forest in northern Colorado. Each data sample includes 54 attributes: elevation (in meters), slope, aspect (compass direction of slope face), vertical distance to nearest surface water features, horizontal distance to nearest surface water features, wildfire ignition points, and roadway, sunlight intensity at 9 am, 12 pm, and 3 pm, 4 binary wilderness area designators, and 40 binary soil type designators. Each sample is classified into one of seven forest cover types: spruce/fir, lodgepole pine, ponderosa pine, cottonwood/willow, aspen, Douglas fir, or krummholz.

The MODIS tile H28V08 image is another sparse dataset for land cover classification. The dataset contains 90 000 samples selected from the time-series MODIS imagery. Each sample of the dataset has 161 features that consists of seven selected bands (Reflectance of middle infrared, blue, near infrared, and red; vegetation index quality, enhanced vegetation index, and NDVI) of 23 time-series MODIS images (obtained every 16 days in 2014). Sometimes the value of some features can be invalid due to the effect of cloud. We use the 250-m resolution land cover map of the finer resolution observation and monitoring of global land cover [1] as the ground truth of this dataset. Each sample is classified into one of eight land cover types: farmland, forest, grassland, shrubland, wetland, water, impervious, or bareland.

The AVIRIS data [37] is a 224-band hyperspectral dataset that covers Salinas Valley, California with high spatial resolution (3.7 m). We reduce the number of bands to 204 by removing 20 water absorption bands. The ground truth of AVIRIS dataset contains 16 classes including vegetables, bare soils, and vineyard fields. As many studies show that spatial information is useful for improving the classification accuracy for hyperspectral data [38], [39], we enlarge the feature dimension of each sample through adding the features of its eight neighbor pixels and obtain the AVIRIS_n8 dataset with 1836 features.

The HYDICE data [40] is another widely used hyperspectral dataset that was collected in October 1995 for the urban area at Copperas Cove, TX, U.S. This hyperspectral image has 307 × 307 pixels and can be classified into six land cover types including asphalt, grass, tree, roof, metal, and dirt. After removing several bands due to dense water vapor or atmospheric effects and including spatial information in the same way as AVIRIS dataset, we obtain the current HYDICE dataset with 1458 features.

The Palm dataset [41] was collected from the QuickBird high-resolution image acquired on November 21, 2006 with a size of 12, 188 × 12, 576 pixels, which is located in the south of Malaysia. The 34 000 samples were collected through human interpretation from several selected regions in the study area and randomly divided into 28 000 training samples and 6000 test samples. These samples can be classified into four types: oil palm tree, background, other vegetation/bare land, and impervious/cloud. Each sample corresponds to a square of 17 × 17 pixels with three bands (red, green, and blue) selected from the original four bands, resulting in the 867 features in total.

### B. Classification Accuracy Assessment

In our experiment, we use the Gaussian Kernel for training and classification because it is the most widely used kernel function. The main parameters of Gaussian Kernel for each dataset are shown in Table V. The parameter settings of ALC dataset are the same as those used in our previous research [34]. The parameter settings of other datasets are selected in a reasonable value range without optimization [42], [43]. To evaluate the correctness of our MMSVM, we select the LIBSVM [23] (a popular standard SVM library) as our benchmark. We use the LIBSVM-3.20 that runs on the Intel Xeon CPU E5645. The classification accuracies of each platform based on our test datasets are shown in Table VI. We can observe that there are only small discrepancies (less than 0.02%) between our MMSVM (on MIC and CPUs) and the standard LIBSVM. The discrepancies between GPUSVM and LIBSVM are much greater due to the differences in the implementations.

Moreover, we compare the classification maps of AVIRIS scene obtained from each platform to further validate the correctness of our implementations. Fig. 4 displays the false color image and the ground truth of AVIRIS hyperspectral image. Figs. 5 and 6 show the classification maps based on AVIRIS dataset and AVIRIS_n8 dataset, respectively. The overall classification results in Fig. 6 are better because the AVIRIS_n8 dataset use both spectral and spatial features for classification. We can observe that our implementations (on MIC and CPUs) can obtain the same classification results as those obtained from the standard LIBSVM, while the discrepancies of classification maps between GPUSVM and LIBSVM are visible in some regions.
Fig. 4. AVIRIS hyperspectral image. (a) False color composition. (b) Ground truth.

Fig. 5. Classification maps based on AVIRIS dataset. (a) LIBSVM. (b) MIC. (c) CPUs. (d) GPU.

Fig. 6. Classification maps based on AVIRIS_n8 dataset. (a) LIBSVM. (b) MIC. (c) CPUs. (d) GPU.

C. Speedup and Cross-Platform Analysis

In this section, a cross-platform performance comparison is conducted to analyze the characteristics of each architecture. The architecture details of the experimental platforms can be found in Sections IV-A and IV-B. Tables VII and VIII show the training time and classification time of different implementations, respectively, each of which refers to the total running time including some serial operations (e.g., file reading and writing, multiclass dataset grouping, etc.) and the parallel core operations (e.g., the SMO algorithm in training or the kernel evaluation in classification). Figs. 7 and 8 show the speedups over LIBSVM in training and classification of different implementations. We can observe that our proposed MMSVM achieves 6.3–31.1 \times \text{(in training)} \text{ and } 4.9–32.2 \times \text{(in classification)} speedups over the serial LIBSVM on Intel Knights Corner MIC, and 6.9–14.9 \times \text{(in training)} \text{ and } 5.5–22.1 \times \text{(in classification)} speedups over the serial LIBSVM on the two Ivy Bridge CPUs.

The speedups depend on many factors including the implementation, the architecture, and the input data pattern, etc. Based
on the speedups in Figs. 7 and 8 and the experimental results of our previous MIC-SVM library, we conduct the following analysis on how to select the most suitable implementation and architecture for specific input data patterns or remote sensing classification problems.

Comparing MMSVM on MIC with MMSVM on the Ivy Bridge CPUs, we can find that MMSVM on MIC outperforms MMSVM on the Ivy Bridge CPUs for AVIRIS_n8 (1836d), HYDICE (1458d), and Palm (867d) datasets, while it performs worse than MMSVM on Ivy Bridge CPUs for ALC (9d), Covtype (54d), MODIS(161d), and AVIRIS (204d) datasets. In addition, we adjust the number of features of the Palm dataset from 200 to 800 and record the corresponding running time of each platform. Tables IX and X show the speedups of Palm dataset over LIBSVM on MIC, Ivy Bridge CPUs, and GPU in training and classification. Experimental results show that the MMSVM on MIC outperforms the MMSVM on the Ivy Bridge CPUs when the number of features exceeds 600. We can conclude that our MMSVM on MIC is more suitable for high-dimensional datasets (e.g., the number of features is higher than 600), whereas our MMSVM on the Ivy Bridge CPUs are more suitable for low-dimensional datasets. The reasons for these results are as follows.

For dense datasets, MMSVM automatically applies data parallelism to process each kernel evaluation through vectorization (SIMD). Ivy Bridge CPUs are based on advanced vector extensions instruction set, which has a 256-bit SIMD register file. However, the SIMD width (512 bits) of knights corner MIC is twice of that on the Ivy Bridge CPUs. Therefore, high-dimensional dense datasets are more likely to benefit from the powerful vectorization scheme on MIC than on the Ivy Bridge CPUs.

Comparing our MMSVM with the GPUSVM used in this study, we can conclude that the GPUSVM used in this study is more suitable for sparse datasets and the datasets with a large number of samples. On one hand, different from other GPUSVM solvers, the GPUSVM used in this study takes advantages of the sparsity in the training set through a novel sparsity clustering approach so that it can handle sparse datasets (e.g., the Covtype dataset and MODIS dataset) cleanly and efficiently. On the other hand, for datasets with a large number of samples (e.g., ALC dataset and Covtype dataset), the large number of samples and the large number of support vectors in SVM model file can provide enough parallelism for millions of fine-grained GPU lightweight threads without suffering from low data-level parallelism.

V. CONCLUSION

In this paper, we design and implement a highly efficient multiclass SVM for \times 86-based multicore and many-core architectures such as the Ivy Bridge CPUs and the Intel Xeon Phi co-processor (MIC) in order to improve the efficiency of SVM in both training and classification phases. Various analysis methods and optimization strategies are employed to fully utilize the multilevel parallelism provided by the studied architectures. Experimental results demonstrate the effectiveness of our proposed method concerning both efficiency and classification accuracy, based on several real-world remote sensing datasets. Moreover, we conduct a cross-platform performance comparison analysis and provide insights on how to select the most suitable implementation and architecture for specific input data patterns or remote sensing classification problems to achieve the best performance. In our future research, we will develop more efficient SVM on distributed platforms. We also plan to design and implement other remote sensing classification algorithms on various high-performance platforms.

REFERENCES


Weijia Li received the Bachelor's degree in information security from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2014. She is currently working toward the Ph.D. degree in ecology from the Department of Earth System Science, Tsinghua University, Beijing, China.

Her research interests include remote sensing image processing, parallel computing, machine learning, and deep learning.
Haohuan Fu (M’07) received the Ph.D. degree in computing from Imperial College London, London, U.K. He is an Associate Professor in the Ministry of Education Key Laboratory for Earth System Modeling and the Department of Earth System Science, Tsinghua University, Beijing, China. He is also the Deputy Director of the National Supercomputing Center, Wuxi, China. His research interests include design methodologies for highly efficient and highly scalable simulation applications that can take advantage of emerging multi-core, many-core, and reconfigurable architectures, and make full utilization of current Peta-Flops and future Exa-Flops supercomputers; and intelligent data Management, analysis, and data Mining platforms that combine the statistic methods and machine learning technologies.

Yang You received the Master’s Degree in computer science from Tsinghua University, Beijing, China, in July 2015. He is currently working toward the Ph.D. degree in computer science from the Computer Science Division, University of California, Berkeley, CA, USA. He is a Siebel Scholar. His research interests include parallel computing, distributed systems, and machine learning. In general, Yang You is working on the intersection between high performance computing and machine learning. He received IPDPS 2015 Best Paper award.

Le Yu received the B.Sc. degree in geographic information systems in 2005, and the Ph.D. degree in remote sensing in 2010, both from the School of Earth Sciences, Zhejiang University, Hangzhou, China. He is currently an Associate Professor in the Department of Earth System Science, Tsinghua University, Beijing, China. His research interests include global land cover/land use mapping, global cropland mapping, regional, and global land change monitoring and modeling for sustainable development. He has published extensively in the field of remote sensing image processing and classification more than 30 journal articles.

Jiarui Fang received the Bachelor’s degree in computer science from the Department of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China, in 2014. He is currently working toward the Ph.D. degree in computer science from the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include parallel computing, geophysics, and earth system science.