Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 252979, 13 pages
doi:10.1155/2012/252979

Research Article

Accelerating Relevance-Vector-Machine-Based Classification of Hyperspectral Image with Parallel Computing

Chao Dong and Lianfang Tian

Key Laboratory of Autonomous Systems and Network Control of the Ministry of Education, School of Automation Science and Engineering, South China University of Technology, Wushan Road No.381, Tianhe District, Guangzhou 510641, China

Correspondence should be addressed to Chao Dong, dcauto@scut.edu.cn

Received 7 December 2011; Revised 23 February 2012; Accepted 23 February 2012

Academic Editor: Jyh Horng Chou

Benefiting from the kernel skill and the sparse property, the relevance vector machine (RVM) could acquire a sparse solution, with an equivalent generalization ability compared with the support vector machine. The sparse property requires much less time in the prediction, making RVM potential in classifying the large-scale hyperspectral image. However, RVM is not widespread influenced by its slow training procedure. To solve the problem, the classification of the hyperspectral image using RVM is accelerated by the parallel computing technique in this paper. The parallelization is revealed from the aspects of the multiclass strategy, the ensemble of multiple weak classifiers, and the matrix operations. The parallel RVMs are implemented using the C language plus the parallel functions of the linear algebra packages and the message passing interface library. The proposed methods are evaluated by the AVIRIS Indian Pines data set on the Beowulf cluster and the multicore platforms. It shows that the parallel RVMs accelerate the training procedure obviously.

1. Introduction

Accompanying the rich spectral information that the hyperspectral imager collects, huge amount of training samples are required to train the classifier precisely. The problem is well known as the Hughes phenomena [1]. Collecting enough training samples is burdensome. Therefore, designing the hyperspectral image classifiers that can deal with the small training set became the theme in the past ten years. The solutions [2–10] can be divided into four categories: (1) regularization of the sample covariance matrix; (2) feature extraction or feature selection; (3) enlarging the training set by semisupervised learning; and (4) low complexity classifiers, such as support vector machine (SVM). Benefiting from the kernel skills, SVM is less affected by the Hughes phenomena. Maximizing the margins of the class pairs guarantees

SVM low training error and good generalization ability. It has been proven that SVM is superior to most supervised classifiers in classifying the hyperspectral image [7–10].

Sparse Bayesian learning-based classifiers, represented by relevance vector machine (RVM), emerged in the remote sensing community since 2006 [11–13]. To avoid overfitting, RVM constrains the predicting model by the automation relevance determination (ARD) framework, which promises a sparse model. Compared with SVM, RVM could acquire a much sparser model with equivalent generalization ability. The sparse property is competitive in classifying the large-scale hyperspectral image, as it requires far less time for prediction.

However, RVM is not widespread, due to its slow training procedure. In each iterative step, RVM carries out transpose, multiplication, and inversion operations on an $N \times N$ Hessian matrix, where N is the number of training samples. These operations are time consuming when N is large, making it inefficient for the large-scale data set. To solve the problem, Tipping and Faul proposed a fast marginal likelihood maximization method, refreshing one coefficient at one time [14]. However, the fast method performs greedy search and is easily stuck into suboptimal solutions. Lei et al. [15] avoided the expensive inversion of the Hessian matrix, by substituting Broyden-Fletcher-Goldfarb-Shanno (BFGS) for iteratively reweighed least squares (IRLSs). Seeger and Ribeiro applied RVM to the large text sets by the ensemble, boosting, and incremental methods, which adopt divide-conquer-merge strategy and enable RVM to process more than ten thousands training samples [16]. The divide-conquer-merge strategy decreases the amount of training samples each weak RVM processes, also helpful for accelerating the training procedure. Yang et al. proposed recursive Cholesky decomposition for RVM and implement it on GPUs. The GPUs-based RVM was proved to be much faster for both single and double precision [17].

Recent development of the multicore platform and the low-cost clusters makes parallel computing popularized in the hyperspectral remote sensing community. Plenty of the classification, unmixing and anomaly detection algorithms have been parallelized successfully [18–20]. Those cases motivate us to accelerate RVM with parallel computing. In this paper, we design three parallel implementations of RVM. The parallelization is revealed from the aspects of matrix operation, multiclassifier strategy, and divide-conquer-merge strategy. The parallel RVMs are tested on the multicore platform and the cluster, acquiring an obvious gain in efficiency.

2. Relevance Vector Machine Classifier

For the training samples $\{\mathbf{x}_n\}_{n=1}^N$ and the class labels $\{t_n\}_{n=1}^N$, RVM uses the linear combination of the kernel functions $K(\cdot,\cdot)$ to describe the input-to-output relationship

$$y(\mathbf{x};\boldsymbol{\omega}) = \sum_{n=1}^{N} \omega_n K(\mathbf{x},\mathbf{x}_n) + \omega_0, \tag{2.1}$$

and the Bernoulli distribution to construct the probability density function

$$p(\mathbf{t} \mid \boldsymbol{\omega}) = \prod_{n=1}^{N} {y_n}^{t_n} (1-y_n)^{1-t_n}. \tag{2.2}$$

The symbols in (2.1) ~ (2.2) are $\boldsymbol{\omega} = (\omega_0, \ldots, \omega_N)^T$, $\mathbf{t} = (t_1, \ldots, t_N)^T$, and $y_n = \sigma\{y(\mathbf{x}_n; \boldsymbol{\omega})\}$. $\sigma(y)$ is the sigmoid function

$$\sigma(y) = \frac{1}{(1 + e^{-y})}, \tag{2.3}$$

mapping $y(\mathbf{x}_n; \boldsymbol{\omega})$ into $[0, 1]$. To ensure the generalization ability, the weights $\boldsymbol{\omega}$ are constrained by

$$p(\boldsymbol{\omega} \mid \boldsymbol{\alpha}) = \prod_{n=0}^{N} N\left(\omega_n \mid 0, \alpha_n^{-1}\right). \tag{2.4}$$

Then, the posterior probability density function $p(\boldsymbol{\omega} \mid \mathbf{t}, \boldsymbol{\alpha})$ can be obtained by the Bayes' rule

$$p(\boldsymbol{\omega} \mid \mathbf{t}, \boldsymbol{\alpha}) = \frac{p(\mathbf{t} \mid \boldsymbol{\omega}) p(\boldsymbol{\omega} \mid \boldsymbol{\alpha})}{p(\mathbf{t} \mid \boldsymbol{\alpha})}. \tag{2.5}$$

Maximizing (2.5), the optimized $\boldsymbol{\omega}$ can be found as follows.

Serial Binary RVM Classification

(1) Initialize $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$.

(2) Fix $\boldsymbol{\alpha}$ and update $\boldsymbol{\omega}$ with

$$\begin{aligned} \mathbf{g} &= \nabla_\omega \log p(\boldsymbol{\omega} \mid \mathbf{t}, \boldsymbol{\alpha})\big|_{\omega_{\mathrm{MP}}} = \boldsymbol{\Phi}^T(\mathbf{t} - \mathbf{y}) - \mathbf{A}\boldsymbol{\omega}, \\ \mathbf{H} &= \nabla_\omega \nabla_\omega \log p(\boldsymbol{\omega} \mid \mathbf{t}, \boldsymbol{\alpha})\big|_{\omega_{\mathrm{MP}}} = -\left(\boldsymbol{\Phi}^T \mathbf{B} \boldsymbol{\Phi} + \mathbf{A}\right), \end{aligned} \tag{2.6}$$

$$\boldsymbol{\omega}_{\mathrm{MP}}^{\mathrm{new}} \leftarrow \boldsymbol{\omega}_{\mathrm{MP}}^{\mathrm{old}} - \mathbf{H}^{-1}\mathbf{g}. \tag{2.7}$$

The details of $\boldsymbol{\Phi}$, $\mathbf{A}$, and $\mathbf{B}$ could be found in [11].

(3) Fix $\boldsymbol{\omega}$ and update $\boldsymbol{\alpha}$ with

$$\boldsymbol{\alpha}_n^{\mathrm{new}} = \frac{1 - \boldsymbol{\alpha}_n^{\mathrm{old}} \boldsymbol{\Sigma}_{nn}}{\left(\boldsymbol{\omega}_{\mathrm{MP}}^{\mathrm{new}}\right)_n^2}, \tag{2.8}$$

where $\boldsymbol{\Sigma} = -\mathbf{H}^{-1}$.

(4) Repeat step (2) ~ (3) until convergence.

(5) Classify the test samples with the estimated model.

3. Parallel Optimization

RVM is a binary classifier. To deal with the multiclass problem, either One Against One (OAO) or One Against ALL (OAA) should be used. The multiclass RVM consists of

multiple independent binary classifiers and could be processed by multiple processing units simultaneously. In [16], Seeger and Ribeiro applied RVM to the large-scale text set with the ensemble, boosting, and incremental methods. They divided the standard RVM into multiple independent local RVMs. The idea was adopted by us for parallelization in this paper. RVM could also be accelerated by parallelizing the expensive matrix operations in (2.6) ~ (2.8). More complicated parallelization could be realized by combing the aforementioned strategies.

The parallel RVMs in this letter focus on the training phase. The test phase is not emphasized due to two reasons. First, with an equivalent amount of training and test samples, the optimization of $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ is far slower than the prediction of the test samples. Secondly, even if tens of thousands of test samples are involved, the prediction of each test sample is always independent. This is a typical data parallel problem. It could be easily solved by dividing the test set into multiple subsets, predicted by multiple processing units simultaneously.

3.1. Parallelizing the Multiclass Strategy

RVM deals with the multiclass problem by OAO or OAA. OAO is preferred, as it processes less training samples in each class pair [12]. Suppose there are C classes; the multiclass RVM by OAO consists of $(C-1)C/2$ uncorrelated binary RVMs. This is a typical task parallel problem and we name the parallel implementation pRVM-MultiClass for short. Load balance must be carefully handled for pRVM-MultiClass. With OAO, the class pairs may differ greatly in the amount of the training samples. The varieties will cause great difference between the CPU time consumptions of the class pairs. Load balance could not be promised if the class pairs are evenly distributed in the parallel environment. To solve the problem, pRVM-MultiClass is organized in the master-slave model. All the class pairs reside in the master and wait to be sent to one idle slave. Each slave is a binary RVM. The master continuously sends the unprocessed class pairs to the idle slaves until the entire class pairs have been sent out. The slaves request new class pair from the master once it is idle. The results of the slaves are collected and synthesized by the master for the classification map. The detail of pRVM-MultiClass is given in Figure 1.

3.2. Parallelizing Multiple Weak RVMs

RVM is extended to process the large-scale text data sets with the incremental, boosting, and ensemble methods [16]. All these methods are based on the divide-conquer-merge strategy. The entire training set is split into multiple small subsets. Each subset is used to train a weak RVM and classify the test set. The weak RVMs cause a decrease in precision, compared with the serial RVM. The loss could be compensated by the integration methods, such as majority voting. The ensemble method has been proven to be superior to the rest [16]. Thus, it is used to construct the weak classifiers in this paper and we named the parallel implementation pRVM-Ensemble. Figure 2 shows the flow of pRVM-Ensemble. Each process randomly extracts p% training samples from each class to train a local RVM. The class labels of test set are inferred by the weak RVMs, respectively, and then enhanced using majority voting.

pRVM-Ensemble is influenced by two parameters, the sampling rate p and the number of weak classifiers M. Suppose that the training set consists of N samples and the serial RVM

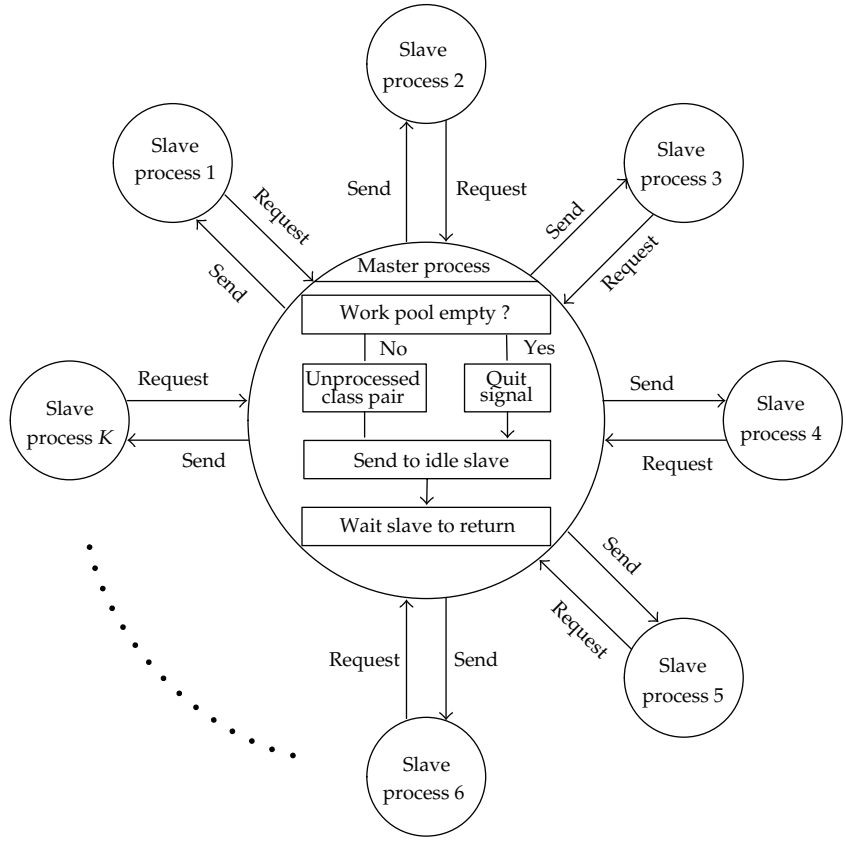


Figure 1: The flow of pRVM-MultiClass.

consumes T_s seconds. With large N, T_s is mainly occupied by the inversion of the Hessian matrix $\mathbf{H}$ in (2.7), who's complexity is $O(N^3)$. Ignoring the minor parts, the complexity of the weak RVM could be approximated by $O(p^3N^3)$. Therefore, the speedup ratio of pRVM-Ensemble is

$$r = \frac{T_s}{M \times p^3 T_s / K} = \frac{K}{p^3 \times M}, \tag{3.1}$$

where K is the number of processes. Small p and M guarantee high speedup ratio, at the cost of losing accuracy. On the contrary, the misclassifications are decreased with a relatively low efficiency. Suppose that M weak RVMs hold M copies of the entire training set and are distributed to M processes. In this extreme case, the efficiency and precision are not improved by pRVM-Ensemble. Fine tune of the parameters is necessary, to balance the trade-off between efficiency and precision.

3.3. Parallelizing the Matrix Operations

RVM is occupied by the matrix operations in (2.6) ~ (2.8), especially the expensive inversion of the Hessian matrix $\mathbf{H}$. It could be easily accelerated by substituting the parallel matrix

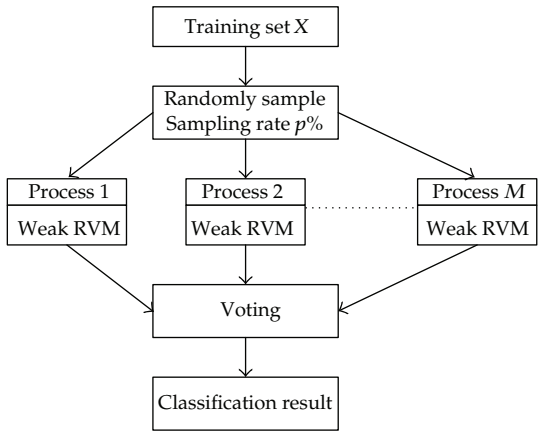
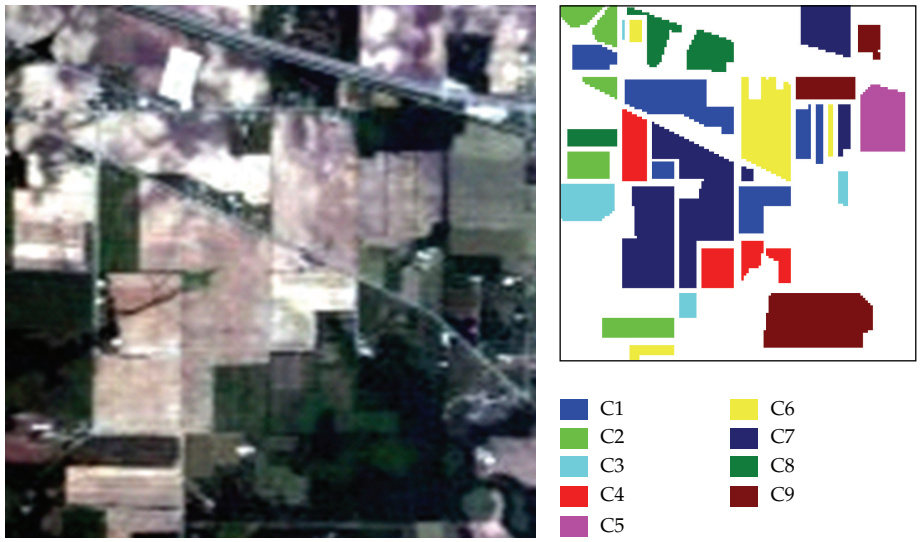


Figure 2: The flow of pRVM-Ensemble.

functions for the serial ones. We named the parallelization RVM-MatOp. The parallel functions have been realized in many parallel linear algebra packages, such as Intel's Math Kernel Library (MKL) and Automatically Tuned Linear Algebra Software (ATLAS). The packages provide optimized matrix multiplication and inversion functions for the large-scale matrices. Although not developed by us, RVM-MatOp is emphasized for two reasons. First, the parallel matrix functions are implicitly controlled by the well-developed packages. The researchers could easily implement pRVM-MatOp on the multicore platforms, even if not familiar with parallel computing. Second, pRVM-MatOp could be combined with pRVM-MultiClass and pRVM-Ensemble for better efficiency. This will be discussed in Section 3.4.

3.4. Hybrid Parallel RVM

The parallel implementations could be further optimized by combination. pRVM-Ensemble uses OAO to deal with the multiclass problem in each weak RVM. Combining the multiclass parallel strategy with the local weak RVMs can accelerate pRVM-Ensemble. In addition, pRVM-MultiClass and pRVM-Ensemble consist of multiple standard binary RVMs. They are also tortured by the expensive matrix operations in (2.6) ~ (2.8). Therefore, pRVM-MatOp could be combined with pRVM-MultiClass and pRVM-Ensemble to optimize the binary RVMs. More complex case is the combination of three parallel implementations. The ensemble skill is used to decompose the serial RVM into multiple independent subtasks, the multiclass parallel strategy optimizes the local weak RVMs, and the binary RVMs are accelerated by the parallel matrix functions. However, the efficiency might be decreased by too complex parallel strategy. The ideal case is the combination of pRVM-MatOp with pRVM-MultiClass or pRVM-Ensemble. RVM is first globally separated into multiple uncorrelated subtasks by the multiclass or ensemble skill. Then the parallel matrix functions are used to optimize the local subtasks. The hybrid structure is popular in parallel programming and suitable to be implemented on the cluster of multicore computers.



(a) Pseudo-color image, bands 24-12-5

(b) The distribution of nine typical classes

Figure 3: Pseudo-color image and classes distribution of the Indian Pine test site.

4. Experiments

To evaluate the proposed methods, we carried out several experiments on the data acquired by Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) over Indian Pines test site in 1992 [21]. The test site covers a 145 × 145 region and 220 bands ranging from 0.4 to 2.5 μm. It contains sixteen different land covers, some of which are hard to separate. Seven classes were discarded for the insufficient labeled samples. The rest nine classes had 8489 samples and were divided into the training and test sets. The details of the data set are in Figure 3 and Table 1.

The algorithms were implemented with the C language, based on the Matlab code of Tipping. The RBF kernel was adopted. The width of RBF was optimized by 5-fold cross-validation and grid search for the serial RVM. The best width was directly used in the parallel RVMs. We focused on the efficiency of RVMs with the best width and ignored the cost of optimizing the parameter.

4.1. Evaluating pRVMs on the Beowulf Cluster

First, we evaluated the parallel RVMs on a sixteen-node Beowulf cluster, running on the CentOS Linux operating system. The nodes are connected by the Gigabit Ethernet. Each node contains one 2 GHz AMD processor and 1 GB memory. pRVM-MultiClass and pRVM-Ensemble were explicitly parallelized with the message passing interface (MPI). The matrix functions are from Basic Linear Algebra Subprograms (BLASs) and Linear Algebra Package (LAPACK). The scalable LAPACK (ScaLPACK) library provided pRVM-MatOp with the parallel matrix functions in the cluster environment. For better performance, we compiled the parallel RVMs with the optimized BLAS package ATLAS.

Table 1: Number of samples in training and test sets.

Class	Name	Training	Test	Total
C1	Corn-notill	632	633	1265
C2	Corn-min	364	364	728
C3	Grass/Pasture	224	225	449
C4	Grass/Trees	335	336	671
C5	Hay-windrowed	228	228	456
C6	Soybeans-notill	425	424	849
C7	Soybeans-min	1133	1135	2268
C8	Soybeans-clean	289	288	577
C9	Woods	614	612	1226
	Total	4244	4245	8489

Table 2: The time cost (in seconds) of parallel RVMs on cluster, with entire training set.

CPUs	pRVM-Ensemble with different p			pRVM-Multiclass	pRVM-MatOp
	20%	40%	60%		
1	1720.1	9164.3	26510.1	3402.9	3402.9
2	867.5	4638.6	13320.2	1886.4	2705.9
4	449.8	2319.3	6811.2	1139.8	1413.6
8	235.7	1214.4	3409.9	834.6	737.5
16	121.1	616.6	1824.4	634.2	388.9

Table 2 shows the efficiency of the parallel RVMs. The serial RVM takes 3402.9 seconds. pRVM-MultiClass and pRVM-MatOp are equal to the serial RVM, when one CPU is involved. pRVM-Ensemble was tested with three different sampling rates, ranging from 20% to 60%. M was set 32 for the load balance purpose. The CPU time of pRVM-Ensemble could be estimated by $M \times p^3 \times T_s$, which is 871.1 seconds for $p = 20\%$, 6969.1 seconds for $p = 40\%$, and 23520.8 seconds for $p = 60\%$. The approximations are on the same order of magnitude as the measured values in Table 2. The time costs of the parallel RVMs decrease rapidly, when more CPUs are involved.

Figure 4 plots the speedup ratio curves of the parallel RVMs. Benefiting from the small sampling rate, pRVM-Ensemble acquires super linear speedup ratios when $p = 20\%$. The ratios drop beneath the ideal cases for the large p. The curve of pRVM-MultiClass descends as the number of CPUs increases. With less than 16 CPUs, pRVM-MultiClass is faster than pRVM-Ensemble ($p = 40\%$ and $p = 60\%$). However, it falls beneath pRVM-Ensemble ($p = 40\%$) with 16 CPUs. The phenomenon is caused by the unbalanced class pairs of the training set. Table 3 shows the time cost of training each class pair with the binary RVM. Classes 3 and 5 have the least number of samples. They only take 6.3 seconds to train a binary RVM. The features of class 7 and class 9 are quite similar. Additionally, they almost have the most samples. It takes 575.3 seconds to train a binary RVM for them. Influenced by the huge difference, the computing nodes could not acquire the same amount of tasks. The load unbalanced problem could be eased by the master-slave model in Figure 1. However, it could never be totally eliminated, especially when more CPUs are involved. pRVM-MatOp is more effective than pRVM-MultiClass and pRVM-Ensemble ($p = 40\%$ and $p = 60\%$). To implement the parallel matrix functions in the cluster, ScaLPACK introduces extra overhead

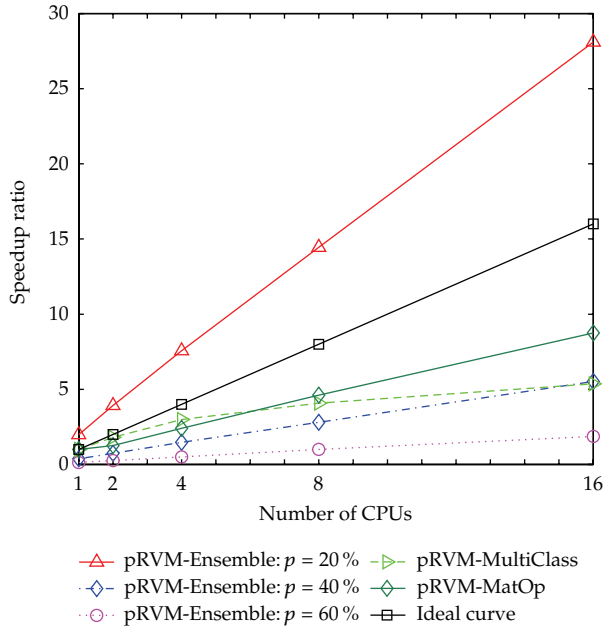


Figure 4: The speedup ratio curves of the parallel RVMs.

Table 3: The time cost of RVM for each class pair.

Class	2	3	4	5	6	7	8	9
1	51.2	37.7	50.7	38.8	63.5	269.1	45.8	284.9
2		13.7	35.9	15.1	26.7	178.9	17.3	189.5
3			11.3	6.3	17.5	148.6	9.1	36.1
4				12.4	23.2	163.0	15.3	41.4
5					19.6	163.6	11.1	102.4
6						197.1	20.4	249.5
7							141.6	575.3
8								119.3

on the network for data exchange, which causes a slight downtrend in the speedup ratio curve of pRVM-MatOp. The downtrend will further increase with more CPUs.

Table 4 lists the overall classification accuracy of the parallel RVMs. pRVM-MultiClass and pRVM-MatOp acquire the same accuracy as the serial, because they do not alter the logic of the standard RVM. The training subsets of the weak RVMs are randomly sampled in pRVM-Ensemble. It will cause the difference among the accuracies of the local classifiers. When p equals 20%, the minimum accuracy of the weak RVMs is 81.67% and the maximal value is 84.43%. There exists a gap of 2.76%. The gap is gradually decreased as the sampling rate p increases. The ensemble accuracy after the majority voting is better than those (Min, Max, and Aver) of the weak RVMs. It exceeds the serial when $p = 60\%$. The accuracies do not linearly increase with the sampling rate p for pRVM-Ensemble. The increment is 2.85%, when p varies from 20% to 40%. It drops to 0.73% for the 40%–60% case. The increment is further decreased to 0.19%, when p reaches 80%. However, it produces a huge growth of the time

Table 4: Overall classification accuracy of parallel RVMs on cluster, with entire training set.

p	pRVM-Ensemble				pRVM-MultiClass	pRVM-MatOp
	Min	Max	Aver	Vote		
20%	81.67%	84.43%	82.85%	87.94%	91.26%	91.26%
40%	86.31%	88.55%	87.43%	90.79%		
60%	88.36%	90.39%	89.28%	91.52%		
80%	89.52%	90.86%	90.29%	91.71%		

Table 5: Speedup ratios of parallel RVMs as data size varies, 16 CPUs involved.

Data set	Data size	pRVM-Ensemble with different p			pRVM-Multiclass	pRVM-MatOp
		20%	40%	60%		
DS-A	848 (20%)	2.96	1.55	1.13	5.7	2.3
DS-B	1697 (40%)	11.91	3.31	1.36	5.2	4.7
DS-C	2546 (60%)	17.05	4.04	1.46	5.3	6.4
DS-D	3395 (80%)	26.13	5.45	1.82	5.6	7.9
DS-E	4244 (100%)	28.10	5.52	1.87	5.4	8.8

cost with so large p. Taking into consideration both efficiency and accuracy, the $p = 20\%$ and $p = 40\%$ cases are preferred.

We also measure the performance of the parallel RVMs for different data sizes. Table 5 shows the speedup ratios of the parallel implementations, when 16 CPUs are involved. The second column of the table gives the amount of samples used to train the classifiers. The number in the parenthesis is the percentage of the used training samples, compared with the entire training set in Table 1. The cost of the serial RVM does not linearly scale with the number of training samples. Therefore, the speedup ratio of pRVM-Ensemble is on a declining curve when the training set decreases. pRVM-MultiClass is stable when the data size varies. The speedup ratio of pRVM-MatOp decreases obviously when less training samples are used. It drops to 2.3 for the smallest data set DS-A. The reduction could be explained from three aspects. First, the parallel matrix functions used in pRVM-MatOp are designed for the large matrix, not suitable for too small data set. Second, the extra overhead caused by the parallelization has a prominently negative impact for the small training set. Third, pRVM-MatOp only parallelizes the matrix operations in (2.6) ~ (2.8). For the small data sets, the unparallelized parts could not be neglected. These factors greatly reduce the efficiency of pRVM-MatOp, once the data size decreases. The experiment indicates that pRVM-MultiClass is basically immune to the data size, but pRVM-Ensemble and pRVM-MatOp are not suitable for too small data sets.

4.2. Evaluate pRVM-MatOp on Multicore Platform

pRVM-MatOp was also tested on two multicore platforms. One is a server with a two-core Intel E5500 processor and 2 GB memory. The other is a workstation with a four-core Intel Q9400 CPU and 4 GB memory. pRVM-MatOp is complied and linked with Intel's complier under Visual Studio 2008. The parallel matrix functions come from the MKL package. The results are given in Table 6.

Table 6: The cost (seconds) and speedup ratio (in parenthesis) of pRVM-MatOp on multicore platforms.

Data set	Data size	Two-core server		Four-core workstation	
		1 Core	2 Cores	1 Core	4 Cores
DS-A	848 (20%)	21.5	16.5 (1.30)	22.7	15.2 (1.49)
DS-B	1697 (40%)	147.4	113.9 (1.29)	152.8	86.3 (1.77)
DS-C	2546 (60%)	409.7	312.3 (1.31)	427.4	207.5 (2.06)
DS-D	3395 (80%)	832.5	634.1 (1.31)	868.5	366.5 (2.37)
DS-E	4244 (100%)	1354.9	1028.3 (1.32)	1413.5	559.3 (2.53)

Like the experiment of Table 5, we also extracted different amounts of samples from the entire training set to assess the parallel matrix functions. The time costs with one core are equal to those of the serial RVM on the platforms. As the data size decreases, the reduction of the speedup ratio is not significant for the two-core platform. This is due to the too few cores. However, on the four-core platform, more processing units are involved. The speedup ratio decreases dramatically when the training set is small. This could also be explained from the aforementioned three factors. For the large training set, the cost is decreased significantly. Learning with the entire training samples, RVM is improved from 1354.9 seconds to 1028.3 seconds on two-core E5500 server. It acquires a 1.32 speedup ratio. The time is further decreased from 1413.5 s to 559.3 s on the four-core Q9400 workstation, with a 2.53 speedup ratio. The multicore platforms exchange the data with the internal bus, which is much faster than the network of the cluster. Therefore, the parallel efficiencies (speedup ratio/CPUs) of pRVM-MatOp on the multicore platforms are slightly better than the counterparts of the cluster. Considering the limited computing resources of the multicore platforms, pRVM-MultiClass and pRVM-Ensemble are not included in this part.

4.3. Further Discussion

The parallel implementations are summarized and compared in Table 7. pRVM-MatOp and pRVM-MultiClass have the same logic as the serial RVM. Therefore, they could not process the large-scale training set. pRVM-Ensemble solves the problem by splitting the large training set into small subsets. pRVM-MatOp is controlled by the linear algebra packages. The parallelization is implicit for the researcher, making pRVM-MatOp easy to use. pRVM-MultiClass and pRVM-Ensemble are explicitly parallelized by the programmer with the *send*, *receive* functions of the MPI library. Designing this kind of parallel algorithms is rather difficult. Affected by the unbalanced class pairs, pRVM-MultiClass could not achieve load balance. It further causes the poor scalability and low parallel efficiency. pRVM-Ensemble is load balance as long as p and M are set properly. It is scalable and can achieve a high parallel efficiency. The load balance of pRVM-MatOp is controlled by the linear algebra packages. Its scalability and efficiency are also unsatisfactory, because of the extra overhead and unparallelized nonmatrix operations.

Considering the easy-to-use characteristic, pRVM-MatOp is preferred when the multicore platform is available. It could be easily realized and achieve a satisfactory speedup. For the large-scale training set, pRVM-Ensemble on the cluster is highly recommended because of its good scalability and high parallel efficiency.

Table 7: Characteristics of the parallel implementations.

	pRVM-MatOp	pRVM-MultiClass	pRVM-Ensemble
Scale of the training set	Small	Small	Large
Explicitly parallelized	No	Yes	Yes
Easy to use	Yes	No	No
Load balance	Yes	No	Yes
Scalability	Poor	Poor	Good
Parallel efficiency	Low	Low	High

5. Conclusion

Parallel computing is used to accelerate RVM classification of the hyperspectral image in this paper. The parallelization is discussed from the matrix operations, the multiclass strategy, and the ensemble skill. Evaluated with the AVIRIS data, the parallel RVMs are proved to be effective. The training procedure is accelerated when more cores or CPUs are involved. Future improvements could be carried on by designing and evaluating the hybrid structure, which is not included in the experiments due to the lack of the testing platform. RVM could be parallelized by the multiclass or the ensemble skill in the global view and by the parallel matrix functions in the local view. The hybrid structure is suitable for the cluster of the multicore platforms, which is the trend of the supercomputer.

Acknowledgments

This work is supported by Fundamental Research Funds for Central Universities (no. 2012ZM0100), China Postdoctoral Science Foundation funded project (no. 20100480750), and Key Laboratory of Autonomous Systems and Network Control, Ministry of Education.

References

[1] G. F. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE Transactions on Information Theory*, vol. 14, pp. 55–63, 1968.

[2] J. P. Hoffbeck and D. A. Landgrebe, "Covariance matrix estimation and classification with limited training data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 763–767, 1996.

[3] C. Lee and D. A. Landgrebe, "Feature extraction based on decision boundaries," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 4, pp. 388–400, 1993.

[4] B. C. Kuo and D. A. Landgrebe, "Nonparametric weighted feature extraction for classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 5, pp. 1096–1105, 2004.

[5] B. M. Shahshahani and D. A. Landgrebe, "Effect of unlabeled samples in reducing the small sample size problem and mitigating the hughes phenomenon," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 32, no. 5, pp. 1087–1094, 1994.

[6] Q. Jackson and D. A. Landgrebe, "An adaptive classifier design for high-dimensional data analysis with a limited training data set," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 12, pp. 2664–2679, 2001.

[7] L. Bruzzone, M. Chi, and M. Marconcini, "Transductive SVM for semi- supervised classification of hyperspectral data," in *Proceedings of the International Geoscience and Remote Sensing Symposium (IGARSS '07)*, pp. 164–167, Seoul, Korea, July 2005.

[8] Y. Bazi and F. Melgani, "Toward an optimal SVM classification system for hyperspectral remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 11, pp. 3374–3385, 2006.

[9] G. Camps-Valls and L. Bruzzone, "Kernel-based methods for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 6, pp. 1351–1362, 2005.
[10] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 8, pp. 1778–1790, 2004.
[11] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine," *Journal of Machine Learning Research*, vol. 1, no. 3, pp. 211–244, 2001.
[12] B. Demir and S. Ertürk, "Hyperspectral image classification using relevance vector machines," *IEEE Geoscience and Remote Sensing Letters*, vol. 4, no. 4, pp. 586–590, 2007.
[13] G. M. Foody, "RVM-based multi-class classification of remotely sensed data," *International Journal of Remote Sensing*, vol. 29, no. 6, pp. 1817–1823, 2008.
[14] M. E. Tipping and A. C. Faul, "Fast marginal likelihood maximization for sparse Bayesian models," in *Proceeding of the 9th International Workshop on Artificial Intelligence and Statistics*, C. M. Bishop and B. J. Frey, Eds., Key West, Fla, USA, January 2006.
[15] Y. Lei, X. Q. Ding, and S. J. Wang, "Visual tracker using sequential Bayesian learning: discriminative, generative, and hybrid," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 38, no. 6, pp. 1578–1591, 2008.
[16] C. Silva and B. Ribeiro, "Towards expanding relevance vector machines to large scale datasets," *International Journal of Neural Systems*, vol. 18, no. 1, pp. 45–58, 2008.
[17] D. Yang, G. Liang, and D. D. Jenkins, "High performance relevance vector machine on GPUs," in *Symposium on Application Accelerators in High Performance Computing*, 2010.
[18] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing*, vol. 66, no. 3, pp. 345–358, 2006.
[19] A. Plaza, D. Valencia, J. Plaza, and C. I. Chang, "Parallel implementation of endmember extraction algorithms from hyperspectral data," *IEEE Geoscience and Remote Sensing Letters*, vol. 3, no. 3, pp. 334–338, 2006.
[20] S. A. Robila and L. G. MacIak, "Considerations on parallelizing nonnegative matrix factorization for hyperspectral data unmixing," *IEEE Geoscience and Remote Sensing Letters*, vol. 6, no. 1, pp. 57–61, 2009.
[21] AVIRIS NW Indiana's Indian Pines 1992 Data Set, 1992, ftp://ftp.ecn.purdue.edu/biehl/MultiSpec/.