SMC-PBC-SVM: A Parallel Pmplementation of Support Vector Machines for Data Classification

Rabie Ahmed[§], Adel Ali, Chaoyang Zhang[*]
School of computing, University of Southern Mississippi, Hattiesburg, USA
Rabie.Ahmed@eagles.usm.edu, Adel.Ali@usm.edu, Chaoyang.Zhang@usm.edu

Abstract- The **Support Vector Machine (SVM) is one of the most effective machine learning algorithms for data classification, which have a significant area of research. Since the training process of large datasets is computationally intensive, there is a need to improve its efficiency using high performance computing techniques. In this paper, we developed an efficient parallel algorithm, SMC-PBC-SVM, which combines a Parallel Binary Class with Serial Multi-class Support Vector Machines for classification. The SMC-PBC-SVM algorithm was implemented using the object-oriented C++ programming language and standard Message passing Interface (MPI) communication routines. The parallel code was executed on an ALBACORE Linux cluster, and then tested with four datasets with different sizes: Earthworm, Protein, Mnist, and Mnist8m. The results show that the SMC-PBC-SVM implementation can significantly improve the performance of data classification without the loss of accuracy. The results also demonstrated a form of proportionality between the size of the dataset and the SMC-PBC-SVM efficiency. As the dataset becomes larger, the SMC-PBC-SVM achieves a higher efficiency.**

Keywords- Classification; SVM; parallel computing.

I. INTRODUCTION

Classifying different categories among large datasets has become one of the most important computing problems. The main objective in classification is to identify patterns in a data set, which helps to analyze the data in order to make decisions. Support Vector Machines (SVMs) are a class of machine learning algorithms based on statistical learning theory, which has received wide attention for classification problems because of its accuracy and generalization property.

SVM classification involves three stages. The first one involves training the model for the classification with the training dataset. The second stage is the testing stage where the model is tested with a combination of the training data and similar unseen data. The third stage involves the actual prediction with unseen data. The training stage is the most computationally expensive process of SVMs.

The main idea behind the SVM classification algorithm is to separate two point classes of a training dataset,

$$D = \{(x_i, y_i), i = 1,2, \dots, n, \quad x_i \in R^M, \quad y_i \in \{-1,1\}\} \quad (1)$$

, with a surface that maximizes the margin between them [1]. This separating surface is obtained by solving a convex quadratic problem (QP) of the form [2]

$$\text{Min } F(\alpha) = \frac{1}{2}\alpha^T G \alpha - \sum_{i=1}^{n} \alpha_i \ , \quad (2)$$

$$\text{sub. to } \sum_{i=1}^{n} \alpha_i \, , 0 \le \alpha_i \le C, i = 1,2, \dots, n$$

, where the entries of the symmetric positive semi-definite matrix G are defined as

$$G_{ij} = y_i y_j K(x_i, x_j), i, j = 1,2, \dots, n, \quad (3)$$

where $K: R^M \times R^M \to R$ is the kernel function.

SVM has been modified to handle non-linear classification. Since the complexity of training of non-linear SVMs has been estimated to be quadratic in the number of training examples [3], it is computationally expensive when large datasets with tens of thousands of training examples are used. To reduce the training time, the optimization problem can be broken into smaller QP problems [4]. Originally, SVM was introduced for binary classification, and then it was extended for multi-class classification. It was improved by caching the kernel calculations [5]. Because of the wide use of the Internet, a large amount of data is being collected. Hence, the importance of using an efficient SVM that utilizes parallel computing facilities and multi-core processing elements for (multi-class) classification of large datasets grows even larger. Therefore, a lot of research efforts were directed to find the optimal parallel algorithm for the different kinds of datasets. For large binary classification problems, there is a need to break it down into smaller pieces, so that the smaller partitions can be computed concurrently. Research has been conducted in this area, and some progress has been made in [6], [7], and [8]. On the other hand, for large multi-class classification problems, progress has been made in [3] and [9]. Also, a lot of work has been done in [5] and [10] to develop kernel computation costs. Some other efforts have been achieved in [11], [12], [13] and [14] to optimize working set size selection. Additionally, other tries have been done in [15], and [16] to develop SVM training by quickly removing most of non-support vectors.

The LIBSVM [17] software is developed for a working set of size two, which tends to minimize the computational cost per iteration. In this case, the inner QP subproblem can be systematically solved without requiring a numerical QP solver and the updating of the objective gradient only involves the two Hessian columns corresponding to the two updated

*Correspondence: USM, School of Computing, Chaoyang.Zhang@usm.edu.
§Permanent address: Beni Suef University, Faculty of Science, Beni Suef, Egypt.

variables. On the other hand, if few variables are updated per iteration, slow convergence is normally implied. The SVM^{light} [18] algorithm uses a more general decomposition strategy, also by common sense it can exploit working sets of sizes larger than two. By updating more variables per iteration, such an approach is more suitable for a faster convergence, but it introduces additional difficulties and costs. A generalized maximal-violating pair strategy for the working set selection and a numerical solver for the inner QP subproblems are required. Moreover, as more variables are updated per iteration, the objective gradient updating is more expensive. While SVM^{light} can run with any working set size, numerical experiences prove that it effectively faces the above difficulties only in the case of small sized working sets, $N_{sp} = O(10)$, where it often exhibits comparable performance with LIBSVM.

Following the SVM^{light} decomposition techniques, another effort to strike a balance between the convergence rate and cost per iteration was introduced in [6]. Unlike SVM^{light}, it is medium or large sized working sets, ($N_{sp} = O(10^2)$ or $N_{sp} = O(10^3)$, that allow the method to converge in a small number of iterations where the most costly tasks are. For example, subproblem solving and gradient updating can be simply and effectively distributed between the available processors. Based on this idea, a new parallel gradient projection-based decomposition technique (PGPDT) is developed and implemented in software to train support vector machines for binary classification problems in parallel as in [8].

Even though all previous results were encouraging, more research was needed to improve the performance of the process. In this paper, we introduce a new classification algorithm which merges parallel binary classification with serial multi-class classification to produce an efficient parallel algorithm for classification. We named the new algorithm SMC-PBC-SVM.

II. SMC-PBC-SVM ALGORITHM

The SMC-PBC-SVM algorithm combines Parallel Binary Class with Serial Multi Class Support Vector Machines for classification. It includes seven steps and works as follows: 1) Reads a dataset from an input file, 2) Groups samples of the same class together, 3) Collects each two classes into one task, 4) Sorts $\frac{K(K-1)}{2}$ tasks based on its size, 5) Divides processes group into two subgroups, mulgroup and bingroup, such that mulgroup is used to build $\frac{K(K-1)}{2}$ binary tasks where K is the number of classes in the dataset, and bingroup is used to solve each binary task from $\frac{K(K-1)}{2}$ tasks in parallel, 6) Builds SVM model after solving all binary tasks, and 7) Writes the SVM model into an output model file which is used to predict testing dataset file. The algorithm flowchart is illustrated in Figure 1.

Since the SMC-PBC-SVM is based on the Parallel Binary Class algorithm implemented by PGPDT, we briefly explain here the fundamental principles and decomposition technique used in [8]. We start that by stating some fundamental principles. At each decomposition iteration, the indices of the variables $\alpha_i, i = 1,2,..,n$, are split into the set B of basic variables, usually called the working set, and the set N = {1,2,…,n} \ B of nonbasic variables. In consequence, the kernel matrix G, the vectors $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)^T$, and the vector $y = (y_1, y_2, \dots, y_n)^T$ can be arranged with respect to B and N as follows:

$$G = \begin{bmatrix} G_{BB} & G_{BN} \\ G_{NB} & G_{NN} \end{bmatrix}, \qquad \alpha = \begin{bmatrix} \alpha_B \\ \alpha_N \end{bmatrix}, \qquad y = \begin{bmatrix} y_B \\ y_N \end{bmatrix}$$

Then, suppose that N_{sp} is the size of the working set, where $N_{sp} = \#B$ and α^* is a solution of QP (2), and N_p is the number of processes which are used for solving that QP, and each of them has a local copy of the training set D (1), where the entries of G are defined by G(3). The decomposition technique used by the PGPDT falls within the general idea stated in PGPDT algorithm, which is shown in Figure 2. Label "Distributed task" in A2 and A3 of PGPDT algorithm refers to the steps where the N_p processors cooperate together to perform the required computation. In these steps, communications and synchronization are needed. In the other steps, the processors asynchronously perform the same computations on the same input data to obtain a local copy of the expected output data.

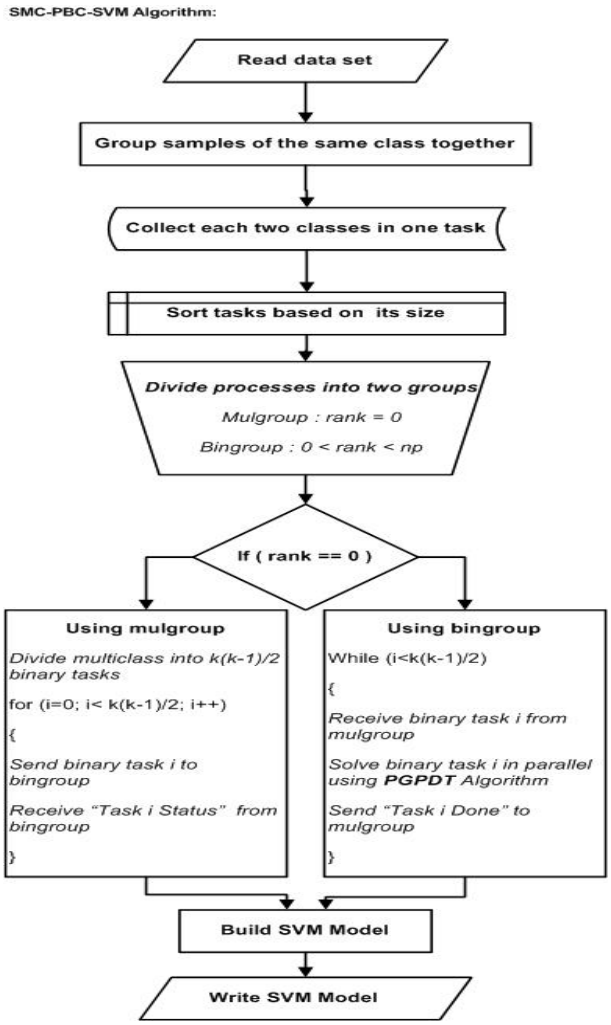

Figure 1. Serial Multi Class Parallel Binary Class Support Vector Machines

A1. *Initialization.* Set $\alpha^{(1)} = 0$ and let n_{sp} and n_c be two integer values such that $n \geq n_{sp} \geq n_c$ n_c even. Choose n_{sp} indices for the working set $\mathcal{B}$ and set $k = 1$.

A2. *QP subproblem solution.* **[Distributed task]** Compute the solution $\alpha_{\mathcal{B}}^{(k+1)}$ of

$$\begin{aligned} \min \quad & \frac{1}{2}\alpha_{\mathcal{B}}^T G_{\mathcal{B}\mathcal{B}}\alpha_{\mathcal{B}} + \left(G_{\mathcal{B}\mathcal{N}}\alpha_{\mathcal{N}}^{(k)} - 1_{\mathcal{B}}\right)^T \alpha_{\mathcal{B}} \\ \text{sub. to} \quad & \textstyle\sum_{i\in\mathcal{B}} y_i\alpha_i = -\sum_{i\in\mathcal{N}} y_i\alpha_i^{(k)}, \\ & 0 \leq \alpha_i \leq C, \quad \forall i \in \mathcal{B}, \end{aligned}$$

where $1_{\mathcal{B}}$ is the n_{sp}-vector of all one; set $\alpha^{(k+1)} = \left(\alpha_{\mathcal{B}}^{(k+1)^T}, \ \alpha_{\mathcal{N}}^{(k)^T} \right)^T$.

A3. *Gradient updating.* **[Distributed task]** Update the gradient

$$\nabla f\left(\alpha^{(k+1)}\right) = \nabla f\left(\alpha^{(k)}\right) + \begin{bmatrix} G_{\mathcal{B}\mathcal{B}} \\ G_{\mathcal{N}\mathcal{B}} \end{bmatrix} \left(\alpha_{\mathcal{B}}^{(k+1)} - \alpha_{\mathcal{B}}^{(k)}\right)$$

and terminate if $\alpha^{(k+1)}$ satisfies the KKT conditions.

A4. *Working set updating.* Update $\mathcal{B}$ by the following selection rule:

A4.1. Find the indices corresponding to the nonzero components of the solution of

$$\begin{aligned} \min \quad & \nabla f\left(\alpha^{(k+1)}\right)^T d \\ \text{sub. to} \quad & y^T d = 0, \\ & d_i \geq 0 \quad \text{for } i \text{ such that } \alpha_i^{(k+1)} = 0, \\ & d_i \leq 0 \quad \text{for } i \text{ such that } \alpha_i^{(k+1)} = C, \\ & -1 \leq d_i \leq 1, \\ & \#\{d_i \mid d_i \neq 0\} \leq n_c. \end{aligned}$$

Let $\bar{\mathcal{B}}$ be the set of these indices.

A4.2. Fill $\bar{\mathcal{B}}$ up to n_{sp} entries with indices $j \in \mathcal{B}$. Set $\mathcal{B} = \bar{\mathcal{B}}, k \leftarrow k+1$ and go to A2.

Figure 2. Parallel Gradient Projection Decomposition Technique Algorithm

III. SMC-PBC-SVM Results Analysis

The SMC-PBC-SVM algorithm was implemented using the object-oriented C++ programming language and the standard MPI communication routines [19]. The experiments are carried out on two different parallel platforms at the Mississippi Center for Supercomputing Research (MCSR) [20], and ALBACORE Linux clusters [21]. The performance analysis was visualized using Jumpshot software [22]. The best performance was achieved with ALBACORE, which contains 12 compute nodes, each node has 2 chips, and each chip has 4 cores. Each core is an Intel(R) Xeon(R) CPU X5570 with 2.93 GHZ and 8192 KB Cache. Each node of node0 and node1 has 16 GB, and node2 to node11 has 12 GB.

The SMC_PBC_SVM has been tested using different size datasets, Earthworm, Protein, Mnist, and Mnist8m, which are small, medium, large, and very large datasets respectively. Table I includes the description of these datasets. The results show that SMC-PBC-SVM is very efficient with very large datasets as Mnist8m dataset, highly efficient with large datasets as Mnist dataset, reasonably efficient in medium datasets as Protein dataset, and less efficient with small datasets as Earthworm dataset. For more details, see Table II, which shows the training run time when number of processes 1, 2, 4, 9, 16, 25, and 36 are used.

TABLE I. Datasets Description

Dataset Name	Earthworm	Protein	Mnist	Mnist8m
Reference	[23] LY10	[24] JW02	[25] YL98	[26] GL07
Classes Number	3	3	10	10
Features Number	869	357	780	784
Training Samples Number	248	17766	60000	8100000
Testing Samples Number	30	662	10000	10000
Best C	32.0	32.0	32.0	32.0
Best γ	0.001953	0.001953	0.001953	0.001953
Accuracy %	100	69.55	98.21	98.73

TABLE II. Training Run Time in Seconds

Processes Number	Earthworm	Protein	Mnist	Mnist8m
NP = 1	460.000	2427.840	4118.800	5111.370
NP = 2	454.031	2299.361	3915.277	4641.645
NP = 4	421.053	1618.583	2187.881	2961.146
NP = 9	398.174	933.000	1409.055	1824.251
NP = 16	367.177	732.000	986.758	1252.153
NP = 25	347.444	516.617	814.520	945.837
NP = 36	334.620	431.059	701.696	768.011

The complexity for the multi-class classification is $O(K\,M\,N^2)$, where K is the number of classes, M is the number of features, and N is the number of training samples [3]. This serial complexity is the worst case scenario for multiclass classification using binary classifiers. But when this job is distributed among P processors, the parallel complexity becomes $O\left(\frac{K\,M\,N^2}{P} + T_c\right)$, where T_c is the complexity due to communication for task scheduling and combining the results.

We evaluate the parallel performance by the relative speedup (S), which is defined as the ratio of the time taken to solve a problem on a single processing element to the time required to solve the same problem on a parallel computer with p identical processing elements, then

$$S = \frac{T_s}{T_p} = \frac{\text{Training Time spent on a single processor}}{\text{Training Time spent on a Np processors}} \quad (4)$$

Then,

$$S = O\left(\frac{K M N^2}{\frac{K M N^2}{P} + T_c}\right) \quad (5)$$

$S \cong O(P)$, if $T_c \to 0$,
when transmitted data between processors is insignificant.

Efficiency (E) is another way to analyze the parallel implementation, which is defined as the ratio of speedup to the number of processing elements, then

$$E = \frac{S}{P} = \frac{\text{Speedup}}{\text{Processes number}} \quad (6)$$

Then,

$$E = O\left(\frac{\frac{K M N^2}{\frac{K M N^2}{P} + T_c}}{P}\right) \quad (7)$$

$S \cong O(1)$, if $T_c \to 0$,
when transmitted data between processors is insignificant.

Figures 3, 4, and 5 show the relationship between the number of processors and run time, speedup, and efficiency respectively. Also, Figures 6 and 7 show the output of Jumpshot, which is a visualization tool to study the performance of parallel programs using log files that are generated from the execution of the SMC-PBC-SVM implementation to Earthworm and Mnist datasets using 16 processors.

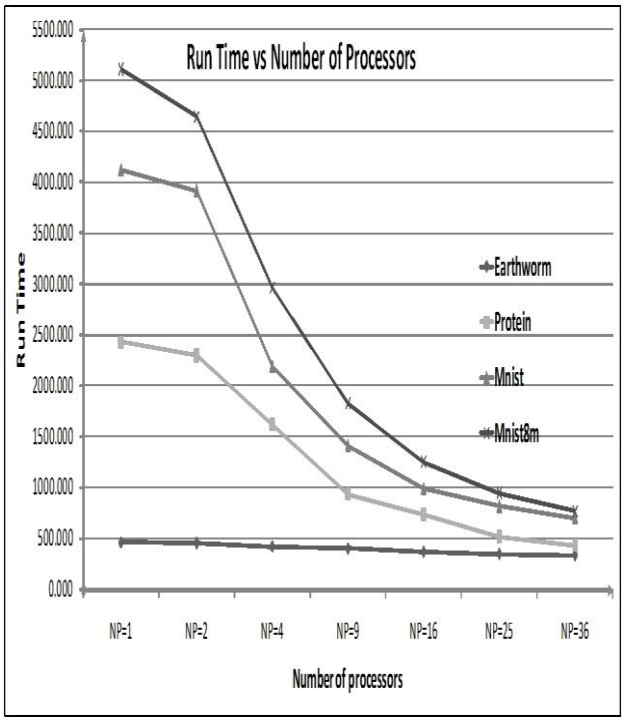

Figure 3. The relationship between Run Time and Number of Processors

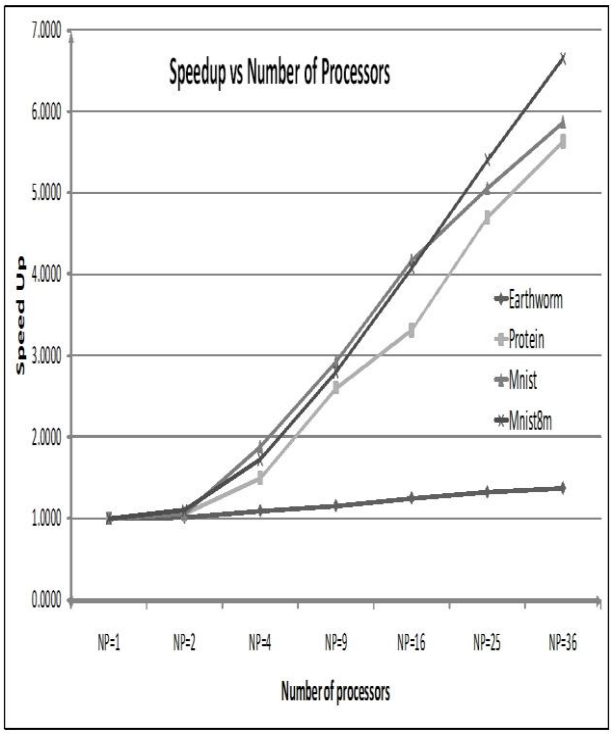

Figure 4. The relationship between Speedup and Number of Processors

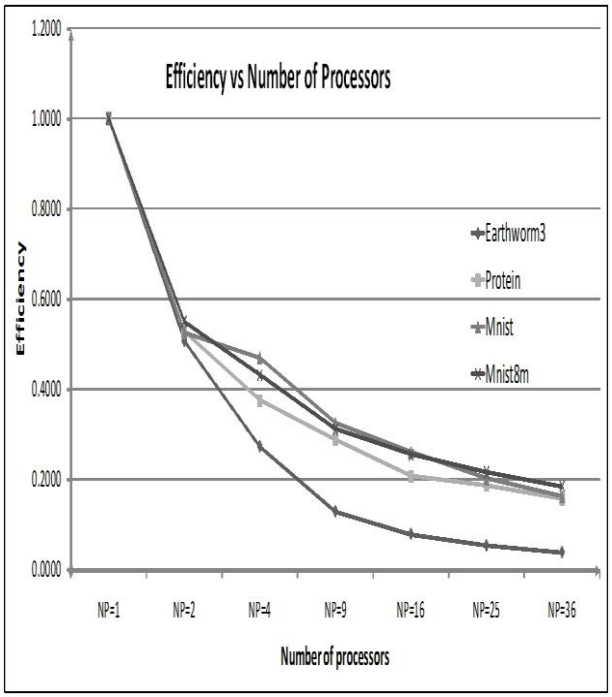

Figure 5. The relationship between Efficiency and Number of Processors

In Figure 3, we can see that the training execution time goes down as the number of processors is increased. It shows that when there is a sufficient work to be done concurrently on an increasing set of processors, there is a related improvement in performance. For small datasets, as Earthworm, there is not

enough work to be done concurrently. Therefore, there is no significant improvement in performance in that case. On the other hand, for very large datasets, as Minst8m, there is a sufficient work to be done concurrently. Therefore, there is a significant improvement in performance with very large datasets. By the same concept, the performances of medium and large datasets, such as Protein and Mnist, fell between the performances of small datasets and very large datasets. We used a Portable batch system (PBS) script which allows us to choose number of processors which are needed for job execution.

Referring to the speedup (5), we can see that, the size of datasets and the size of classes are significant factors in speedup. Therefore, it may not be possible to avoid the idling of some processes. Also, as the number of processes is increased, the speedup is less than linear, indicating idle time for few processors while waiting for other processors. In Figure 4, we can observe that the speedup is closed to linear speedup from very large dataset to small dataset.

Efficiency (7) is defined as the ratio of speedup to the number of processors. Therefore, a higher speedup ensures good efficiency, implying efficient use of the parallel resources. In Figure 5, we can see, the efficiency of SMC-PBC-SVM is gradual from a very large dataset to a small dataset.

In Figure 6, we can observe that there is a lot of lost time through idling especially with a large number of processes where the dataset size is small. Therefore, this explains the low efficiency with small datasets. While in Figure 7, we can see that there is no lost time because most time is spent in computation without idle time, where the dataset size is large. Therefore, this explains the high efficiency with large datasets.

IV. CONCLUSION AND FUTURE WORK

In this paper, the problem of solving multi-class classification using an efficient parallel support vector machine implementation was investigated. SMC-PBC-SVM is an efficient parallel algorithm, which combines Parallel Binary Class with Serial Multi-Class Support Vector Machines for classification. The SMC-PBC-SVM algorithm was implemented using the object-oriented C++ programming language and standard Message Passing Interface (MPI) communication routines. The parallel code was executed on an ALBACORE Linux cluster, and then tested with four datasets with difference sizes: Earthworm, Protein, Mnist, and Mnist8m. The results show that the SMC-PBC-SVM implementation can significantly improve the performance of data classification without loss of accuracy. As the dataset becomes larger, the SMC-PBC-SVM achieves a higher efficiency. In this paper, we used one processor in mulgroup which means SMC, and more than one processes in bingroup which means PBC.

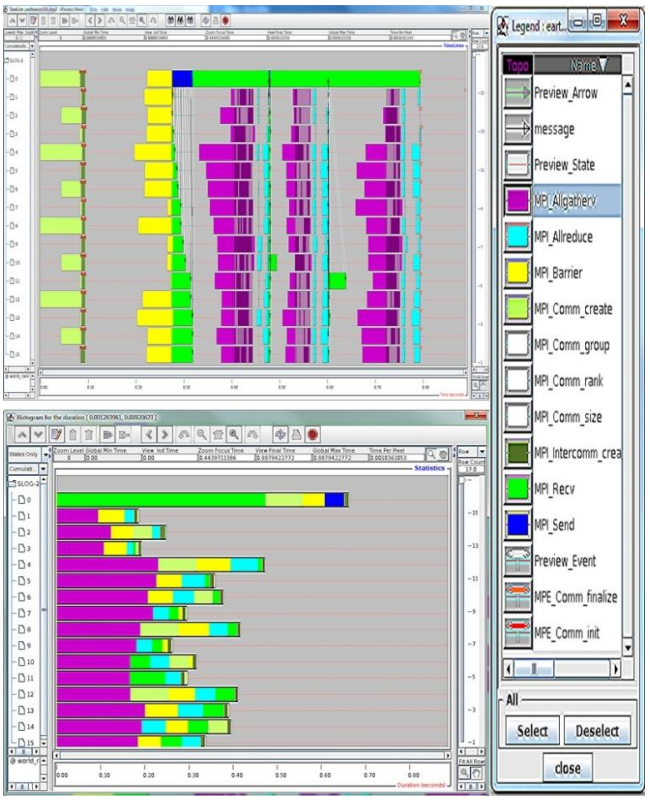

Figure 6. Jumpshot Timeline, Histogram and legend windows of Earthworm dataset using 16 processors

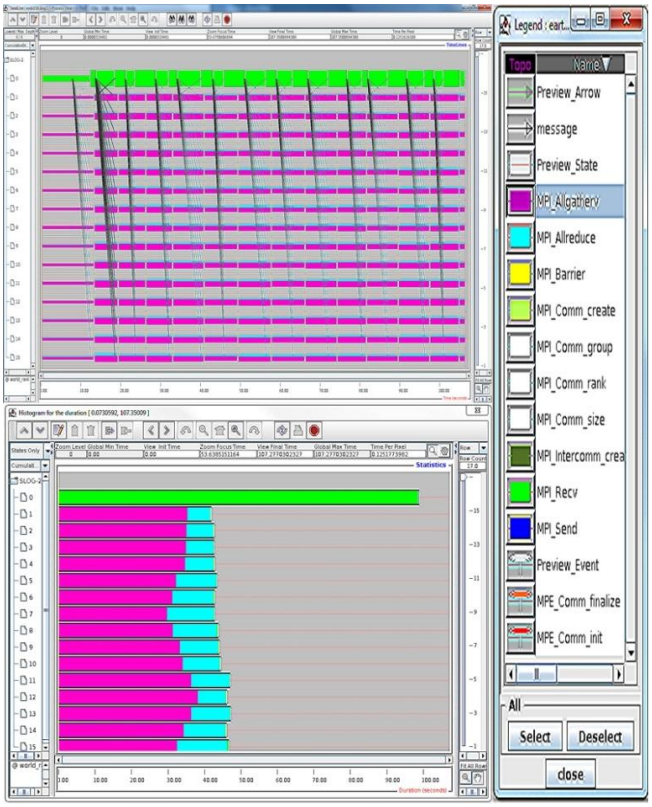

Figure 7. Jumpshot Timeline, Histogram and legend windows of Mnist dataset using 16 processors

In the future work, the scope of mulgroup will extend to include more than one processor, (Add comma) where each of which has its own bingroup. Therefore, these processes in mulgroup work in parallel, and then we will produce PMC-PBC-SVM implementation which will improve the performance and will address the limitations of SMC-PBC-SVM.

ACKNOWLEDGMENT

The authors gratefully acknowledge the Beni Suef University for their financial support.

REFERENCES

[1] C. Cortes, and V. Vapnik, "Support-vector networks Machine Learning," 1995.

[2] V. Vapnik, "The Nature of Statistical Learning Theory," Springer-Verlag, 1999.

[3] A. Rajendran, "Parallel Support Vector Machines for Mulicategory Classification of Large Scale Data," Dissertation, University of Southern Mississippi, 2007.

[4] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," IEEE Workshop Neural Networks for Signal Processing, pp. 276–285, 1997.

[5] S. Qiu, and T. Lane, "Parallel computation of RBF kernels for support vector classifiers," Fifth SIAM International Conference on Data Mining, 2005.

[6] G. Zanghirati, and L. Zanni, "Parallel solver for large quadratic programs in training support vector machines," Parallel Computing 29, pp. 535-551, 2003.

[7] L. Zanni, T. Serafini, and G. Zanghirati, "Parallel software for training large scale support vector machines on multiprocessors systems," Journal of Machine Learning Research 7, pp. 1467-1492, 2006.

[8] T. Serafini, G. Zanghirati, and L. Zanni, "PGPDT: Parallel Gradient projection based on Decompostion Technique," Technical report, http://dm.unife.it/gpdt/,2007.

[9] C. Zhang, P. Li, A. Rajendran, and Y. Deng, "Parallelization of multicategory support vector machines for classifying microarray data," BMC Bioinformatics, 2006.

[10] T.Eitrich, and B. Lang, "Efficient Implementation of serial and parallel support vector machine training with a multi-parameter kernel for large-scale data mining," Proceedings of World Academy of Science, Engineering, and Technology 11, 2006.

[11] T. Serafini, L. Zanni, "On the Working Set Selection in Gradient Projection-based Decomposition Techniques for Support Vector Machines," Optim. Meth. Soft. 20, pp. 583-596, 2005.

[12] R. Fan, P. Chen, and C. Lin, "Working set selection using second order information for training support vector machines," Journal of Machine Learning Research, 2005.

[13] T.Eitrich, and B. Lange, "On the optimal working set size in serial and parallel support vector machine learning with the decomposition algorithm," Proceedings of the fifth Australasian Conference on Data mining and analysis 61, pp. 121-128,2006.

[14] J. Platt, "Fasting training of support vector machines using Sequential Minimal Optimization," In Advances in Kernel Methods Support Vector Learning, MIT press, pp. 185-208, 1999.

[15] J. Dong, A. Krzyzak, and C. Suen, "A fast parallel optimization for training support vector machine," Proceedings of 3rd Int. Conf. Machine Learning and Data Mining, pp. 96-105, Germany, 2003.

[16] J. Dong, A. Krzyzak, and C. Suen, "Fast SVM training algorithm with decomposition on very large data sets," IEEE Transactions on Pattern Analysis and Machine Intelligence 27, pp. 603-618, 2005.

[17] C. Change and C. Len, "LIBSVM: a library for support vector machines," Technical report, http://www.csie.ntu.edu.tw/_cjlin/libsvm/, 2005.

[18] T. Joachims, "SVM ^{Light}, Support vector machine," Developed at the university of Dortmund , and it is available at http://svmlight.joachims.org/, 1994.

[19] MPI: A Message Passing Interface Standard (Version 2.2), Message Passing Interface Forum, September 4, URL http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf, 2009.

[20] MCSR: Mississippi center for supercomputing research, Technical report, http://www.mcsr.olemiss.edu.

[21] USM: Albacore Cluster, Chemistry lab at The University of Southern Miss. http://albacore.st.usm.edu/cgi-bin/portal.cgi.

[22] A. Chan, D. Ashton, R. Lusk, and W. Gropp, "Jumpshot-4 Users Guide," Mathematics and Computer Science Division, http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm, 2007.

[23] Y. Li, N. Wang, E. Perkins, C. Zhang, and P. Gong, "Identification and Optimization of Classifier Genes from Multi-Class Earthworm Microarray Dataset," PLoS ONE 5(10): e13715. doi:10.1371/journal.pone.0013715, October 2010.

[24] J. Wang, "Application of support vector machines in bioinformatics," Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University, 2002.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, 86(11):2278-2324, MNIST database available at http://yann.lecun.com/exdb/mnist/, 1998.

[26] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," In Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston, editors, Large Scale Kernel Machines, pages 301-320. MIT Press, Cambridge, 2007.