Parallel hybrid Implementation of Cascade Support Vector Machine (SVM) with parallelized Sequential Minimal Optimization (SMO) for classification.

Md. Imbesat Hassan Rizvi, Prateek Kushwaha
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
imbesatrizvi@ssl.serc.iisc.in, prateek@ssl.serc.iisc.in

***Abstract*—Support Vector Machines (SVMs) are powerful but computationally expensive machine learning (ML) algorithm for supervised classification task whihc is frequently witnessed in the ML domain. For optimization of objective function SMO is widely used while for large dataset Cascading approach is well suited. Both of these are parallelizable in orthogonal sense i.e. independent of each other. Motivated by these, in this project, we have implemented a hybrid version of both the above mentioned techniques with SMO being implemented using CUDA while cascading being implemented using MPI.**

I. Introduction

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.This is currently a most powerful tool due to its effictveness in higher dimension spaces but usually accompanied by high compute requirements which arises due to the need of solving a Quadratic Optimization problem (or Quadratic Programming, QP) for determining Lagrange Multipliers and Support Vectors. General-purpose QP solvers tend to scale with the cube of the number of training vectors ($O(k^3)$).

The standard two-class soft-margin SVM classification problem (C-SVM), which classifies a given data point $x \in \mathbb{R}^n$ by assigning a label $y \in \{1, 1\}$.

A. SVM Training

Given a labeled training set consisting of a set of data points $x_i, i \in \{1, ..., l\}$ with their accompanying labels $y_i, i \in \{1, ..., l\}$, the SVM training problem can be written as the following Quadratic Program:

$$max \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\alpha^T Q \alpha \tag{1}$$
$$s.t. 0 \leq \alpha_i \leq C, \forall i \in 1...l$$
$$y^T \alpha = 0$$

where α_i are the lagrange multipliers (weights, one for each training points), which are being optimize to determine SVM classifier. C is a parameter which trades off wide margin with a small number of margin failures and $Q_{ij} = y_i y_j \phi(x_i, x_j)$, where $\phi(x_i, x_j)$ is a kernel function. We are using the standard Gaussian kernel function $\phi(x_i, x_j) = exp\{-\gamma \left\| x_i - x_j \right\|^2\}$.

B. SVM Classification

The SVM classification problem is as follows: for each data point z which should be classified, compute

$$z' = sgn \left\{ b + \sum_{i=1}^{l} y_i \alpha_i \phi(x_i, z) \right\} \tag{2}$$

where $z \in \mathbb{R}^n$ is a point which needs to be classified, b is the bias derived from the solution to the SVM training problem (1), and all other variables remain as previously defined.

II. Related Work

Solving the quadratic optimization problem with several linear constraints is a combinatorial search method over the linear constraints. The problem becomes more gruesome when the dataset for classification becomes too large. Hence, several approaches have been proposed and tried to tackle with both these issues.Standard approaches for solving QP like Gradient Projection method, Interior Point Method etc are already there.

However, much research has been done to accelerate the training time. The training time for SVM can significantly be reduced if the optimzation problem, which is the core of the SVM, can be parallelized. Some of these techniques are Osunas decomposition approach (Osuna et al., 1997) [1], Sequential Minimal Optimization (SMO) algorithm (Platt, 1999) [2]. The popularity of SMO is due to it's better scaling properties than standard chunking algorithms that uses Projected Conjugate Gradient or other optimization techniques. Works have also been done to parallelize the standard QP optimization techniques such as Parallel Gradient Projection (PGP) Technique by Zanni et al (2006) [3] and Parallel Interior Point Method (PIPM) by Wu et al (2006) [4]. Collobert et al (2002) [5] proposes a method where the several smaller SVMs are trained in a parallel fashion and their outputs weighted using a Artificial Neural Network. Since then, several combination based SVMs have been proposed.

On the front of dataset decomposition also, which is particularly useful for large datasets, several attempts have been made to parallelize the SVM. The Cascade SVM introduced by Graf et el (2005) [6] uses a method of Divide and Conquer to solve SVM over large training datasets. A similar approach with only one level of computation and label determination by

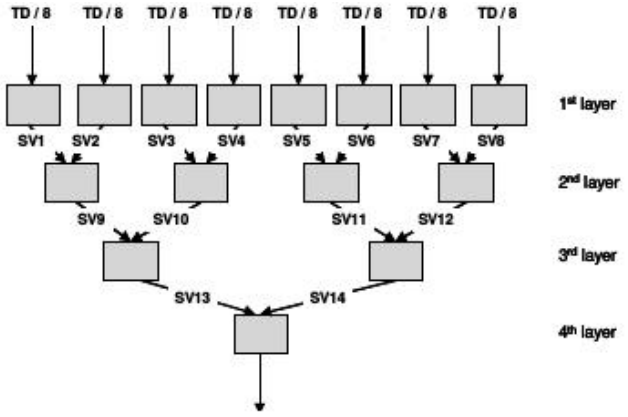

Fig. 1: Binary cascade SVM. TD: Training data, SVi: Support vectors produced by optimization i.

maximum agreement among processors, usually referred to as baggage approach, is also there.

III. Methodology

A. The Cascade SVM

Cascade support vector machines have been introduced as extension of classic support vector machines that allow a fast training on large data sets.It is a filtering process which eleminates the non-support vectors early from the optimization.he method described by Graf et al. [6]

- Divide data into *n* disjoint subsets of preferably of equal size.
- Independently train an SVM on each of the data subsets.
- Combine the SVs two-by-two to create new subsets for next layer.
- Continues until only one set of vectors is left.

In the original paper, Graf et el. also discussed the multiple run strategy for cascade SVM to obtain global optimum. Often a single pass through cascade produces satisfatory accuracy. For speed reasons we only perform one pass of through the cascade.

B. SMO Algorithm

Sequential Minimal Optemization is an iterative algorithm for solving a quadratic programming(QP) optimization problem that arises during the training of SVM.It takes advantage of the sparse nature of the support vector problem and the simple nature of the constraints in the SVM QP to reduce each optimization step to its minimum form: updating two i weights. The bulk of the computation is then to update the Karush-Kuhn-Tucker(KKT) optimality conditions for the remaining set of weights and then reduce to find the two maximally violating weights, which are then updated in the next iteration until convergence.

The optimality conditions can be tracked through the vector $f_i = \sum_{j=1}^{l} \alpha_i y_i \Phi(x_i, x_j) - y_i$, which is constructed iteratively as the algorithm progresses. Following (Keerthi et al., 2001) [7], we partition the training points into 5 sets, represented by their indices:

Algorithm 1 Sequential Mininal Optimization

Input: training data x_i,labels y_i, $\forall i \in \{1....l\}$
Initialize: $\alpha_i = 0$, $f_i = -y_i$, $\forall i \in \{1....l\}$
Compute: b_{high}, I_{high}, b_{low}, I_{low}
Update α_{high} and α_{low}
repeat:
  Update f_i, $\forall i \in \{1....l\}$
  Compute b_{high}, I_{high}, b_{low}, I_{low}
  Update α_{high} and α_{low}
untill $b_{low} \leq b_{high} + 2\tau$

1. (Unbound SVs) $I_0 = \{i : 0 < \alpha_i < C\}$
2. (Positive NonSVs) $I_1 = \{i : y_i > 0, \alpha_i = 0\}$
3. (Bound Negative SVs) $I_2 = \{i : y_i < 0, \alpha_i = C\}$
4. (Bound Positive SVs) $I_3 = \{i : y_i > 0, \alpha_i = C\}$
5. (Negative NonSVs) $I_4 = \{i : y_i < 0, \alpha_i = 0\}$

where C remains as defined in the SVM QP. We then define $b_{high} = min\{f_i : i \in I_0 \cup I_1 \cup I_2\}$, and $b_{low} = max\{f_i : i \in I_0 \cup I_3 \cup I_4\}$, with their accompanying indices $I_{high} = argmin_{i \in I_0 \cup I_1 \cup I_2}\ f_i$, and $I_{low} = argmax_{i \in I_0 \cup I_3 \cup I_4}\ f_i$.

As shown in algorithm 1, at each iteration we search for the values of b_{low} and b_{high}. Then update their α weights according to following:

$$\alpha_{low}^{'} = \alpha_{low} + \frac{y_{low}(b_{high} - b_{low})}{\eta} \tag{3}$$

$$\alpha_{low}^{'} = \alpha_{low} + y_{low} y_{high} (\alpha_{low} - \alpha_{low}^{'}) \tag{4}$$

where $\eta = \phi(x_{I_{high}}, x_{I_{high}}) + \phi(x_{I_{low}}, x_{I_{low}}) - 2\phi(x_{I_{high}}, x_{I_{low}})$. To ensure that this update is feasible,$\alpha_{I_{low}}$ and $\alpha_{I_{low}}$ must be clipped to the valid range $0 \leq \alpha_i \leq C$. After the α update, the optimality condition vector f is updated for all points as follows:

$$f_i^{'} = f_i + (\alpha_{I_{high}}^{'} - \alpha_{I_{high}}) y_{I_{high}} \phi(x_{I_{high}}, x_i) + (\alpha_{I_{low}} - \alpha_{I_{low}}) y_{I_{low}} \phi(x_{I_{low}}, x_i) \tag{5}$$

The majority of work is performed during above updation in this algorithm. The bias is given by $b = \frac{(b_{high} + b_{low})}{2}$.

C. Cascade Implementation using MPI

Initially the data($l \times m$, l instances and m fetures) are divided into p processors i.e. each processor will get approx $\frac{l}{p}$ number of instances to learn for first level (i.e. level 0) of cascade.

At a given level, the processors with $rank\%2^{level} = 0$ are only active and idependently performs SVM computation for their subset of data, thereby, learning α and b. The SVs represented by the indices of non-zero components of α, calculated by the odd numbered processes of current level, are then communicated to even numbered processes for generating the new subset for the next level.

For the sake of simplicity, we have considered number of processors only to be powers of 2. At the final level, the processor with $rank = 0$ learns the α vector as well as the bias b, which is then broadcasted to all the other processors for the testing purpose.
For the testing purpose, the test data set is also divided into p processors and each of them calculate class predictions for the test cases. The total number of correctly predicted cases are then communicated to processor with $rank = 0$ using MPI_Reduce.

D. SMO Implementation using CUDA

There are two CUDA kernel implementations. First is for formation of kernel matrix, which is one time computation and it resides in global memory so that each block can access the matrix required for further computation in each iteration of SVM training. The second one is for calculating update $f^{'}$, which reflects the impact of the optimization step on the optimality condition of the remaining data points.

IV. Experiments and Results

A. Experiment Setup

We used Gaussian Kernel in our experiment, since it is widely employed. The dataset used is Adult dataset [**?**], [**?**] presents the task of classifying a persons income region being greater or less than $50,000/year based on US census data.There are 32561 instances and 14 features with real and categorical data. The features were processed and transformed into a total size of 123 feature components. These components were obtained by converting the real valued data into quartiles and categorical data into indivial features per category. Thus the final feature set had at most 14 features with a value 1 indicating that the data were present only for these while the rest had a value of 0. The data is divided as 80% for training purpose and 20% for testing purpose and we used the value of $C = 100$ and $\gamma = 0.5$.

The variation in the number of processors used are 2, 4 & 8, which leads varying amount of data decomposition. Work for each processor reduces as the number of processor is increased.

B. Results

The hybrid implementation of Cascade SVM using MPI with SMO on the GPU is compared with Sequential SVM with SMO implementation as well as only GPU and only MPI implementation with processor numbers 2, 4 and 8. The implementations were compared in terms of both accuracy and Speedup for both training as well as testing cases.

The results obtained are reported in the following table:

TABLE I: Performance report of various implementations.

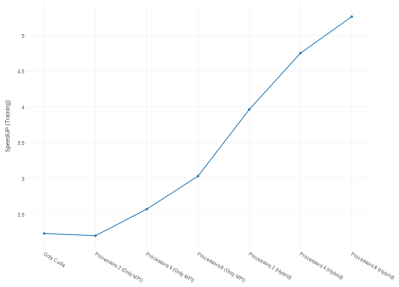

Fig. 2: SpeedUp comparison among different implementations for Training

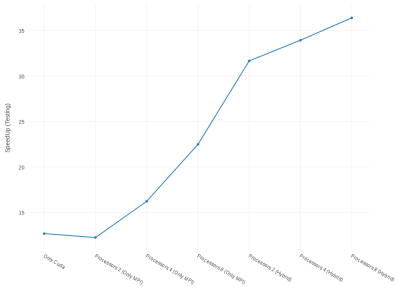

Fig. 3: SpeedUp comparison among different implementations for Testing

Approach	Training time (sec)	Training Speedup	Testing time (sec)	Testing Speedup	Accuracy (%)
Sequential	143.274	1	22.156	1	81.15
Only Cuda	64.357	2.226	1.751	12.653	81.15
Only MPI					
Processors 2	65.274	2.195	1.813	12.221	73.46
Processors 4	55.837	2.566	1.367	16.208	71.53
Processors 8	47.321	3.028	0.986	22.471	65.29
Hybrid (Cuda + MPI)					
Processors 2	36.126	3.96	0.701	31.65	73.76
Processors 4	30.324	4.75	0.653	33.92	71.53
Processors 8	27.198	5.26	0.609	36.38	65.29

The reports as obtained in TABLE I suggests that although with increasing number of processors, the execution time is decreasing, there is also an increasing amount of error introduced while classifying the test data. This is usually the trade off and the decision to select the number of processors is dependent on the size of the dataset. The larger the set, the

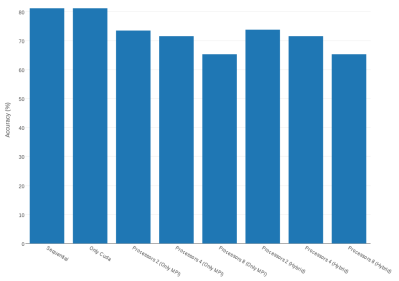

Fig. 4: Accuracy comparison among different implementations

more number of processors one can utilize for the cascading effect with little accuracy loss over sequential implementation.

Another point to note is that although the speedup for training phase is not as impressive as that shown in the case of testing, but since the training is to be performed once and testing is often performed repeatedly for the datasets being obtained in future, this significant improvement is encouraging.

As the Speedup results are highly dependent on the dataset representation, dataset decomposition and the machine specifications. Moreover, we have implemented a hybrid version of SVM and hence direct comparisons with existing papers which have considered the parallelization job in isolation either for cascade SVM or for parallelized SMO only. Nevertheless, we are reporting the execution times at least for the income dataset which we found to be used and reported by Catanzaro et.al [9] for parallelized SMO. Comparison with LibSVM tools was also performed in the same paper. The training time, testing time, training Speedup, testing speedup and accuracy percentage over the LibSVM obtained for the Income classification dataset were 36.312 sec, 0.570 sec, 15.1, 132.5 and 82.73 %. The reason for the contrasting speedup compared to our implementation is mostly due to the execution time of sequential implementation of LibSVM on their machine being 550.178 sec for training and 75.65 sec for testing. As for cascade SVM, we were not able to find any MPI implimentation since most of the cascade parallelization available were implemented on Map Reduce framework.

V. Conclusions

A clear improvement in training and testing time for Hybrid implementation is observed over Sequential, Only Cuda and only MPI implementations. However, it is also advisable to decide the number of decompositions, hence, number of processors to be used be obtained as a trade off between execution time and accuracy. This will greatly depend on the dataset under consideration. The suitable number of processors in our case came out to be either 2 or 4.

It is also to be noted that, we have performed the analysis by considering the training and testing times only. However, cross-validation is also an important aspect in all the machiner learning techniques, which are used for deciding the best or near to best suited values of hyperparameters like the Regularizer constant C and γ. The present work can be extended to encompass the cross-validation phase too.

References

[1] Osuna, E., Freund, R., Girosi, F. (1997). An improved training algorithm for support vector machines. *Neural Networks for Signal Processing* [1997] VII. Proceedings of the 1997 IEEE Workshop, 276-285.

[2] Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. *In Advances in kernel methods: support vector learning*, 185208. Cambridge, MA, USA: MIT Press.

[3] Zanni, L., Serafini, T., Zanghirati, G. (2006). Parallel software for training large scale support vector machines on multiprocessor systems. *J. Mach. Learn. Res.*, 7, 14671492.

[4] Wu, G., Chang, E., Chen, Y. K., Hughes, C. (2006). Incremental approximate matrix factorization for speeding up support vector machines. KDD 06: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 760766). New York, NY, USA: ACM Press

[5] Collobert, R., Bengio, S., Bengio, Y. (2002). A parallel mixture of svms for very large scale problems. *Neural Computation*, 14, 11051114

[6] Graf, H. P., Cosatto, E., Bottou, L., Dourdanovic, I., Vapnik, V. (2005). Parallel support vector machines: The cascade svm. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems* 17, 521528. Cambridge, MA: MIT Press.

[7] Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., Murthy, K. R. K. (2001). Improvements to Platts SMO Algorithm for SVM Classifier Design. Neural Comput., 13, 637649.

[8] Asuncion, A., Newman, D. (2007). *UCI machine learning repository*.

[9] B. Catanzaro, N. Sundaram, and K. Keutzer, Fast support vector machine training and classification on graphics processors, in *Proceedings of the 25th international conference on Machine Learning, ser. ICML* 08.ACM, 2008, pp. 104111.