GPU Implementation of Parallel Support Vector Machine Algorithm with Applications to Intruder Detection

Xueqin Zhang, Yifeng Zhang
East China University of Science and Technology, Shanghai, China
Email: zxq@ecust.edu.cn, outshine-george@163.com

Chunhua Gu
Shanghai University of Electric Power, Shanghai, China
Email: guchunhua@shiep.edu.cn

***Abstract*—The network anomaly detection technology based on support vector machine (SVM) can efficiently detect unknown attacks or variants of known attacks, however, it cannot be used for detection of large-scale intrusion scenarios due to the demand of computational time. The graphics processing unit (GPU) has the characteristics of multi-threads and powerful parallel processing capability. Based on the system structure and parallel computation framework of GPU, a parallel algorithm of SVM, named GSVM, is proposed. Extensive experiments were carried out onKDD99 and other large-scale datasets, the results showed that GSVM significantly improves the efficiency of intrusion detection, while retaining detection performance.**

***Index Terms*—Network Intrusion Detection, Support Vector Machine, GPU, Parallel Algorithm**

I. INTRODUCTION

[1] Intrusion detection is essentially a problem of classification. The anomaly detection technology based on machine learning can efficiently detect unknown attacks or variants of known attacks on the host or network. Support Vector Machine (SVM) is a powerful statistical learning algorithm based on VC dimension (Vapnik-Chervonenkis Dimension) and the structural risk minimizing theory [1]. SVM has been applied to many classification problems, such as speech reorganization and face gaze detection, with some degree of success [2-3]. However, solving the SVM problem involves a quadratic programming (QP) problem where memory and time complexity increase as a function of l^2, where l is the number of training samples. Consequently, for large-scale network intrusion detection problems, with a rapid increase of l, the SVM training becomes impractical due to huge costs. To reduce the training time of SVM, the original optimization problem of the SVM can be decomposed into a series of sub-problems with heuristic methods such as the Osuna decomposition algorithm [4], the Sequential Minimal Optimization (SMO) algorithm [5], the SVM^{light} algorithm [6], and the LIBSVM algorithm [7].However, a large-size SVM problem still requires a large amount of computational time.

The graphics processing unit (GPU) has a powerful computational capability with its high degree of parallelism. It has been widely used in general-purpose computing in recent years, such as in image and video processing, computational biology, power estimating [8-11]. In order to solve a large-scale network intrusion detection problem, a parallel algorithm of support vector machine, namely GSVM, is presented in this paper. This algorithm is based on the system structure and parallel implementation framework of the GPU. Extensive experiments were carried out on large-scale dataset and the results showed that GSVM can improve the training and testing speed of SVM Dramatically while retaining classification accuracy.

II. BASIC THEORY OF SUPPORT VECTOR MACHINE

A. Support Vector Machine for Classification

SVM seeks an optima hyper plane to separate two-class samples with the maximal margin.

Given training samples: $(x_1, y_1),...,(x_1, y_1), x \in R^d, l$ is the number of training examples, d is the dimension of feature vector, and $y_1$is the class label. SVM derives the weight vector $\boldsymbol{w}$ and offset b by solving the following optimization problem:

$$\min_{w,b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \tag{1}$$

$$\text{subject to } y_i(\mathbf{w}^T\phi(x_i)+b) \ge 1-\xi$$

$$\xi_i \ge 0, i = 1,2,...,l .$$

Where function $\phi()$ maps vector x_i from the input space into a high-dimensional feature space, C is the penalty coefficient, and ξ is a relax variable, $\xi_i \ge 0$.

[1]Manuscript received September 29, 2013; revised October 29, 2013; accepted October 30, 2013.
corresponding author: Chunhua Gu (guchunhua@shiep.edu.cn)

Based on the Wolfe Dual theory, the dual optimization problem of the above formulation can be expressed as:

$$\min_{\alpha} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \boldsymbol{p}^T\boldsymbol{\alpha} \quad (2)$$

subject to $\boldsymbol{y}^T\boldsymbol{\alpha} = 0,\ 0 \le \alpha_i \le C, i = 1,2,...,l$.

Where α_i is the Lagrange multiplier, $\boldsymbol{p} = [-1,-1,...,-1]^T$, Q is a $l \times l$ semidefinite matrix, $Q_{ij} \equiv y_i y_j \mathbf{K}(x_i, x_j)$, and $\mathbf{K}(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ is the kernel function. Common kernel functions include linear kernel, polynomial kernel, radial basis function (RBF) kernel and sigmoid kernel. When the kernel function is decided, $(a_1^*, ..., a_l^*)^T$ can be obtained by solving this problem.

The decision function is:

$$\operatorname{sgn}(\boldsymbol{w}^T\phi(\boldsymbol{x}) + b) = \operatorname{sgn}(\sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b) \quad (3)$$

Among them, sgn() is a sign function, $\boldsymbol{w}$ satisfies

$$\boldsymbol{w} = \sum_{j=1}^{i} y_i \alpha_i \phi(\boldsymbol{x}_i).$$

B. The Sequential Minimal Optimization Algorithm

When solving equation (2), as matrix Q is generally nonzero, a large amount of computer memory is required to store the kernel matrix for a large size problem, because the number of elements of $\boldsymbol{K}$ is equal to l^2.Cortes and Osuna etc. used a decomposition algorithm to solve this problem. They decomposed the primal large QP problem into a series of small QP problems and only a subset of training data are optimized at each step according to a certain strategy. The extreme case of this method is sequential minimal optimization (SMO) [5]. The SMO algorithm only allows a subset containing two samples to be optimized at each step. Although this approach increases the number of iterations, the problem size as well as the computational time is greatly reduced at each step, so that the total performance of the algorithms is improved dramatically.

The SMO algorithm is described as follows:

Step 1: Set $\boldsymbol{\alpha}^1 = [0,0,...,0]^T, \mathbf{G}^1 = [-1,-1,...,-1]^T$ as the initial feasible solution, and set the number of iterations $k = 1$. The gradient of $f(\boldsymbol{\alpha})$ is $\mathbf{G} = \nabla f(\boldsymbol{\alpha})$.

Step 2: If $\boldsymbol{\alpha}^k$ is an optimal solution of equation (2), then the algorithm terminates. Otherwise, find a subset of two elements to form work set $\mathbf{B} = \{i, j\} \subset \{1,...,l\}$ with WSS working set selection algorithm. Let $N \equiv \{1,...,l\}$, and define $\boldsymbol{\alpha}_B^k$ and $\boldsymbol{\alpha}_N^k$ as sub vectors corresponding to $\mathbf{B}$ and $\mathbf{N}$, respectively.

Step 3: Update $a_i, a_j, G_t (t = 1,...,l)$.

Step 4: Set $k \leftarrow k+1$, and jump back to step 2.

1. Algorithm termination conditions

According to the Karush-Kuhn-Tucker (KKT) conditions, the termination condition of the SMO is:

$$m(\boldsymbol{\alpha}^k) - M(\boldsymbol{\alpha}^k) \le \varepsilon. \quad (4)$$

Here, ε is the error threshold,

$$m(\boldsymbol{\alpha}) \equiv \max_{i \in I_{up}(\boldsymbol{\alpha})} - y_i \nabla f(\boldsymbol{\alpha})_i,\ M(\boldsymbol{\alpha}) \equiv \min_{i \in I_{low}(\boldsymbol{\alpha})} - y_i \nabla f(\boldsymbol{\alpha})_i,$$

$$I_{up}(\boldsymbol{\alpha}) \equiv \{t \mid \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = -1\}$$

$$I_{low}(\boldsymbol{\alpha}) \equiv \{t \mid \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = -1\}, t = 1,...,l$$

2. The working set selecting algorithm WSS

For the working set $\mathbf{B}$, considering the maximal violating pair, the working set selecting algorithm WSS is:

Step 1: For all of $t, s, t = 1,...,l, s = 1,...,l$, define:

$$a_{ts} \equiv K_{tt} + K_{ss} - 2K_{ts}, b_{ts} \equiv -y_t \nabla f(\boldsymbol{\alpha}^k)_t + y_s \nabla f(\boldsymbol{\alpha}^k)_s > 0 \quad (5)$$

and
$$\bar{a}_{ts} \equiv \begin{cases} a_{ts}, & \text{if } a_{ts} > 0 \\ \tau, & \text{otherwise} \end{cases}, \quad (6)$$

Then, select:

$$i \in \arg\max_t \{-y_t \nabla f(\boldsymbol{\alpha}^k)_t \mid t \in I_{up}(\boldsymbol{\alpha}^k)\}$$

$$i \in \arg\max_t \left\{-y_t \nabla f(\boldsymbol{\alpha}^k)_t \mid t \in I_{up}(\boldsymbol{\alpha}^k)\right\}$$

$$j \in \arg\min_t \left\{ -\frac{b_t^2}{\bar{a}_{it}} \mid t \in I_{low}(\boldsymbol{\alpha}^k) - y_t \nabla f(\boldsymbol{\alpha}^k)_t < -y_i \nabla f(\boldsymbol{\alpha}^k)_i \right\}$$

Step 2: Get $\mathbf{B} = \{i, j\}$.

The working set selecting is essentially a process to seek the extreme value of function.

3. Lagrange multiplier $\boldsymbol{\alpha}$ and Gradient $\nabla f(\boldsymbol{\alpha})$ updating

In each iteration, the SMO algorithm selects two Lagrange multipliers α_i, α_j and fixes others to make optimization. At the same time, it updates one of the two multipliers by adjusting the other one to follow the linear constraints. Lagrange multiplier α_i, α_j can be obtained as follows:

$$\alpha_j^{k+1} = \alpha_j^k - \frac{y_j[(\nabla f(\alpha^k)_i - \nabla f(\alpha^k)_j)]}{K(x_i, x_i) + K(x_j, x_j) - 2y_i y_j K(x_i, x_j)} \quad (7)$$

Applying constraint condition, then:

$$\alpha_j^{k+1} = \begin{cases} M & \text{if } \alpha_j^{k+1} \ge m \\ \alpha_j^{k+1} & \text{if } m < \alpha_j^{k+1} < M \\ m & \text{if } \alpha_j^{k+1} \le M \end{cases}$$

M is on the upper boundary of α_j^k and m is on the lower boundary of α_j^k. Then:

$$\alpha_i^{k+1} = \alpha_i^k + y_i y_j (\alpha_j^k - \alpha_j^{k+1}).$$

The gradients of all Lagrange multipliers are updated as follows:

$$\nabla f(a^{k+1})_t = \nabla f(a^k)_t + \Delta\alpha_i y_i K(x_i, x_t) + \Delta\alpha_j y_j K(x_j, x_t) \quad (8)$$

Here, $t = 1,...,l$, $\Delta\alpha_i = \alpha_i^{k+1} - \alpha_i^k$, $\Delta\alpha_j = \alpha_j^{k+1} - \alpha_j^k$.

4. Calculating value b

If α_i satisfies $0 < \alpha_i < C$, considering the KKT conditions, $b = -y_i \nabla f(\alpha)_i$, and to avoid the numerical error, let:

$$b = -\frac{\sum_{0<\alpha_i<C} y_i \nabla f(\alpha)_i}{\sum_{0<\alpha_i<C} 1}. \qquad (9)$$

Otherwise, the condition becomes:

$$m(\alpha) \le b \le M(\alpha)$$

Here,

$$M(\alpha) = \max\{y_i \nabla f(\alpha)_i \mid \alpha_i = 0, y_i = -1 \text{ or } \alpha_i = C, y_i = 1\}, i = 1,\dots,l$$

$$m(\alpha) = \min\{y_i \nabla f(\alpha)_i \mid \alpha_i = 0, y_i = -1 \text{ or } \alpha_i = C, y_i = 1\}, i = 1,\dots,l$$

Then the value of midpoint in this range is b :

$$b = \frac{M(\alpha) + m(\alpha)}{2}. \qquad (10)$$

III. PARALLELANALYSIS AND DESIGN OF SMO ALGORITHM

A. Parallel Analysis

According to the above analysis, it can be observed that:

1) In the sequential SMO algorithm, the optimization process accounts for most of the computation time. In particular, over 90% of the total computational time is used for updating $\nabla f(\boldsymbol{\alpha})$ [12]. However, according to (9), updating $\nabla f(\boldsymbol{\alpha})$ is evaluated one at a time. This is suitable for parallelization. Therefore, updating can be implemented in a parallel design.

2) In the SMO algorithm, kernel evaluations consist of a number of steps (e.g. the calculation of $\boldsymbol{\alpha}$, $\nabla f(\boldsymbol{\alpha})$ and the decision function). The calculation involves examining all of the training data points (e.g. dot product operation).Although there are a very large amount of floating-point calculations, the instructions are very simple. As GPU is very effective in dealing with this kind of SIMD problems, the kernel evaluations can also be performed in a parallel fashion.

3) The WSS algorithm is a process to seek the maximal or the minimal value, which requires comparison of all the data points one by one in order to obtain the extremum. The efficiency of the CPU is very low in dealing with this kind of problem. By using the GPU parallel reduction algorithm, the global problem can be decomposed into local problems for acceleration.

4) In the SMO algorithm, a lot of matrix operations, such as multiplication and cumulative sum, are involved. These operations also belong to simple instruction problems and can be executed in a parallel design by the GPU to a high degree.

5) The efficiency of the GPU is poor in processing conditional branch problems. However, in the SMO algorithm, each iteration is terminated by conditional decisions. For large-size problems, the number of executing branch statement will increase greatly. Therefore this part should still be implemented on the CPU.

B. Parallel Design of SMO Algorithm based on GPU

In order to improve the efficiency in the determination of intrusion with large scale problems, a parallel SVM algorithm based on GPU and CUDA (Compute Unified Device Architecture) [13] named GSVM is developed. GSVM consists of two independent parts: training and testing. The framework of these two parts based on the GPU is shown in Figures 1 and 2.

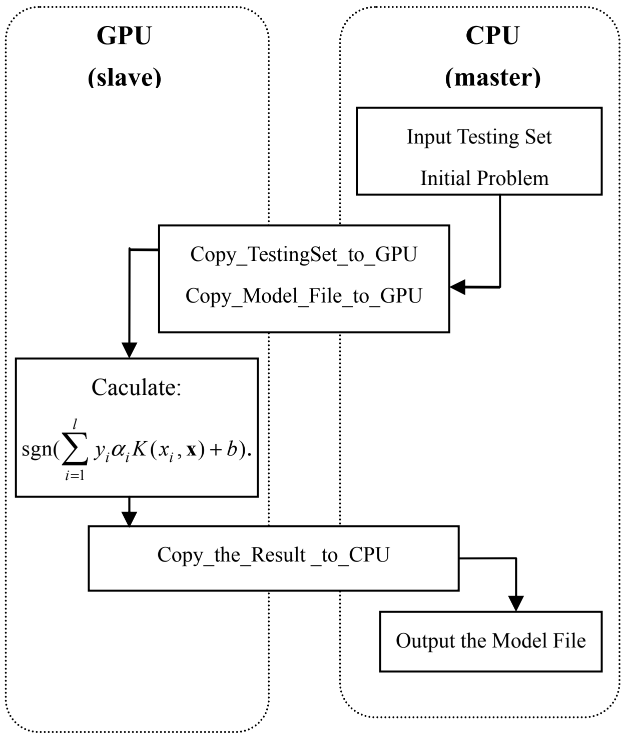

Figure 1. GSVM classification algorithm process

C. Time Complexity Analysis

For sequential SMO algorithm, updating $\nabla f(\boldsymbol{\alpha})$ and selecting working set B are the most time consuming steps.Cao and co-workers showed that,in each of iteration, if the time complexity of kernel calculating is $O(l)$,the time complexity of the whole algorithm is $\#Iter \times O(ld)$ when the most of columns of matrix Q are not in the cache; otherwise it is $\#Iter \times O(l)$ [12]. Here, *#iter* denotes the number of iterations and is proportional to*l*.

For parallel SMO algorithm, let p denotes the number of threads used, #iter' represents the number of iterations in GPU end. If the most columns of matrix Q are in the cache, calculation is insignificant. Therefore the time complexity is $\#Iter' \times O(l)$. Otherwise, it is $\#Iter' \times O(kl + \Delta)$ .Here, Δ is the cost of threads synchronization and k is a coefficient associated with d and p.

IV. APPROACHES USED TO IMPROVE GPU PARALLEL COMPUTING PERFORMANCE

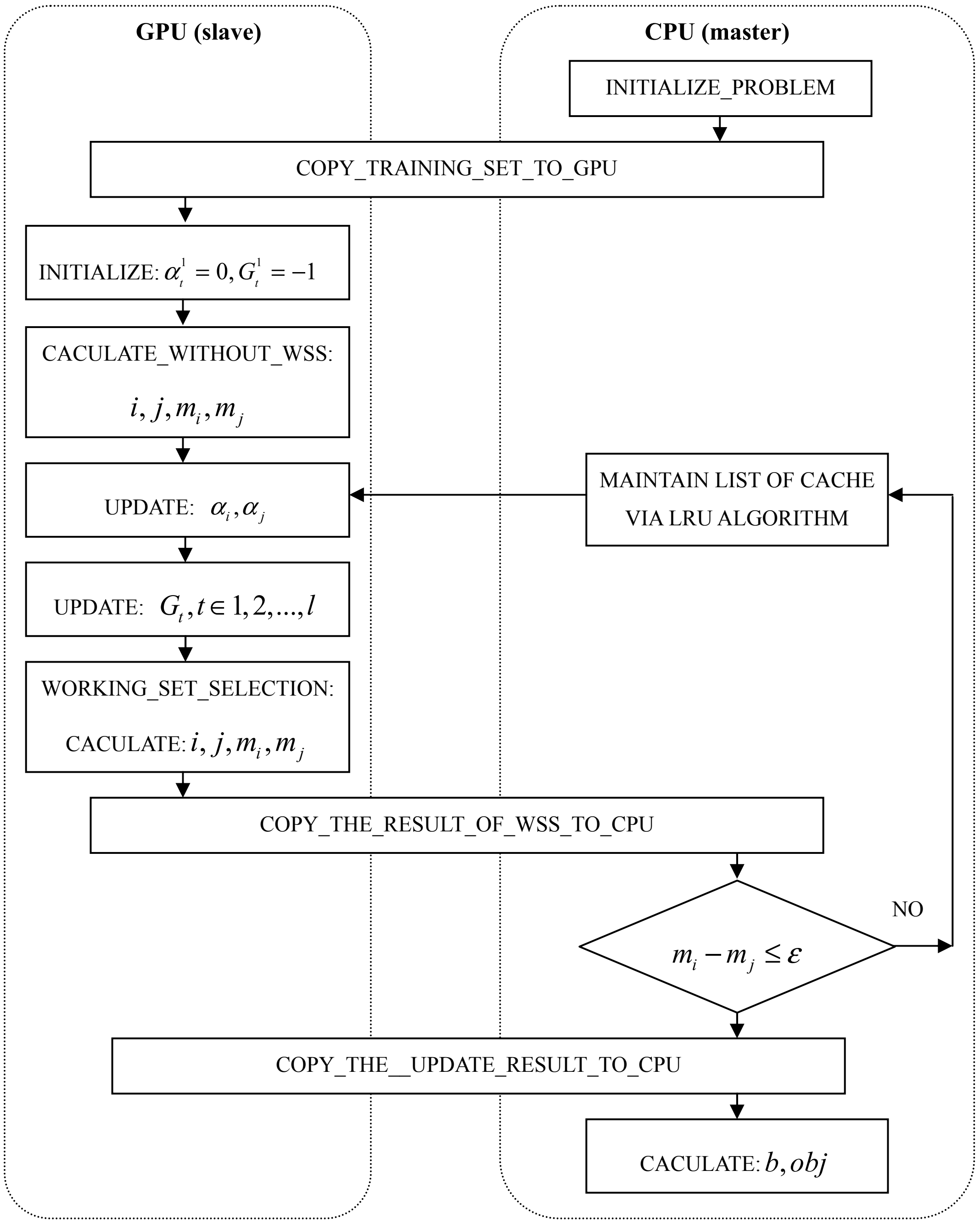
GPU (slave)
CPU (master)
INITIALIZE_PROBLEM
COPY_TRAINING_SET_TO_GPU
INITIALIZE: $\alpha_t^1 = 0, G_t^1 = -1$
CACULATE_WITHOUT_WSS: i, j, m_i, m_j
UPDATE: α_i, α_j
MAINTAIN LIST OF CACHE VIA LRU ALGORITHM
UPDATE: $G_t, t \in 1, 2, ..., l$
WORKING_SET_SELECTION: CACULATE: i, j, m_i, m_j
COPY_THE_RESULT_OF_WSS_TO_CPU
$m_i - m_j \le \varepsilon$
NO
COPY_THE__UPDATE_RESULT_TO_CPU
CACULATE: b, obj

Figure2. GSVM training algorithm process

A.SPRG Parallel Computing Method

SVM training and testing processes involve matrix operations such as multiplication, cumulative sum and seeking extreme value. The computational complexities of these operations are proportional to the size of data sets. GSVM algorithm uses parallel reduction methods to optimize these calculation processes. The basic idea of this method is described as follows. First, the data are divided into *n* parts and transferred to parallel computing nodes. Second, each computing node summarizes its data and executes corresponding operations, such as multiplication and addition. Finally, each parallel computing node transmits its operation results to the aggregation node for implementing the last operation. Because all of the data are split and computed in parallel, the total computational time is reduced.

To be specific, in this paper, the SPRG (scatter-parallel-reduce-gather) parallel reduction method is designed as follows.

1) Scatter: Data sets are partitioned into smaller subsets according to the number of blocks and threads used and moved from global memory to shared memory in line with the corresponding address sequence. Considering that GPU usually accesses the shared memory in 4 clock cycles, far less than the 400-600 clock cycles that are needed to access the global memory, this operation also improves the subsequent access speed.

2) Parallel-reduce: All of the threads in blocks execute the same instruction and obtain the reduction sub-results.

These results are stored in the head address units of blocks.

3) Gather: Considering that the data in the GPU blocks are unable to communicate with each other, the sub-results in each block need to be transferred from the shared memory back to the global memory according to the corresponding address sequence in order to do the further reduce.

4) Repeat steps 1 and 2 until the sub-results are reduced in one block. The final result is output to the global memory for ease of communication with CPU end.

Figure 3 shows the process of solving the extreme value based on the SPRG method.

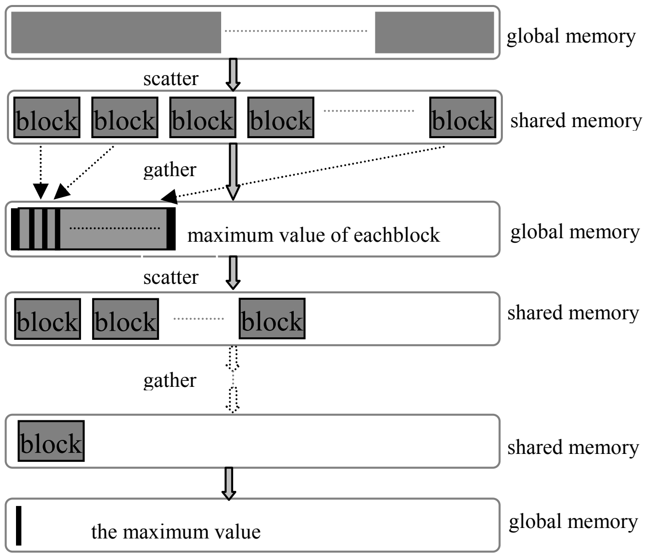

Figure 3.extreme value solving based on SPRG

In the GSVM algorithm, the SPRG method is widely used, such as to solve the gradient extremum in the training period and achieving the cumulative sum in the testing period. If the time required for data transmission and synchronization can be ignored, the computational complexity can then be decreased by one order of magnitude compared with the serial algorithm. This has a great advantage in processing large-scale arrays.

B. Optimization Methods

1) Cache technology: In sequential SMO, cache technology is used to store calculated results of kernel function and LRU (Least Recently Used) algorithm is used to manage a two-way circular linked list of cache. In GSVM, considering the complexity of the list structure and LRU algorithm, the task of maintaining the list is still executed by the CPU, while caches are allocated in the GPU. In each iteration, when the working set selection is completed, a judgment will be made by CPU end (Host end), and then inform GPU end (Device end) to execute the subsequent operations.

2) Algorithmic Optimizations: ① In the shared memory, when applying the reduction technology with tree-based interleaved addressing mode, bank conflicts in the shared memory usually occur and result in memory access operation serialization. In this case, the effective bandwidth decreases exponentially. Therefore the sequential addressing principle should be followed to unify the operations of all the threads in warps to avoid this problem.②In addition, when branch statements(e.g. "if-then-else") are executed by threads in one warp, the CUDA GPU deals with these threads sequentially. As such branch statements should be avoided in one warp.

3) Instruction Optimizations: ①The ability of GPU's integer processing unit is weak, therefore the integer modulo and division operation costs are usually very high. Thus the bit operation is adopted instead.②In the CUDA programming, all the threads in a block typically need to be synchronized in each loop. However, since the operations in one warp always meet consistency, this step is not needed at this case. Loop unrolling is a common instruction optimization method to improve the execution speed, even for very long expression. In many cases, loop unrolling can improve performance by 20%. Thus for an inner loop, the loop unrolling method can be used to deal with the last few iterations in order to accelerate the speed.

V. Experiment and Conclusions

A. The Experiment Environment

In our experiment, the CPU is Intel Pentium dual-core E5500 with a frequency of 2.80GHz. The GPU is NVIDIA GeForce GTS 250.The memory capacity is 512 MB, the number of multiprocessors is 16, the number of cores (stream processors) is 128, multiprocessors shared memory size is 16KB, and memory bandwidth is 70.4 GB/S. The integrated development environment is Microsoft Visual Studio 2008 with CUDA SDK 2.3. LIBSVM is used as the sequential SVM algorithm.

B. Experiment and Results

1. Experiment 1

(1) Description of the Experimental Data

Nine groups of large-scale datasets from the field of machine learning are used to verify the validity and superiority of the GSVM algorithm. The size and dimension of these datasets are shown in TABLE I. Here, #Tr is the number of training samples, and #Te is the number of testing samples.

(2) Experimental in SVM Training Period

In the SVM training period, the LIBSVM uses the default parameters. Set the RBF kernel as the kernel function, $C=1,\varepsilon=0.001$.

The LIBSVM and GSVM training algorithms were executed for the same training data set to derive the training models. The number of the support vectors, iterations and the training time, as shown in TABLE II. Here, *#SV* indicates the number of the support vectors, *#iter* denotes the number of iterations, *T* represents training time, and *ratio* is the speed-up ratio. In order to display the result visually, the comparison of training time is also shown in Figure 4.

(3) Experiment in testing period

The LIBSVM and GSVM testing algorithms were executed for the same test dataset with the two models built above to compare the classification accuracy and testing time. Classification results are shown in TABLE III. In this table, *T* denotes testing time, and *Acc* is the accuracy. The comparison of the testing time is also listed in Figure 5.

(4). Experimental Analysis and Conclusions

From the above results, it can be seen that:

1) The numbers of support vectors, iterations and the classification accuracy obtained from GSVM and LIBSVM are virtually the same. These demonstrate the correctness of the GSVM algorithm.

2) Comparing the experiment result of GSVM with LIBSVM on large-scale datasets, the training time decreased 2 to 43 times, and the classification time was reduced 40 to 349 times. These results demonstrate the superiority of the GSVM algorithm in computational speed.

3) Comparing the parallel training algorithm with the testing algorithm, the performance improvement of the later is remarkable due to the absence of conditional branch operations in the testing period. The testing part in SVM is more suitable to be parallelized.

4) Cost for communication between CPU and GPS is always considered as the bottleneck for GPU parallel implementation. It is noted that the much improved computational speed from GSVM already included this cost. As such, this algorithm could find valuable practical applications.

2. Experiment 2

(1) Description of Experiment Data

In order to verify the effectiveness of GSVM in intrusion detection, KDD99 dataset is adopted in this set of experiments. The numbers of the training samples and the test samples are 49407 and 49405, respectively.

(2) Results and conclusions

For better understanding of the cost of various subparts in the sequential and parallel SMO, the computation time in different steps (initialization, optimizing $\boldsymbol{\alpha}$, updating $\nabla f(\boldsymbol{\alpha})$ and selecting work set **B**) are listed in TABLE III. The table shows that updating $\nabla f(\boldsymbol{\alpha})$ and selecting work set **B** cost large amount of training time, and is better performed in parallel.

The experiment evaluates the intrusion detection system based on GSVM from the training time, testing time, the accuracy (Acc), the detection rate (DR) and the false positive rate (FP). The experiment results are shown in TABLE IV, V and VI.

VI. Conclusions

In order to solve the SVM time-complexity problem, a parallel SVM algorithm based on the GPU, namely GSVM, is presented in this paper. According to the parallel implementation architecture of GPU, the training and testing part of the sequential SVM algorithm are parallelized. Additional optimization measures are designed to improve the performance of the GSVM. Experimental results derived from large-scale datasets show that the GSVM algorithm can speed up SVM training and testing process dramatically without loss of classification accuracy. The speedup effectiveness becomes more obvious with the increase of the dataset size. The GSVM algorithm has good real-time processing capabilities on massive data and can be utilized on abnormality detection effectively.

From the experimental results on KDD99, it was concluded that the intrusion detection system based on the GSVM parallel algorithm has virtually the same accuracy, detection rate and false positive rate compared with results obtained from the LIBSVM algorithm. Moreover, the training and classification time are decreased by 13.3 times and 64.92 times, respectively, in the GSVM parallel algorithm. Thus, the GSVM parallel algorithm based on the GPU is very suitable for constructing intrusion detection classifier.

TABLE I.
THE DATA SET DESCRIPTION

Dataset	#Tr	#Te	Dim	Source
cod-rna	59,535	271,617	8	BMC
covtype	300,000	281,012	54	UCI
epsilon	20,000	20,000	2000	PASCAL Challenge2008
ijcnn1	49,990	91,701	22	IJCNN
webspam	150,000	200,000	128	CEAS
face	1,996,201	2,005,601	10	FRGC
sonar	10,000	9,375	60	UCI
adult	32,561	16,281	123	UCI
w8a	49,749	14,951	300	UCI

TABLE II.
COMPARISON OF SVM TRAINING RESULTS

Data set	CPU Training			GPU Training			ratio
	#SV	#iter	T (sec)	#SV	#iter	T (sec)	
cod-rna	16,179	65,638	1,301	16,214	65,712	123	10.542
covtype	299,997	489,969	128,770	299,998	490,404	2,968	43.381
epsilon	19,833	10,038	4,575	19,834	10,102	107	42.766
ijcnn1	8,970	5,563	119	8,986	5,713	11	11.912
webspam	63,930	35,854	8,867	63,926	35,791	254	34.936
face	3,842	2,178	1,396	3,841	2,277	45	30.737
sonar	1,430	1,271	7	1,434	1,718	3	2.33
adult	11,951	7,848	300	11,951	8,186	16	18.75
w8a	2,949	2,843	208	2,944	2,824	9	23

TABLE III.
COMPARISON OFTHE COMPUATION TIME IN DIFFERENT STEPS OF TRAINING PERIOD

CPUTime (sec)			GPU Time (sec)		
Initialization	α	$\nabla f(\boldsymbol{\alpha})$ +WSS	Initialization	α	$\nabla f(\boldsymbol{\alpha})$ +WSS
0.0	0.0	24.8	0.0	0.0	1.63

TABLE IV.
COMPARISON OF TRAINING RESULTSON KDD99

CPU			GPU		
Acc(%)	DR	FP	Acc(%)	DR	FP
98.92	98.33	0.188	98.92	98.32	0.188

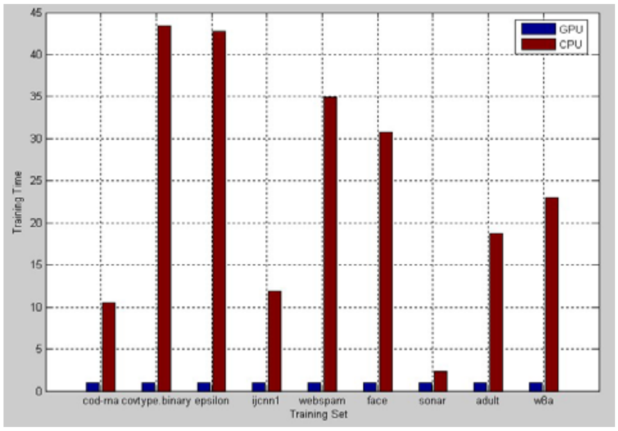

Figure 4.comparison of the training time of GPU and CPU

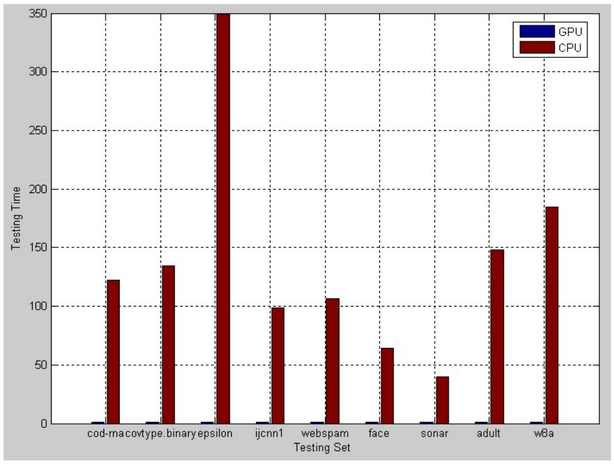

Figure 5.comparison of the classification time of GPU and CPU

TABLE VI.
COMPARISON OF SVM CLASSIFICATION RESULTS

Data set	CPU Training			GPU Training			ratio
	#SV	#iter	T (sec)	#SV	#iter	T (sec)	
KDD99	975	849	26.1	976	998	2.03	13.3

TABLE V.
KDD99 CLASSIFICATION PERFORMANCE

Data set	CPU Classification		GPU Classification		ratio
	T (sec)	Acc (%)	T (sec)	Acc (%)	
KDD99	17.4	98.917	0.27	98.915	64.96

REFERENCES

[1] N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge, 2000.

[2] B.Yu, H.F.Li, C.Y. Fang."Speech Emotion Recognition based on Optimized Support Vector Machine", *Journal of Software,* vol. 7, no. 12, pp.2726-2733, 2012.

[3] M.h. Zhang, L.h. Gang."A Detection Method of Driver's Face Orientation Based on Visual Cues and SVM",*Journal of Software*, vol. 8, no. 4,pp.924-931,2013.

[4] E.Osuna, R.Freund, F.Girosi, "*An improved training algorithm for support vector machines*",*Proc. IEEE Workshop on Neural Networks and Signal Processing*, Piscataway: IEEE Press,1997, pp. 276-285.

[5] J. Platt,"Fast training of support vector machines using sequential minimal optimization", *Advances in Kernel Methods-Support Vector Learning*, Cambridge, MA, 1998, pp. 185-208.

[6] T. Joachims,"Advances in kernel methods-support vector learning", Cambridge, USA, 1998.

[7] C. Chih-Chung, L. Chih-Jen, LIBSVM: a Library for Support Vector Machines.[online].Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[8] K. Yadav,A. Mittal, and M. A. Ansar, et al., "Parallel Implementation of Compressed Sensing Algorithm on CUDA- GPU", *International Journal of Computer Science and Information Security*, vol. 9,no.3,2011.

[9] B. Pieters,D. Van,R.Wesley,et al., "Performance Evaluation of H.264/AVC Decoding and Visualization using the GPU," *Applications of digital image processing* , vol.6696,no.1,2007.

[10] Meredith, J.S.Alam, S.R.Vetter, et al. , "Analysis of a Computational Biology Simulation Technique on Emerging Processing Architectures. Parallel & Distributed Processing Symposium (IPDPS)",*2007 IEEE International*, Long Beach, CA. 2007,vol. 1.

[11] H.f.Wang, Q.k. Chen. "Power Estimating Model and Analysis of General Programming on GPU". *Journal of Software*, vol. 7, no. 5, pp.1164-1170,2012

[12] L.Cao, S.Keerthi, and Ong, et al., "Parallel sequential minimal optimization for the training of support vector machines", *IEEE Transactions on Neural Networks*, vol.17,pp.1039-1049,2006.

[13] "NVIDIA Corporation NVIDIA CUDA C Programming Guide3.2," 2010,[online].Available: http://developer.download.nvidia.com/compute/cuda/3_2 /toolkit/docs/CUDA_C_ Programming_Guide.pdf.

Xueqin Zhang received the Ph.D.degreein Detection Technology and Automation Devices from East China University of Science and Technology (ECUST), Shanghai, China, in 2007. Since 1998, she has been in the Electrical and Communication Engineering Department, ECUST, where she is currently an ASSOCIATE PROFESSOR.

At 2006, she worked as a visiting scholar in University of Wisconsin Madison. Her research interests include pattern classification, information security and data mining etc.

Yifeng Zhang received the M.S. degree in Electrical and Communications from East China University of Science and Technology (ECUST), Shanghai, in 2012. His research interests are parallel computation, pattern classification etc.

Chunhua Gu received the Ph.D. degree in Control Science and Engineering from East China University of Science and Technology (ECUST), Shanghai, China, in 2007. From 1992 to 2013, he was in the Computer Science and Engineering Department, School of Information Science and Engineering, ECUST, where he is currently a PROFESSOR. Now he is in the School of Computer Science and Technology at Shanghai Institute of Electric Power.

At 2002, he worked as a visiting scholar in School of Computing and Information Sciences, Florida International University. His research interests include intelligent computing, software engineering and information security.