High Performance Relevance Vector Machine on GPUs

Depeng Yang, Getao Liang, David D. Jenkins, Gregory D. Peterson, and Husheng Li
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN, 37996

***Abstract*—The Relevance Vector Machine (RVM) algorithm has been widely utilized in many applications, such as machine learning, image pattern recognition, and compressed sensing. However, the RVM algorithm is computationally expensive. We seek to accelerate the RVM algorithm computation for time sensitive applications by utilizing massively parallel accelerators such as GPUs. In this paper, the computation procedure of the RVM algorithm is fully analyzed. Recursive Cholesky decomposition, the key step in the RVM algorithm, is implemented on GPUs. The GPU performance is compared with a CPU using LAPACK and a hybrid system using the MAGMA library. Results show that our GPU implementation in both single and double precision is approximately 4 times faster than the CPU using LAPACK and faster than the hybrid MAGMA code when the matrix size is small.**

I. Introduction

Relevance Vector Machine (RVM) algorithm [1] shows lots of advantages compared with the traditional Support Vector machine (SVM) algorithm and is widely utilized in various fields, such as machine learning, image pattern recognition, and compressed sensing [2] [3]. However, the RVM algorithm is associated with intensive matrix operations, such as matrix-matrix multiplication, Cholesky decomposition, and matrix inversion, leading to unacceptably large computation time. This hampers applying the RVM algorithm into time-sensitive applications, such as real-time video pattern recognition, on-line model regression, and fast signal reconstruction. Modern GPUs enable massively parallel computation, which is suitable for accelerating the RVM algorithm.

This paper discusses implementing the RVM algorithm on GPUs for acceleration. The RVM computation procedure is firstly fully analyzed in order to exploit the massively parallel computation on GPUs. In the RVM algorithm, the key step is to calculate the Cholesky decomposition recursively. In contrast to the traditional Cholesky decomposition, we implement a right-looking blocked LDL^T Cholesky decomposition on GPUs based on the previous results in the computation loop.

Performance of the GPU implementation is compared with that of a CPU implementation using LAPACK and a hybrid system using MAGMA. We compare the execution times of the recursive Cholesky decomposition on the GTX480 GPU card using OpenCL, on a CPU using LAPACK, and on a hybrid system using MAGMA. Results show that our GPU implementation in both single and double precision is approximately 4 times faster than LAPACK and faster than the hybrid mode using the MAGMA library when matrix size is small.

II. RVM computation procedure

The RVM regression problem can be expressed as:

$$\boldsymbol{t} = \boldsymbol{\Phi w} + \boldsymbol{\epsilon}, \tag{1}$$

where the weight vector is $w :\in R^{1\times N}$, the projecting matrix is $\Phi :\in R^{M\times N}$, and the $t :\in R^{1\times M}$ is the measurement vector. The additive noise ϵ is Gaussian distributed with zero-mean and a variance of β^2.

Given the vector t, the matrix Φ, and noise variance β^2, the RVM regression algorithm computes the weight vector w. Actually, there are many ways to solve the RVM regression problem, but the Expectation-maximization (EM) method [1] and the fast RVM method [2] are the two main ways. Because of its intrinsic parallelism, the EM method is very suitable for parallel computation. The weight vector w can be recursively solved. Assume at the i-th iteration, the weight vector w^i is solved by:

$$\boldsymbol{w^i} = \beta^{-2}\boldsymbol{\Sigma^i}\boldsymbol{\Phi^i}^T\boldsymbol{t} \tag{2}$$

$$\boldsymbol{\Sigma^i} = (\beta^{-2}(\boldsymbol{\Phi^i})^T\boldsymbol{\Phi^i} + \boldsymbol{A})^{-1} \tag{3}$$

where $\boldsymbol{A}$ is a diagonal matrix containing hyperparameters on the diagonal line. The symmetric $\boldsymbol{\Phi^i}$ matrix is calculated from the given matrix Φ^i, A, and the vector t.

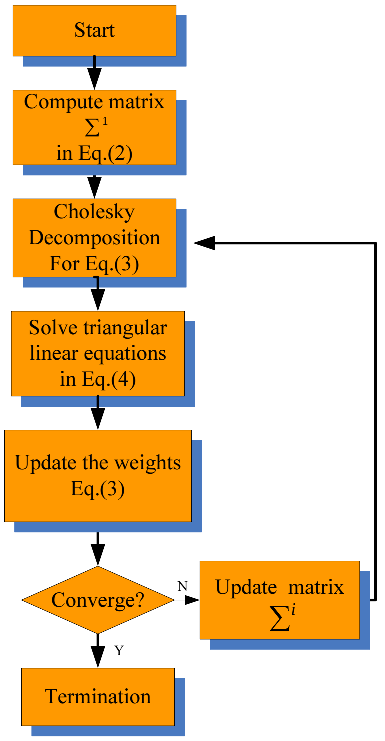
Based on the symmetric $\boldsymbol{\Phi^i}$ matrix, the weight vector w^i can be solved by utilizing the Cholesky decomposition, which is given by:

$$\boldsymbol{\Sigma^i} w^i = (LDL^T)^i w^i = (\boldsymbol{\Phi^i})^T\boldsymbol{t} \tag{4}$$

where L is a unit lower triangular matrix and D is a diagonal matrix [4]. Then, $\boldsymbol{\Sigma^i} = (LDL^T)^i$ is the most computational expensive step, which has $O(\frac{1}{3}N^3)$ complexity at each iteration.

At the $(i+1)$-th iteration computation, the matrix $\boldsymbol{\Sigma^{i+1}}$ is augmented based on the previous matrix $\boldsymbol{\Sigma^i}$. Then the weight vector w^{i+1} is updated by solving Eq. (4). Note that $\boldsymbol{\Sigma^{i+1}}$ cannot be obtained until the previous weight vector w^i is solved. This computation does not stop until the convergence condition is satisfied. The convergence condition is to satisfy the term, for any $w_j \in w$, $\|w_j^n - w_j^{n-1}\| < \theta$ at the n-th iteration, where θ is set to a small value.

The computation procedure is illustrated and summarized in Fig.1. The computation procedure requires matrix-vector and matrix-matrix multiplication, and Cholesky decomposition. Inside the computation loop, the main step is to calculate the Cholesky decomposition for the augmented symmetric matrix each iteration.



Start
Compute matrix Σ^1 in Eq.(2)
Cholesky Decomposition For Eq.(3)
Solve triangular linear equations in Eq.(4)
Update the weights Eq.(3)
Converge?
N
Update matrix Σ^i
Y
Termination

Fig. 1: Computation procedure for the RVM algorithm

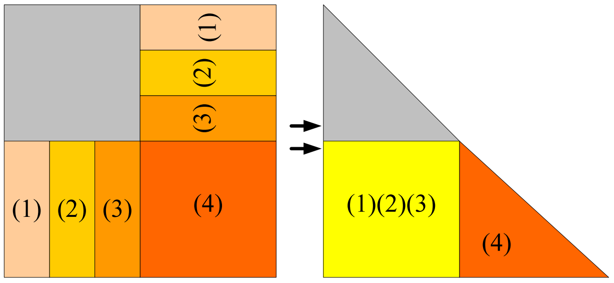


Fig. 2: Computation procedure on GPUs for recursive Cholesky decomposition

III. Implementation on GPUs

The most computationally expensive step is the Cholesky decomposition for the augmented matrix $\mathbf{\Sigma}$ each iteration. Other computations in RVM including matrix-vector and matrix-matrix addition and multiplication, can be solved by calling level-2 and level-3 subroutines in CUBLAS [7], which has been optimized manually and automatically for the best performance. This paper will only focus on recursive Cholesky decomposition for the RVM computation acceleration on GPUs.

A. *GPU implementation*

A recursive LDL^T Cholesky decomposition for the RVM algorithm is proposed and implemented on GPUs for acceleration. Each iteration, a Cholesky decomposition is performed for the symmetric matrix Σ^{i+1} based on the previous results $\Sigma^i = L^i D^i (L^i)^T$. In contrast to the normal Cholesky decomposition, our problem is how to utilize the previous results $L^i D^i (L^i)^T$ for factoring the augmented matrix Σ^{i+1}.

Assume at the $(i+1)$-th iteration, the Cholesky decomposition can be written as:

$$\begin{aligned}\Phi^{i+1} &= \left(\begin{array}{c|c}\Phi^i & \Phi_a^T \\ \hline \Phi_a & \Phi_b\end{array}\right) = L^{i+1}D^{i+1}(L^{i+1})^T \\ &= \left(\begin{array}{c|c}L^i & 0 \\ \hline L_a & L_b\end{array}\right)\left(\begin{array}{c|c}D^i & 0 \\ \hline 0 & D_b\end{array}\right)\left(\begin{array}{c|c}(L^i)^T & L_a^T \\ \hline 0 & L_b^T\end{array}\right)\end{aligned}$$

where $\Phi^i = L^i D^i (L^i)^T$ are the previous factoring results. Note that L_a is a dense matrix and L_b is a symmetric matrix. Given the matrix Φ^i and its decomposition, the Cholesky decomposition of the matrix Φ^{i+1} is obtained as long as the matrix L_a is solved. Then we have:

$$L_a = \Phi_a (L^i D^i)^{-T} \rightarrow L^i D^i L_a^T = \Phi_a^T \tag{5}$$

To solve the dense matrix L_a, one needs to solve the triangular linear equations (forward substitution) since $L^i D^i$ is a lower triangular matrix. On GPUs, the matrix L_a is divided into several strip matrices for massively parallel computation. Computations are executed along the column vectors for the best performance.

Fig. 2 demonstrates the computation procedure of our recursive Cholesky decomposition on GPUs. Note that the big grey left-top matrix is previously obtained, which is $\Phi^i = L^i D^i (L^i)^T$. WLOG, we illustrate (1)(2)(3) steps to solve the dense matrix L_a along the strip column vectors. And in the step (4), we adopt a "right-looking" algorithm [10] [5] for LDL^T Cholesky decomposition for the best performance.

We optimize the GPU implementation from many perspectives. The matrix is partitioned to 16x16 blocks to improve efficiency. Shared memory is fully utilized. We also optimize memory access patterns. Data transfers between global and shared memory are performed in a consecutive manner [12]. Banking conflicts in shared memory are avoided by increasing the memory volume. All these efforts benefit the final GPU performance.

IV. Results

The recursive Cholesky decomposition for the RVM algorithm is implemented and tested on GPUs and CPUs. The testing system includes a 2.67GHz Intel Quad Core i7 with 12 GB RAM and an NVIDIA GeForce GTX 480 GPU card [11]. The software compilers include gcc (version 4.3.3) and NVIDIA CUDA/OpenCL (version 3.0) [6] [12]. We implement our algorithm in OpenCL, which make it suitable for different GPU cards without changing any codes. We also utilize the MAGMA and LAPACK libraries for Cholesky decomposition in the RVM algorithm. For comparison purposes, we separate the Cholesky decomposition step in the RVM computation loop. The execution time is only for the recursive Cholesky decomposition for the RVM algorithm. Other computation in the RVM algorithm, including matrix-vector and matrix-matrix multiplication, and addition is ignored here because the Cholesky decomposition dominates the runtime.

Fig.3 shows the performance comparison in single precision on GPUs, CPUs using LAPACK [9], and a hybrid mode using the MAGMA library [8]. Assume at each iteration,

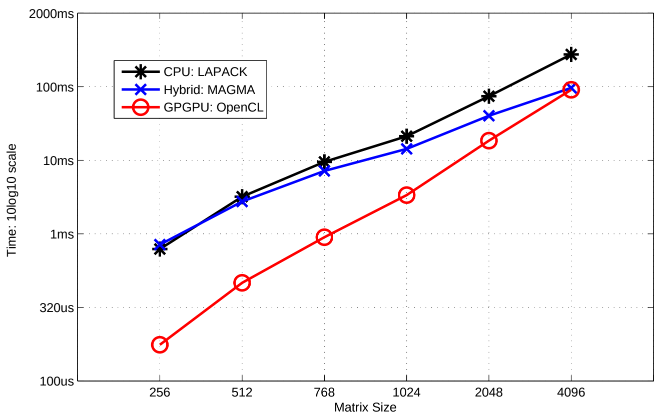


Fig. 3: Comparison of computation time for Relevance Vector Machine in single precision

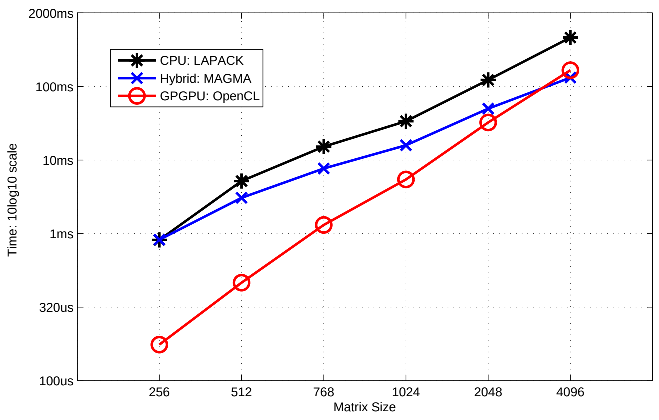


Fig. 4: Comparison of computation time for Relevance Vector Machine in double precision

the symmetric matrix is increased by 256x256. At the first iteration the matrix size is 256x256. The second iteration the matrix size is increased to 512x512. For convenience, the x-axis only marks the matrix size at some iteration. For the 4096x4096 matrix decomposition, in the RVM computation loop the recursive Cholesky decomposition has to start with the first 256x256 matrix. Our recursive Cholesky decomposition can take advantage of previous results to improve the performance while the LAPACK and MAGMA libraries have to do traditional Cholesky decomposition iteratively. Note that MAGMA is in a hybrid mode where serial computation tasks are performed on the CPU while the massively parallel computation tasks, such as matrix-matrix multiplication, are executed on the GPU simultaneously for the best performance. Also the data communication from host CPU to the slave GPU is well hidden by using streaming functions. In Fig. 3, it is observed that our GPU implementation averages about 4 times speedup over the CPU using LAPACK at all iterations from the small 256x256 matrix to the 4096x4096 matrix. Compared with the MAGMA library, our GPU implementation is still better when the matrix size is small. With the growth of the matrix, the performance gap between the pure GPU and the hybrid MAGMA decreases. When the matrix size is 4096x4096, the execution time on the GPUs is almost the same as the time using the MAGMA library.

Fig. 4 shows a performance comparison in double precision on GPUs, CPUs using LAPACK, and a hybrid mode using the MAGMA. Compared with Fig. 3, our GPU performance is still much faster than LAPACK at all iterations. And our double precision performance is almost as good as the single precision performance because double precision performance in a GTX480 GPU can achieve up to one half of single precision peak performance, which is much better than the previous GPU products such as NVIDIA Tesla. When the matrix size is 4096x4096, performance using MAGMA in the hybrid mode is better than our GPU implementation in double precision.

V. Conclusion

This paper proposes a recursive Cholesky decomposition for accelerating the RVM algorithm computation on GPUs. The RVM computation procedure is analyzed. The recursive LDL^T Cholesky decomposition is implemented and tested on GPUs. Performance comparison shows that our GPU implementation in single and double precision is much faster than the CPU using LAPACK and faster than the hybrid mode using MAGMA for smaller matrix sizes.

References

[1] M. E. Tipping, “Sparse Bayesian learning and the relevance vector machine”, *Journal of Machine Learning Research*, vol.1, pp.211-244, 2001.
[2] D. Yang, H. Li, and G. D. Peterson, ”Space-time Turbo Bayesian Compressed Sensing for UWB system,” *IEEE Information Conference on Communications (ICC) 2010*, Kapton, South Africa, 2010.
[3] D. Yang, H. Li, G. D. Peterson and A. E. Fathy, “UWB Signal Acquisition in Locationing Systems: Compressed Sensing and Turbo Signal Reconstruction,” *Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, Mar., 2009.
[4] D. Yang, G. D. Peterson, H. Li and J. Sun, “An FPGA Implementation for Solving Least Square Problem,“ *The 17th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, California, April., 2009.
[5] D. Yang, J. Sun, G. Tao, D. Jenkins, G. D. Peterson and A. E. Fathy, “Performance comparison of Cholesky decomposition on GPGPUs and FPGAs,” *Symposium on Application Accelerators in High Performance Computing (SAAHPC)*, Knoxville, TN, July, 2010.
[6] NVIDIA CUDA programming guide. [Online] http://www.nvidia.com/object/cuda_home.html#
[7] NVIDIA CUBLAS library. [Online] http://developer.download.nvidia.com/compute/CUBLAS_Library
[8] Matrix Algebra on GPU and Multicore Architectures (MAGMA) user guide. [Online] http://icl.cs.utk.edu/magma/
[9] Linear Algebra PACKage (LAPACK), Version 3.2.1 [Online] http://www.netlib.org/lapack/
[10] Cholesky factorization algorithm. [Online] http://userweb.cs.utexas.edu/~plapack/icpp98/node2.html
[11] NVIDIA Tesla GTX480 computing processor. [Online] http://www.nvidia.com/object/product_geforce_GTX480_us.html
[12] OpenCL programming guide. [Online] http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_OpenCL_ProgrammingGuide.pdf