ACC-SVM: Accelerating SVM on GPUs using OpenACC

Rengan Xu[†], Dounia Khaldi[‡], Abid M. Malik[†]and Barbara Chapman[‡†]
[†]Dept. of Computer Science, University of Houston, Houston, TX
Email: {rxu6, ammalik3}@uh.edu
[‡]Institute for Advanced Computational Science, Stony Brook University, Stony Brook, NY
Email: {dounia.khaldi, barbara.chapman}@stonybrook.edu

***Abstract*—GPUs have been successfully applied in scientific computing in the last decade. Many machine learning algorithms have also used GPUs to accelerate their computations. This includes the Support Vector Machine (SVM) which is a classical machine learning algorithm that has been successfully used in many applications such as text classification and image recognition. There have been many open-source CUDA SVM implementations. However, CUDA versions of SVM are not portable and difficult to maintain or redesign. Porting SVM to a directive-based portable model like OpenACC will make it possible to target multiple accelerators.**

In this paper, we use OpenACC programming model to parallelize SVM and produce ACC-SVM. Since the high-level programming model simplifies the programming by sacrificing performance, there is a performance gap between the OpenACC and the CUDA SVM versions. In order to improve the performance of ACC-SVM, we apply our auto-tuning framework to decrease the gap between the CUDA and the OpenACC performance results. The performance difference between the optimized ACC-SVM resulting from the auto-tuner and the CUDA SVM is 15.58% for the kernels code and 7.87% with respect to the whole application. For many applications this loss of performance is more than made up for by the convenience and portability of the high-level approach.

***Index Terms*—Accelerators; GPU; OpenACC; CUDA; Auto-tuning**

I. Introduction

Heterogeneous architectures that comprise CPU processors and computational accelerators such as GPUs have been increasingly adopted for scientific computing. The low-level programming models for GPUs such as CUDA and OpenCL offer users programming interfaces with execution models closely matching the GPU architecture. Effectively using these interfaces for creating highly optimized applications require programmers to thoroughly understand the underlying architecture, as well as significantly change the program structure and algorithms. This affects both productivity and performance. On the other hand, standardized directive-based models such as OpenACC [13] and OpenMP 4.x for accelerators [2] require developers to insert directives and runtime calls into the existing source code offloading portions of Fortran or C/C++ codes to be executed on accelerators.

Directives are high-level language constructs that programmers can use to provide useful hints to compilers to perform certain transformations and optimizations on the annotated code region. The use of directives can significantly improve programming productivity. Users can still achieve high performance of their program comparable to code written in CUDA or OpenCL, subjected to the requirements that a 'careful' choice of directives and compiler optimization strategies be made. One such scenario encountered quite commonly in a program is loop scheduling.

Many machine learning algorithms have also used GPUs to accelerate their computations. This includes the Support Vector Machine (SVM) which is a classical machine learning algorithm mainly used for classification and regression analysis. This technique has been successfully used in many applications such as text classification and image recognition. There have been many open-source CUDA SVM implementations including cuSVM [7], GPUMlib [12] and GPU-LibSVM [4]. CUDA versions of SVM are not portable and difficult to maintain or redesign. Porting SVM to a directive-based portable model like OpenACC will make it possible to target multiple accelerators. To the best of our knowledge, the only directive-based implementation of SVM is of Codreanu et al. [8] using their own toolkit called GPSME and their own directive syntax.

In this paper, we use OpenACC programming model to parallelize SVM. Since the high-level programming model simplifies the programming by sacrificing performance, there is a performance gap between an OpenACC program and the same program developed using CUDA. In order to improve the performance of SVM OpenACC program, we apply the loop scheduling optimization [17] to improve the performance.

The main contributions of this paper includes:

- We develop the SVM algorithm using the high-level directive-based programming model OpenACC.
- We apply loop scheduling optimization to improve the loop mapping from the loop nests to the GPU threads.
- We compare the performance of ACC-SVM and CUDA SVM and we get comparable results.

The organization of this paper is as follows. Section II gives an overview of GPU architecture, OpenACC programming model, and SVM. In Section III, we explain the methodology used for developing SVM using OpenACC and the application of loop scheduling optimization in order to improve performance. Performance results are discussed in Section IV. We conclude our work in Section V.

II. Background

A. GPU Architecture

GPU architectures differ significantly from that of traditional processors. Employing a Single Instruction Multiple Threads (SIMT) architecture, NVIDIA GPUs have hundreds of cores that can process thousands of software threads simultaneously. GPUs organize both hardware cores and software threads into two-level of parallelism. Hardware cores are organized into an array of Streaming Multiprocessors (SMs), each SM consisting of a number of cores named as Scalar Processors (SPs). Each SM has its own L1 cache which is not cache coherent, and all SMs share a unified L2 cache. Since GPU architecture is widely known and due to space constraints, we will not detail the structure of the memory hierarchy of GPUs.

B. OpenACC Programming Model

Directive-based high-level programming models for accelerators, such as OpenACC and OpenMP 4.x, provide extensions to Fortran, C and C++ to support accelerators. They have been designed to address the programmability challenges of GPUs. Using these programming models, programmers insert compiler directives into a program to annotate portions of code to be offloaded onto accelerators for execution. This approach relies heavily on the compiler to generate efficient code for thread mapping and data layout. It could be potentially challenging to extract optimal performance using such an approach rather than using other explicit programming models such as CUDA. However, directive-based models simplify programming on heterogeneous systems and save development time, while also preserving the original code structure assisting in code portability. OpenACC allows users to specify three levels of parallelism in a data parallel region: gang, worker and vector parallelism to map the loop nests to the multiple-level-thread hierarchy of GPUs. The user provides hints to map these three-levels of parallelism to GPU threads. However, the effectiveness of the mapping relies on the compiler and runtime implementation strategies. In this paper, we use an open-source, validated OpenACC compiler called OpenUH [15].

Usually it is easy to develop a correct OpenACC program, but it is difficult to achieve comparable performance to the one using low-level models such as CUDA. Therefore, different optimizations need to be applied. In a previous work [17], we developed an auto-tuning framework that automatically tunes the loop schedules in different kernels of OpenACC applications. The methodology of the framework is to retrieve the optimal loop schedule, threads topology, and threads size in different dimensions based on the modeled memory access cost. The framework also takes the data locality into account by modeling the cache hit rate for all loop schedules.

C. Overview of Support Vector Machine (SVM)

Support Vector Machine (SVM) [16] is a classical machine learning algorithm to perform classification and regression analysis. In this section, we present three main SVM algorithms: (1) the classical linear SVM that separates the dataset by a separating hyperplane, (2) non-linear SVM that can be used in non-linear classifications, and (3) multi-class SVM that handles multi-class classifications.

1) Linear SVM: Given l samples $(x_1, y_1), ..., (x_l, y_l)$ with $x_i \in R^n$, n is the dimension size of each data sample. Consider Figure 1, in which x's denote positive training samples, o's denote negative training samples. The positive samples are labeled as "+" and the negative samples are labeled as "-". The goal of SVM is to form a separating

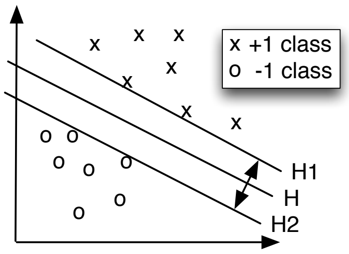

Fig. 1: SVM Classification

hyperplane H that separates the positive and negative samples as far as possible, which means the margin between the lines $H1$ and $H2$ is maximized. The output or the class label of each training sample x_i can be computed by

$$O(x_i) = w^T x_i + b \tag{1}$$

where w is the normal vector to the separating hyperplane and $(x_i,\ y_i)$ is a training sample. b is a intercept term. For linearly separable datasets, the following inequalities are satisfied:

$$w^T x_i + b = \begin{cases} \geq 1 & \text{if } y_i = 1 \\ \leq -1 & \text{if } y_i = -1 \end{cases} \tag{2}$$

This is equivalent to $y_i(w^T x_i + b) \geq 1$. The nearest negative samples are in line $H2$ which satisfy $O(x_i) = -1$, and the nearest positive samples are in line $H1$ which satisfy $O(x_i) = 1$. The lines $H1$ and $H2$ are called gutter lines. The margin is defined as twice the distance between the hyperplane and the gutter line

Maximizing the margin is an optimization problem with a convex quadratic objective and only linear constraints. The method of Lagrange multiplier can be used to solve this optimization problem. This optimization problem is a Quadratic Programming (QP) problem where the objective only depends on Lagrangian multiplier Once this problem is solved , then we can easily compute the normal vector w and the intercept b. Once the model using the training samples is built, we can predict a new sample x. Then, we can assign a class to x: positive or negative based on the y value.

2) Non-linear SVM: In some cases, the samples are not linearly separable. SVM, however, can still linearly separate them by mapping the original input space to a higher dimensional dot-product feature space. Such mapping is done by a non-linear kernel function $K(x_i, x_j)$. Some commonly used kernel functions are Linear, Polynomial, Radial basis, and

Sigmoid explained in [9]. Although mapping data to a high dimensional feature space usually increases the chance that the data is separable, we cannot guarantee that it always succeeds. And in some cases, finding a separating hyperplane might be susceptible to outliers. To make the algorithm less sensitive to outliers, $l1$ regularization is applied to the optimization problem using a new parameter C that determines the tradeoff between the maximization of the margin and the minimization of the errors.

3) Multi-class SVM: The classical SVM was originally designed for binary classification. In multi-class classification, N independent binary classifiers f^i can be combined together to solve the multi-class problem. Assume for the l samples, $y_i \in Y$, $\forall i$ and $Y = 1, ..., K$ which means every sample belongs to one of the K classes. An output code matrix R with size $K \times N$ is constructed, where K is the number of classes, N is the number of tasks, and $R_{ij} \in \{-1, 1\}$. Then each sample is trained by all of the built classifiers. There are several ways to construct the total number of binary classifiers [9].

In this paper, we develop the OpenACC version of SVM. Sequential Minimal Optimization (SMO) [14] is a popular algorithm used to solve the SVM QP problem by iteratively solving a series of smaller QP subproblems with only two unknown variables that are solvable analytically. Cao et al. [6] proposed the parallel SMO algorithm called PSMO to parallelize SVM by distributing the dataset into multiple computing nodes. Herrero-Lopez et al. [9] improved this algorithm and developed P2SMO algorithm for multi-class classification. Our OpenACC implementation is based on the CUDA version of P2SMO implementation.

III. ACC-SVM: OpenACC Implementation of SVM

We develop the SVM OpenACC version from the CUDA P2SMO algorithm for multi-class classification. The advantage of this porting is to make the SVM code portable on other accelerators besides GPU, and easy to maintain. To convert a CUDA code into OpenACC code, we need to work on two main parts: data and computation. The data part includes memory allocation, memory deallocation, data movement and data synchronization. The generic methodology to convert CUDA related APIs to OpenACC directives is the following.

1) If a data is allocated by `cudaMalloc()` and freed by `cudaFree()`, then the host variable name of this data will be put in `acc data` directive if the data lifetime can be within a structured block. In OpenACC, the device variable name for that data is no longer needed since the runtime has a global hash table to maintain all data that have both the host copy and device copy [15]. If the data memory allocation and deallocation are in different functions, then they will be put in `acc enter data` and `acc exit data` directives.
2) If a data has both host and device copies and uses `cudaMemcpy()` to be moved from host to device right after device memory allocation, then it is put into OpenACC `copyin` data clause.
3) If a data has both host and device copies and uses `cudaMemcpy()` to be moved from device to host right before device memory deallocation, then it is put into OpenACC `copyout` data clause.
4) If a data has both host and device copies and uses `cudaMemcpy()` to be moved from host to device right after device memory, and moved from device to host right before device memory free, then it is put into OpenACC `copy` data clause.
5) If a data has both host and device copies but does not use any `cudaMemcpy()` to be moved from host to device right after device memory allocation, or moved from device to host right before device memory free operation, then it is put into OpenACC `create` data clause.
6) If a data does not have host copy which means its device memory is allocated with `cudaMalloc()` and it is a temporary data used on device, then we use OpenACC `acc_malloc()` to replace the CUDA API call.
7) If there is any data movement within the lifetime of a data, then we use `acc update host` or `acc update device` to replace the `cudaMemcpy()` call. Whether `host` or `device` is used depends on the data movement direction.

After handling the data part, the CUDA kernels are replaced by the sequential loop nests code. However, it is error prone to replace all CUDA kernels at one time. To guarantee the correctness of the conversion, one can firstly use OpenACC `host_data` directive to call these kernels. Then these kernels are replaced to sequential code one by one. The conversion from a CUDA kernel to OpenACC kernel is the reverse operation of transforming a loop nest to CUDA kernel. Different loop transformations may be applied for different kernels.

In this work, we followed the methodology presented above to develop an OpenACC version of SVM. Then, we applied our auto-tuning framework to improve performance.

IV. Performance Evaluation

The experimental platform is Intel Xeon processor E5520 with frequency 2.27GHz and 32GB main memory and an Nvidia Quadro K6000 GPU card which uses K40 architecture. The CUDA SVM [1] is compiled by nvcc compiler with flag "-O3". The OpenACC SVM is compiled by OpenUH compiler. The dataset we use come from different sources. *adult* is from UCI [3] dataset, *letter* and *shuttle* are from Statlog dataset [5]; both *mnist* [11] and *usps* [10] are hand-written datasets used for text recognition. The characteristics of each dataset are presented in Table I where C is the regularization parameter and γ is the stopping parameter of the SMO algorithm.

TABLE I: Characteristics of the experiment dataset

Dataset	Training Samples	Features	Classes	C	γ
adult	32561	123	2	100	0.001
mnist	30000	780	10	10	0.125
usps	7291	256	10	100	0.001
letter	15000	16	26	100	0.001
shuttle	43500	9	7	100	0.001

Figure 2 shows the kernel performance of ACC-SVM against CUDA-SVM. As we can see, there is significant performance improvement after applying loop scheduling optimization enabled by our auto-tuning framework. The average kernel performance speedup for all datasets is 1.92x. The kernel performance gap between the two versions is 15.58%.

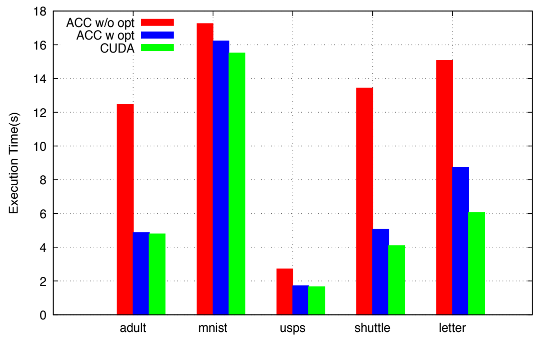

Fig. 2: Kernel performance of ACC-SVM against CUDA-SVM

Figure 3 shows the total execution time of the application of ACC-SVM against CUDA-SVM; this includes the kernels time plus the data movement and the host code time. The average speedup after the loop scheduling optimization for the whole application is 1.23x. The application performance gap between ACC-SVM with optimization and CUDA-SVM is 7.87%.

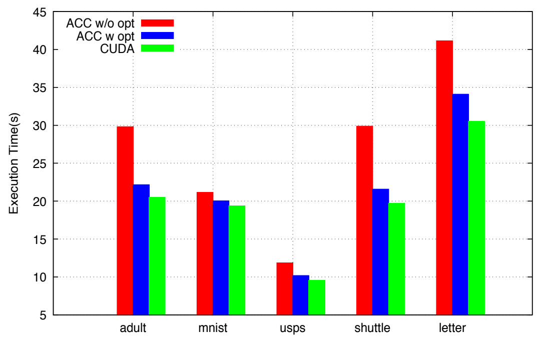

Fig. 3: Application performance of ACC-SVM against CUDA-SVM

V. Conclusion

This paper presented the directive-based implementation of SVM using OpenACC. We start from the CUDA SVM version and show the methodology of transformations needed when using OpenACC. Porting SVM to OpenACC makes it possible to target multiple accelerators and renders the code easy to maintain and redesign. Our approach converts all CUDA kernels in the application incrementally. Moreover, we applied our auto-tuning framework that optimizes loop nest scheduling. This improves significantly the kernel performance as well as the whole application performance. The performance difference between the optimized ACC-SVM resulting from the auto-tuner and the CUDA SVM is 15.58% for the kernels code and 7.87% with respect to the whole application. For many applications this loss of performance is more than made up for by the convenience and portability of the high-level approach.

Acknowledgment

This research work is supported by the US Office of the Assistant Secretary of Defense for Research and Engineering (OASD(R&E)) under agreement number FA8750-15-2-0119. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US NSF, or the Office of the Assistant Secretary of Defense for Research and Engineering (OASD(R&E)) or the U.S. Government.

References

[1] CUDA SVM. https://code.google.com/p/multisvm/, 2016.
[2] OpenMP ARB. OpenMP Application Program Interface, Version 4.5. http://www.openmp.org/mp-documents/openmp-4.5.pdf.
[3] Arthur Asuncion and David Newman. UCI Machine Learning Repository, 2007.
[4] Andreas Athanasopoulos, Anastasios Dimou, Vasileios Mezaris, and Ioannis Kompatsiaris. GPU Acceleration for Support Vector Machines. In *WIAMIS 2011: 12th International Workshop on Image Analysis for Multimedia Interactive Services, Delft, The Netherlands*, 2011.
[5] P Brazdil and J Gama. Statlog Datasets. *Inst. for Social Research at York Univ., http://www. liacc. up. pt/ML/statlog/datasets. html*, 1999.
[6] Li Juan Cao, S Sathiya Keerthi, Chong-Jin Ong, Jian Qiu Zhang, and Henry P Lee. Parallel Sequential Minimal Optimization for the Training of Support Vector Machines. *IEEE Transactions on Neural Networks*, 17(4):1039–1049, 2006.
[7] AUSTIN Carpenter. cuSVM: A CUDA Implementation of Support Vector Classification and Regression. *patternsonscreen. net/cuSVMDesc. pdf*, 2009.
[8] Valeriu Codreanu, Bob Dröge, David Williams, Burhan Yasar, Po Yang, Baoquan Liu, Feng Dong, Olarik Surinta, Lambert RB Schomaker, Jos BTM Roerdink, et al. Evaluating Automatically Parallelized Versions of the Support Vector Machine. *Concurrency and Computation: Practice and Experience*, 2014.
[9] Sergio Herrero-Lopez, John R Williams, and Abel Sanchez. Parallel Multiclass Classification using SVMs on GPUs. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 2–11. ACM, 2010.
[10] Jonathan J Hull. A Database for Handwritten Text Recognition Research. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(5):550–554, 1994.
[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
[12] Noel Lopes and Bernardete Ribeiro. GPUMLib: An Efficient Open-source GPU Machine Learning Library. *International Journal of Computer Information Systems and Industrial Management Applications*, 3:355–362, 2011.
[13] OpenACC 2.5 Specification. http://www.openacc.org/sites/default/files/OpenACC_2pt5.pdf, 2015.
[14] John C Platt. 12 Fast Training of Support Vector Machines using Sequential Minimal Optimization. *Advances in kernel methods*, pages 185–208, 1999.
[15] Xiaonan Tian, Rengan Xu, Yonghong Yan, Zhifeng Yun, Sunita Chandrasekaran, and Barbara Chapman. Compiling A High-Level Directive-based Programming Model for GPGPUs. In *Intl. workshop on LCPC 2013*, pages 105–120. Springer International Publishing, 2014.
[16] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 2013.
[17] Rengan Xu, Sunita Chandrasekaran, Xiaonan Tian, and Barbara Chapman. An Analytical Model-based Auto-tuning Framework for Locality-aware Loop Scheduling. In *Proceedings of 2016 International Supercomputing Conference*, 2016.