

# SCALING KERNEL MACHINE LEARNING ALGORITHM VIA THE USE OF GPUS

BALAJI VASAN SRINIVASAN, RAMANI DURAISWAMI

Algorithms based on kernel methods play a central role in modern machine learning and nonparametric statistics. At the core of these algorithms are a number of linear algebra operations on matrices of kernel functions which take as arguments the training and testing data. These algorithms range from the simple matrix-vector product, to more complex matrix decompositions, and iterative formulations of these. Often these algorithms scale quadratically or cubically both in memory and operational complexity, and as data sizes increase, kernel methods scale poorly. In this paper, we address this lack of scalability by using parallelized kernel algorithms on a GPU.

The space complexity can be addressed by computing the kernel function on-the-fly thus reducing it from  $O(NM)$  to  $O(N + M)$ . The chief bottlenecks in the computations in these algorithms can be classified into three categories, (1) Weighted summation of kernel functions, (2) Solution of linear system involving kernel matrix, and (3) Decomposition of kernel matrices (LU, QR, etc). We particularly concentrate on categories (1) and (2); GPU based accelerations for matrix decompositions like QR, LU has already been dealt in [2].

Given the training data  $x_i, i = 1, \dots, N$  and the test data  $y_j, j = 1, \dots, M$ , a weighted summation of the kernel function evaluated at the training and test points is given by,  $f(y_j) = \sum_{i=1}^N q_i k(x_i, y_j)$ , where  $k(x_i, y_j)$  is the kernel function. Some popular kernel functions are the Gaussian, Matern, Periodic and Epanichnikov. Such kernel summations and its derivatives commonly occur in many machine learning algorithms like kernel density estimation, support vector machine, learning a ranking function [6], Gaussian process regression [4].

We shall call the  $x$ 's as source points,  $y$ 's as evaluation points. If each thread is assigned to evaluate the effects of a particular source on all evaluation points, it would have to update the value at each evaluation point in the global memory, thus requiring a number of global memory writes. In our algorithm, we therefore assign each thread to evaluate the kernel sum on an evaluation point. Suppose there are  $N$  source points, each thread would be requires to read  $N$  source points from the global memory. We reduce the total accesses to global memory, by transferring the source points to the shared memory. The shared memory is not large enough to house the entire data. So it is required to divide the data into chunks and load them according to the capacity of the shared memory (the number of source points that can be loaded to the shared memory is limited by its size and dimension). The size of each chunk is set to be equal to the number of threads in the block, and each thread transfers one source element from the global memory to the shared memory, thus ensuring coalesced memory reads. The weights corresponding to each source is also loaded to the shared memory in the same way. Once the source data is available in the shared memory, all the threads update the terms of the kernel sums involving the source points in the shared memory. If  $d$  is the dimension of the source and evaluation points, then the proposed algorithm would require  $d + 1$  registers per thread and  $d + 1$  shared memory location per thread. We achieved speedups in the range of 450X to 550X for the various kernels mentioned above. To accelerate algorithms in category 2, we couple the GPU based kernel summation algorithm developed here with popular iterative approaches like conjugate gradient, Levenberg-Marquardt, etc.

In all our experiments the host processor is an Intel Core2 Duo, 2.4GHz and 2GB RAM. The GPU used is the GT200 graphics processor. GT200 is a 192 core processor and has 1GB of global memory, 16384 registers per thread block, 64kB constant memory and 16kB shared memory. We applied the proposed algorithm to accelerate several kernel based learning approaches and achieved considerable speedups in each case. In kernel density estimation [1], to estimate the densities at 10,000 datapoints based on 10,000 samples, while the CPU based implementation took  $\sim 16$ s, our GPU algorithm took only  $\sim 13$ ms. For estimating the optimal bandwidth of the Gaussian kernel [3], the CPU version took  $\sim 930$ s whereas the GPU version took  $\sim 2$ s for the same data as before. In Gaussian process regression [4], the algorithm complexity is  $O(N^3)$ . For regressing the 8-dimensional data *Kin8nm*, GPU based prediction took only  $\sim 2$ s while the direct version took hours for the same prediction. Applying the GPU-based approach to learn a ranking function like in [6] for the *California Housing* dataset, the linear algorithm in [6] took  $\sim 45$ s, whereas our approach took only  $\sim 2$ s inspite of being quadratic. For mean shift clustering [5] on a  $256 \times 256$  image, CPU version took  $\sim 1300$ s for clustering, whereas our GPU approach took  $\sim 4$ s. We further compare the GPU based Gaussian kernel summation with a linear algorithm FIGTREE [7], and show that for a 3-dimensional data, the proposed approach outperforms the linear algorithm upto a data size of 100,000.

## REFERENCES

- [1] R. Duda, P Hart, and D Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.
- [2] V Volkov and J Demmel. Benchmarking GPUs to tune dense linear algebra. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1–11. IEEE Press, 2008.
- [3] S Sheather and M Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(3):683–690, 1991.
- [4] C Rasmussen and C Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, December 2005.
- [5] D Comaniciu and P Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.
- [6] V Raykar, R Duraiswami, and B Krishnapuram. A fast algorithm for learning a ranking function from large-scale data sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1158–1170, 2008.
- [7] V Morariu, B Srinivasan, V Raykar, R Duraiswami, and L Davis. Automatic online tuning for fast Gaussian summation. In D Koller, D Schuurmans, Y Bengio, and L Bottou, editors, *NIPS*. MIT Press, 2008.