# Machine Learning Algorithms on GPU

Tugcan Olgun

***Abstract*—Machine Learning is basically a computer learns how to do things without explicitly told how to do. It is not a new phenomena but recently it became vastly popular. As it requires data to be trained with and as data collection has become a new trend for the a few past decades, the amount of data that is used to train has become larger in size. This lengthened the training time immensely. Machine learning algorithms traditionally have been executed using only CPU. CPUs, as great devices as they are, cannot race with the growth of data. Enter the GPU computation era where machine learning algorithms are divided into pieces where the code blocks where can be parallelized executed within GPU and the blocks where cannot, still uses CPUs. CUDA, a framework developed by NVIDIA handles this execution and task separation. Matrix computation gains a huge benefit from using GPU blocks as the time it takes to calculate a scalar multiplication is only as long as it takes to calculate a single block of the resulting matrix.**

***Index Terms*—machine learning, GPU, CUDA, cublas**

## I. Introduction

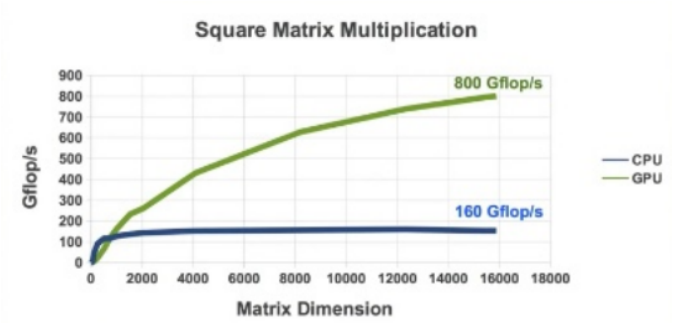
MACHINE learning is a field of computer science that basically getting computers to learn without explicitly programming. Even thought it has become very popular in the past decade or so, it is not a new phenomena. It was known as pattern recognition and computational learning theory. It has now evolved into a very popular field with the usage of Artificial Neural Networks (ANN) and Deep Learning and made it possible very useful tools such as self-driving cars, practical speech recognition, effective web search.

Machine Learning requires a training process. First, the problem is defined and the result that is required to be achieved is decided on. Secondly, depending on the problem, data is shaped into a form that enables the algorithm to learn from it. Then the training starts. After this, modifications are made so the result is improved and finally the results are presented.

Traning phase of this process takes time. Trainings traditionally has been done using single processor environments where due to the limitations, the way the code is being written and the data being set was more important than the speed of the processor. These bottlenecks are still stand as an issue. However, with the increasing data availability and the areas that the machine learning are being used lately, training time takes immense amount of time. Because the CPUs are not meant to be used for such cases. They are fast but not fast enough.

This paper, based on machine learning algorithms on gpu for the class SWE 578, Bogazici University is sent to Ali Taylan Cemgil (fatih.alagoz@boun.edu.tr) on 14 th of May 2018.

Olgun, T. studies Software Engineering at Bogazici University. He also works as a back-end developer for Ambeent Wireless A.S. at Istanbul Technical University Ari Teknokent building. (e-mail: tugcan.olgun@boun.edu.tr)




A solution to use the high amount of cores found on high end graphics processing units (GPUs) via utilizing general purpose computing on graphics processing units (GPGPU). This is done via parallelizing the processes that can be parallelized. A comparison graph of square matrix multiplication with various matrix dimensions on GPU and CPU can be seen on Figure I.

Unfortunately, not every operation can take advantage of GPGPU because of parallelism issues. Parallelism itself is an open research problem. It is caused by the equation NC=?P. According to research by Berkeley, n-body problems maybe relatively difficult to efficiently parallelize as n increases because there are $n^2$ interactions between all the n points. However, as machine learning trainings are carried with Big Data, the necessity of finding a solution would worth the effort given how time consuming the process of these data. Also, this extensive trouble would without a doubt help the processing of relatively small data as well. As a number of machine learning tasks can be time consuming due to their computational requirements, parallelizing the blocks where sequential execution not necessary may benefit the execution time all together.

Flops are floating point operations per second. Below can be found a simple comparison of speed between a sample CPU and GPU.

| Name | Cores | Gigaflops | Billion transistors |
|---|---|---|---|
| GeForce TITAN | 2688 | 4500 | 7.1 |
| Core i7 3960X | 6 | 316 | 2.3 |

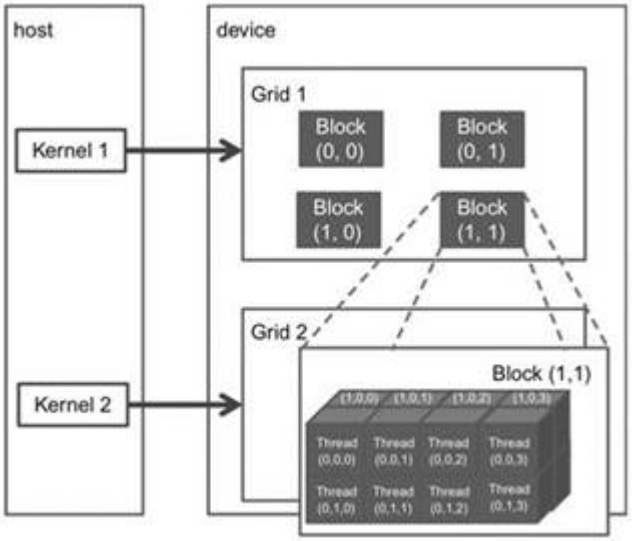
Several regression, classification and clustering techniques especially linear regression can benefit from parallelizing algorithms which are shared among several machine learning paradigms such as matrix multiplication processes. But unfortunately as parallelism works wonders with the execution time, the chain is only as strong as its weakest link which are the tasks that cannot benefit from these improvements.

## II. CUDA

CUDA, formerly known as Compute Unified Device Architecture is an implementation of GPGPU. CUDA has hosts and devices. Hosts are the traditional CPUs and devices are the GPUs which consists large numbers of arithmetic units. It supports traditional programming languages and it is basically a bridge between CPUs and GPUs by enabling parallel execution methods which are called kernels.

A large number of CUDA threads are executed when kernel is launched. A collection of large number of threads in this manner are called a block of threads. Also the collection of blocks are called grids. Threads are arranged in 3-dimensional layouts within blocks, which are, in turn, arranged in 3-dimensional layouts within grids [Figure II].

Hosts and devices are independent of their own memory space to one another. A singular global memory space is shared by CUDA. Firstly, the data from host is copied to device memory by enabling kernels and spawning numerous device threads. By CUDA's extensions the results are copied back the reverse direction.




It is however very important to manage memory carefully because too many allocation of blocks to kernels may cause some threads to be idle where too few would greatly reduce computation. As device memory allocation can be managed by users, with the help of CUDA's semi-automatic memory allocation via the calculation the size of the data depending on the type such as matrix and array and its approximate memory requirement, this is no longer a nightmare.

Model parallelism: split up a single model

Data parallelism: split up data to train a single model

For an example how the matrix multiplication which is a part of linear regression is being done via CUDA, with two matrices to multiply. The matrix P will be the result matrix of M and N. In order to start with this multiplication, allocation for these matrices in global memory is necessary. This example assumes that all matrices fit into a block. Each thread of the block will be used to calculate an element of P. The parallelism in the calculation is as follows: To calculate an element of P, a thread takes a row of M and a column of N and do the dot multiplication. Because every thread works by itself and not dependent on the others, the total execution time for whole P matrix to be calculated is the time that it takes an individual thread's. Once the computation is finished, the P matrix copied back so it can be used with other methods and usually after copying is finished, the allocated memory would be deallocated.

Discovering unlimited possibilities with parallel processing in machine learning may only be feasible once there is a merge of parallelizable portions of code with appropriate amount of data, solid knowledge of algorithm implementation, and ambition. Simplifying the demanding process of recognizing the parallelizable portion of code might be supported by the application of the following methods: matrix multiplication, distance calculation, and k-fold cross-validation. The first one involving the use of linear regression algorithms with reduced time benefits. The second one covering continuous and simultaneous calculation over various algorithms, and the third one analyzing randomly divided k-subsets. K-fold cross-validation proves to be more detailed, but also lengthy when performed one after another, as it may cover a large amount of data with each subset to be tested once and be in a training k-1 times, according to previously and individually selected criterion on the size and the number of tests to be run. This technique may also be used with linear regression matrix multiplication to achieve even more accurate results for model building.

## III. Conclusion

Machine learning algorithms takes great advantages for using modern GPUs where they provide greater matrix process capabilities. GPUs are meant to be created drawing on 3D screen. This means that any given moment, every pixel's vector, surface and point is calculated in every frame. These calculations are matrix operations. Machine learning algorithms do have matrix operations and can be greatly improved in speed if to use the benefit of the GPUs. The parallelization of code blocks in the algorithms allows the running time of the code to be as less as a section of the computation.

## References

[1] Asanovic, K. (2018). The Landscape of Parallel Computing Research: A View from Berkeley — EECS at UC Berkeley. [online] Www2.eecs.berkeley.edu. Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[2] Dean, J. and Corrado, R. (2018). Large Scale Distributed Deep Networks. Google Inc., Mountain View, CA. https://static.googleusercontent.com/media/research.google.com/en//archive/large_deep_networks_nips2012.pdf

[3] He, K. (2015). Deep Residual Learning. MSRA @ ILSVRC COCO, [online] 2015. Available at: http://kaiminghe.com/ilsvrc15/ilsvrc2015_deep_residual_learning_kaiminghe.pdf