Accelerating Machine Learning Algorithms in Python

Patrick Reilly, Leiming Yu and David Kaeli

Department of Electrical and Computer Engineering
Northeastern University, Boston, MA
reilly.pa@husky.neu.edu, {ylm,kaeli}@ece.neu.edu

Abstract

Developing machine learning products in Python has been facilitated by the availability of powerful machine learning libraries, such as scikit-learn. With more and more data to process, prototyping tools in Python becomes time-consuming. Graphic Processing Units (GPUs), consisting thousands of light-weight cores, become an ideal library accelerator. In this paper, we report on a study that optimizes a set of machine learning algorithms on a GPU. We report on the performance of two popular GPU integration tools developed in Python, Cython and PyCUDA. Utilizing the parallel performance advantages of a GPU, we can achieve speedups of 20x-200x over multi-threaded scikit-learn CPU-based implementations.

1 Introduction

Prototyping machine learning tools in Python is flexible and can be very efficient. Most machine learning libraries are developed to run on a CPU. When working with large data sets, given the high computational demands of many machine learning algorithms, application throughput can be limited when running on traditional (i.e., CPU-based) platforms. Designed as massively parallel architectures, GPUs have become popular when running many machine learning applications, including deep learning and computer vision. Two popular Python-based frameworks that support high-performance computing on GPUs are Cython [2] and PyCUDA [5]. Cython is a static compiler that wraps C/C++/GPU code in Python. PyCUDA allows users to directly access NVIDIA’s CUDA driver API, compile the kernel in a just-in-time fashion, and move data freely between Python data objects and GPU memory. In this paper, we have optimized a set of machine learning algorithms on a GPU, developing corresponding implementations using Cython and PyCUDA. Equipped with our GPU-accelerated machine learning libraries, we are able to achieve significant speedups as compared to the baseline scikit-learn version.

2 GPU Computing

Modern GPUs consist of massively parallel programmable processing cores [9]. These cores are grouped into multiple streaming multiprocessors (SMM) on the device, as shown in Figure 1. On the NVIDIA Maxwell architecture there are 128 CUDA cores, with 4 warp schedulers to dispatch instructions. Data residing in global memory is shared across the SMMs. The L1 cache and texture are unified. The shared memory is used to cache data that is frequently read and written over the span of a block execution. For the read-only data, the constant memory can be utilized for fast access.

3 Performance Optimizations

We optimize three machine learning algorithms on the GPU. The algorithms include: k-nearest neighbor (KNN), logistic regression (LR) and Kmeans. For KNN, the NumPy argpartition function is used for sorting. For LR, the cuBLAS is applied for the matrix operations. For Kmeans, the block histogram for each cluster is atomically added to the global memory.

4 Performance Evaluation

Our performance evaluation testbed includes an Intel Core i7-4790K CPU and a NVIDIA 950 GTX GPU. The Anaconda 4.2 framework is installed for all the required python libraries, including the scikit-learn 17.1, scikit-cuda 5.1, Cython 25.2, and PyCUDA 2016.1.2. The CUDA Toolkit 7.0 is used. Single precision data is used for comparison. The performance efficiency between Cython and PyCUDA is shown in Figure 3. For Cython, the Unified Memory feature introduces more overheads. For PyCUDA, we load the precompiled binaries to
minimize the compilation overhead. Due to the additional memory management during the initialization and running steps, Cython achieves comparable performance than PyCUDA. We use PyCUDA instead of Cython for our work.

As shown in Figure 2, speedups of 27.9x-106.9x are achieved for KNN. For LR, we achieve a speedup of 46.4x-197.9x. For Kmeans, speedups range from 41.7x to 129.8x.

5 Related Work
Accelerating machine learning algorithms on GPUs has been extensively studied in previous work[7, 8, 4]. To accelerate Python libraries, Larsen extended the NumPy operations to the GPU using CUDArray [6]. GPU memory with Python bindings are built in Theano for deep learning [3]. PyCUDA and PyOpenCL has been developed to leverage the computing power of CPUs and GPUs [5]. Based on LLVM, Numba supports CUDA GPU and HSA APu programming that allow native kernels to access the NumPy arrays directly [1]. Speedups ranging between 10x and 100x are often reported when moving from a multi-threaded CPU to a GPU.

6 Conclusions
GPUs are becoming an attractive accelerator for machine learning algorithms, especially as data set sizes continue to grow. Equipped with optimized GPU kernels and PyCUDA, high-level Python code can enjoy the same performance benefits as the native GPU implementations. In this work, we focus on optimizing machine learning algorithms on a GPU and developing the convenient drop-in GPU-accelerated machine learning libraries in Python.

References