AYASDI

Topological Data Analysis Acceleration on the GPU

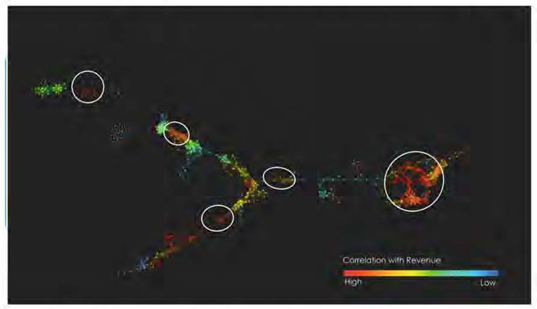
Topology provides a mathematical framework for applying a complete range of statistical, geometric, and machine learning methods, revealing insights from the geometry of your data. Ayasdi utilizes topological data analysis (TDA) in its advanced analytics software to simplify the analysis of complex, multivariate datasets. We present our findings on how we can leverage the GPU to accelerate key operations in TDA.

Topological Data Analysis (TDA)

Topology is a mathematical discipline that studies shape. Topological data analysis refers to the adaptation of this discipline to analyzing highly complex data. It draws on the philosophy that all data has an underlying shape and that shape has meaning.

Ayasdi's TDA Framework

Ayasdi's approach to TDA draws on a broad range of machine learning, statistical and geometric algorithms. The analysis creates a compressed representation the data in the form of networks, from which the user or a downstream machine learning algorithm can extract insight.



Ayasdi's network construction employs several compute-intensive steps, including distance computation and data transformation.

Focus for GPU Acceleration

For our current work, we focus on distance computation and data transformation.

Point-to-point distances are used extensively in various aspects of Ayasdi's TDA framework. The parallel nature of distance computation makes it a good candidate for acceleration.

Ayasdi's framework employs an extensive set of algorithms to transform data, many of which are candidates for acceleration. In this poster, we describe work on one of these algorithms.

System configuration:
CPU: Core i7-4790k, 4C/8T, 4.0 GHz
GPU: Nvidia Tesla K40c, 15 MP, 745 MHz

[1]We based our work primarily on V. Volkov's research. http://www.cs.berkeley.edu/~volkov/
[2]Heaton Research. http://www.heatonresearch.com/wiki/CUDA
[3]TSNE. http://lvdmaaten.github.io/tsne/

Questions? ryan.hsu@ayasdi.com
Learn more about TDA and Ayasdi: http://www.ayasdi.com/resources/

Computing Point-to-Point Distances

Distance computation between two data points has the general form of comparing all features and summing the comparisons. For example, Euclidean distance (L2) between point p_i and point p_j has the form

$$d_{ij} = \sqrt{\left(p_{j,1} - p_{i,1}\right)^2 + \left(p_{j,2} - p_{i,2}\right)^2 + \cdots + \left(p_{j,n} - p_{i,n}\right)^2}$$

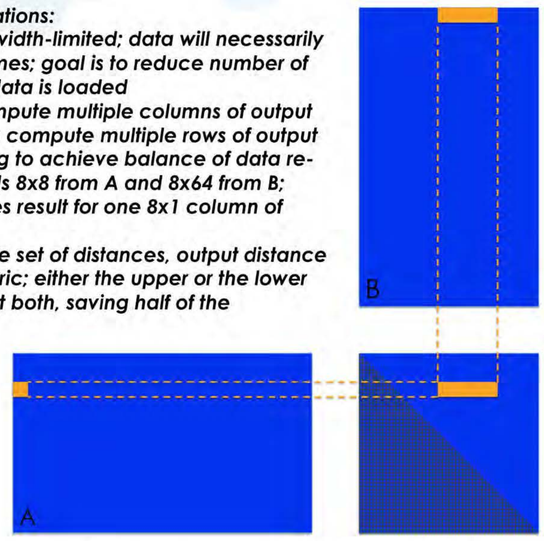
Point-to-point distances are commonly stored in a structure known as the distance matrix, where the entry at row *i* and column *j* is the distance between p_i and p_j.

$$D = \left(d_{ij}\right)$$

Given the distance computation pattern and the output storage format, one realizes that point-to-point computation resembles matrix-matrix multiplication. As such, it is possible to leverage the extensive research[1] that exists in this area.

Observations & Optimizations:

- ***Computation is bandwidth-limited; data will necessarily be loaded multiple times; goal is to reduce number of times each piece of data is loaded***
- ***One row in A can compute multiple columns of output***
- ***Each column of B can compute multiple rows of output***
- ***Use thread-coarsening to achieve balance of data re-use; block of 8x8 loads 8x8 from A and 8x64 from B; each thread computes result for one 8x1 column of result***
- ***If computing complete set of distances, output distance matrix will be symmetric; either the upper or the lower triangle is needed, not both, saving half of the computations***



The CPU-based version of distance computation is fully AVX-accelerated and threaded (8). The GPU time includes data transfer. The number of points is kept constant (5120) while number of features varies. In the case where number of feature is small, data transfer overhead overwhelms compute uplift, but all other cases see benefit.

	L1 X up	L2 X up	orrelati X up	Angle X up
5120 x 64	-1.75	-1.63	**1.69**	**1.52**
5120 x 512	**2.65**	**2.34**	**6.80**	**5.21**
5120 x 1024	**4.19**	**4.10**	**9.36**	**7.68**
5120 x 2048	**9.58**	**5.53**	**11.59**	**8.82**
5120 x 4096	**6.69**	**6.66**	**12.75**	**9.81**
5120 x 8192	**7.64**	**7.44**	**13.92**	**10.46**
5120 x 16384	**7.92**	**7.90**	**15.01**	**11.17**

Table 1. Full distance computation time GPU vs. CPU. GPU timing includes data transfer.

Data Transformation Acceleration Example

Lens transforms data in high-dimensional space to lower-dimensional space. One such lens calculates Gaussian density. This lens provides, for one point, a measure of how many other data points are nearby.

In this simple example, ● would have a higher Gaussian Density value than would ● .

For a given distance matrix (see, previous section), perform the following:

1. Compute the variance for the entire data set
2. For each point, compute and accumulate Gaussian distribution value for distance between this point and every other point

# Points	X up
1000	**5.92**
2000	**9.79**
4000	**8.70**
8000	**8.78**
16000	**8.76**

Figure 2. Gaussian Density computation GPU vs CPU. GPU time includes data transfer.

Future Work

In this work, we presented just one example where GPU can be leveraged for accelerating lens computation. We will continue this work in two ways:

1. For existing algorithms, we will investigate additional candidates that can use GPU acceleration. We will look to leverage existing CUDA libraries, such as CUBLAS, when possible.
2. We're also investigating dimensionality reduction algorithms that may have been once considered prohibitively expensive without acceleration. These include interesting work in neural networks (along with packages such as Encog[2]) and t-SNE[3].

Ayasdi's TDA algorithm and software analyzes terabytes of data by decomposing the complete set into subsets and processing these subcomponents in divide-and-conquer manner. However, even these smaller subsets may exceed the memory capacity of even the highest-end Nvidia Tesla GPUs. To overcome these limitations, we plan on breaking these subsets into even smaller components and leveraging CUDA streams to form pipelines to overlap computations and data transfer.