PCA-Augmented Vector Quantization on the GPU

Jake Lever, Won-Ki Jeong, Hanspeter Pfister
School of Engineering and Applied Sciences, Harvard University

Introduction

Many applications including several image formats use a finite-sized codebook to compress data to a set of indices. This can result in high compression rates and extremely fast decompression speeds. However increasing the compression speed remains an important challenge. This research examines how Principle Component Analysis (PCA) implemented on the GPU can be used to improve the speed and error of the Vector Quantization algorithm.

Using an iterative clustering algorithm augmented with PCA, the algorithm finds an appropriate codebook for the data. Previous research has examined how the GPU can optimize Lloyd's Algorithm [3]. A PCA splitting method to improve the final error of Lloyd's Algorithm is the main focus of this research. Large scale image and geometry viewers, such as [2], can be a direct application by allowing for fast on-the-fly compression of giga-pixel images.

Lloyd's Algorithm

Lloyd's Algorithm (shown in **Figure 1**) aims to reduce the sum of squared distances from each data point to its assigned cluster. Each data point is assigned to its nearest cluster. The CUDA implementation breaks down into three main kernels:

- Find the nearest cluster point for all data points
- Collect and total up the data points assigned to each cluster point
- Create the new cluster points from the totals

For small cluster counts, a simple linear search works very quickly to find the closest cluster points. This can be optimized through use of GPU constant memory. Current research is looking at how KD-trees can be used on the GPU to speed up this process.

Due to limited shared memory, each dimension is executed individually so that a thread-block can total up the values for each cluster-point using a simple reduction. The collection kernel for large number of cluster points, splits the clusters between a larger number of threads, so that thread-blocks ignore cluster points that are out of their range.

PCA Splitting

The initial clusters used in Lloyd's Algorithm greatly affects the final outcome. In fact with poor initial vectors, the algorithm can settle to a local minimum that has a significantly larger error compared to the global minimum. A binary partitioning scheme with a PCA-based split plane is used to create the initial cluster points for the Lloyd's algorithm. **Figure 2** shows a an example of split plane for a small group of 2D points.

Each split iteration the cluster with the largest sum-squared error is split into two clusters. The splitting plane is defined using the principal component of the cluster as the normal and intersecting the mean point.

Principle Component Calculation

This research examines two different methods for calculating the principal component of the data which signifies the major axis of the data. The EM algorithm [4] iteratively finds a solution using the data while a classical solution calculates the main eigen-vector of the data's covariance matrix.

Both solutions involve large number of sum reductions of the data set using CUDPP [1]. Generally the EM algorithm is significantly faster. However the classical approach works very quickly for low dimensions.

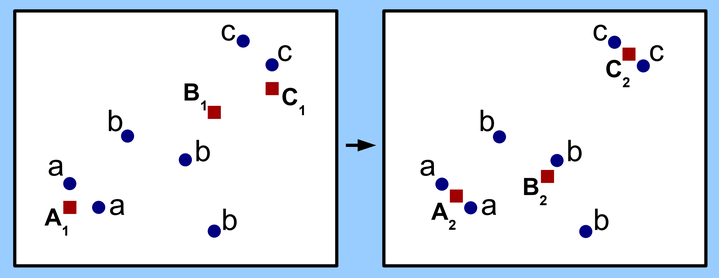

Figure 1. An iterative step of Lloyd's Algorithm. Cluster points (red) are redefined as the means of assigned data points (blue).

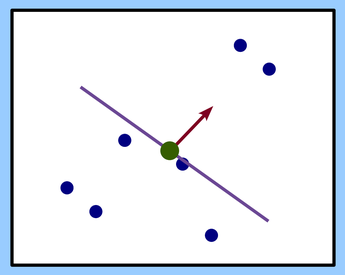

Figure 2. The principal component (red) defines the splitting hyper-plane (purple) that runs through the mean (green)

Cluster Splitting

The main splitting algorithm is outlined in the steps below and shown in:

- Classify each data point to one side of the hyper-plane
- Store a classification (e.g. use 32-bit integers as below)
  - 0x00000001 for the first group
  - 0x00010000 for the second group
- Apply a parallel prefix sum over the identifiers (see **Figure 3**)
- Use output indices to copy data into new grouped locations
- Calculate means and errors for new groups using sum reductions

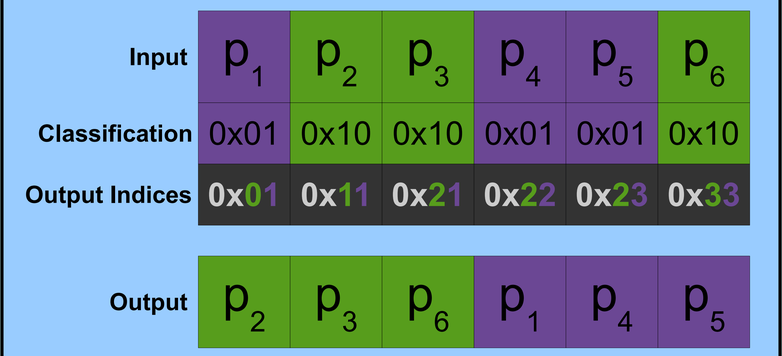

Figure 3. Example of grouping technique for PCA splitting. Input data is classified into two groups. Prefix sum of classifications gives the output indices. The first and second half of the output index gives the index for the first (green) and second (purple) groups respectively

GPU/CPU Hybrid

The GPU gives a significant speed-up for the splitting algorithm for data-sizes over approximately 3000. However, for data sets smaller than 3000, the CPU gives better performance due to the overhead of kernel launches. Hence a hybrid version is currently being developed that uses the GPU and CPU appropriately for different data sizes.

When a split occurs on the GPU, it checks to see whether the new clusters are smaller than a threshold. If this is the case, the data is copied to the CPU and further splits on those clusters are executed with the CPU.

Results

As shown in **Figure 4**, Lloyd's Algorithm performs up to 8x faster than an equivalent optimized CPU implementation. The KD-tree implementation on the CPU shows the importance of fast nearest neighbor searching but is still over 2x slower than the GPU's linear search for larger data sizes.

The PCA algorithm performs very well on the GPU for large clusters, offering speed-ups of over 10x compared to an optimized CPU version.

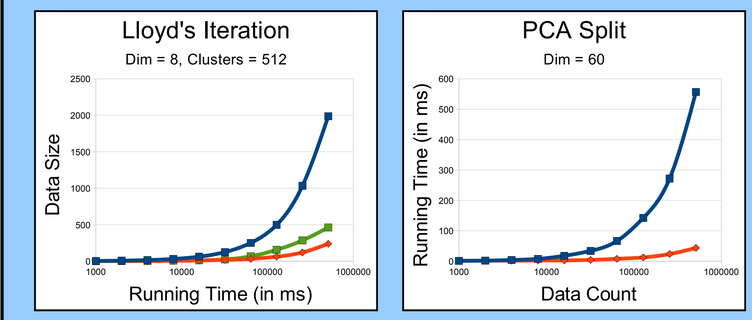

Figure 4. Graphs showing the performance of an iteration of the Lloyd's Algorithm and a PCA split. GPU shown as red, CPU as blue and CPU using KD-tree as green

Conclusion and Future Work

For large data sets, the GPU executes both Lloyd's Algorithm and the PCA splitting technique very quickly. However for best performance, the CPU should handle splits of small clusters and a hybrid version is a future project for development. Furthermore Lloyd's Algorithm scales very well for large data counts and would greatly benefit from efficient use of KD-trees which is future area of research.

References

[1] CUDPP 1.1: CUDA Data Parallel Primitives Library, 2009. http://gpgpu.org/developer/cudpp/
[2] Dick C., Schneider J., Westermann R.: Efficient Geometry Computation for GPU-based Decoding in Real-time Terrain Rendering, 2009, Comput. Graph. Forum 28(1): 67-83
[3] Hall J., Hart, C.: GPU Acceleration of Iterative Clustering, 2004, manuscript for poster at SIGGRAPH 2004
[4] Phani Kumar N. S. L., Satoor, S., Buck I.: Fast Parallel Expectation Maximization for Gaussian Mixture Models on GPUs using CUDA, 2009, IEEE International Conference on HPC and Communications