CS 240A Parallel Computing Project Final Report

Christopher H. Gorman and Nimisha Srinivasa

March 18, 2016

1 Introduction

With the vast quantities of information being is collected in this technological age, hidden connections may be found between data sets. Given that these large amounts of information are likely distributed, effectively using computing resources is critical to glean the most links between data sets as quickly as possible. Because of this, a variety of techniques have been developed in recent years to tackle these challenges. In this report we will talk about Principle Component Analysis (PCA), one method for decomposing data into its primary components. We will briefly review PCA in the next section. Later sections will be devoted presenting our results of using a graphical processing unit (GPU) to help speedup the computations.

2 Principle Component Analysis

Principle Component Analysis is based on the idea that certain directions contain more information. PCA systematically changes the current dataset into one which coordinate axes contain the maximum variance; that is, the first components contain the most important information. More detailed information about Principle Component Analysis can be found in *Principle Component Analysis* by I. T. Jolliffe.

2.1 Brief Review of Linear Algebra

We let A be our original data set, with each row corresponding to a trial and each column corresponding to a particular feature. The shifted data

$$X = A - \overline{A}, \tag{1}$$

where $\overline{A}$ is the matrix of mean values for each feature. The resulting matrix X then has mean 0 in each feature. Principle Component Analysis entails computing the Eigenvalue Decomposition (EVD) of the matrix

$$\frac{1}{n-1} X^T X = V J V^{-1}, \tag{2}$$

where V is an invertible matrix and J is a diagonal matrix whose diagonal elements are the eigenvalues of X. Now, because $X^T X$ is a real symmetric matrix, we actually know that V can be chosen to be an orthogonal matrix and so $V^{-1} = V^T$.

Instead of focusing on computing the EVD, we can also compute the Singular Value Decomposition (SVD) of X:

$$X = U\Sigma V^T, \tag{3}$$

where U and V are orthogonal matrices and Σ is a diagonal matrix whose elements are nonnegative real numbers which decrease in magnitude and are called the *singular values* of the matrix. We see that this is equivalent, for we have

$$\begin{aligned} \frac{1}{n-1}X^TX &= \frac{1}{n-1}V\Sigma^TU^TU\Sigma V^T \\ &= V\left(\frac{1}{n-1}\Sigma^2\right)V^T. \end{aligned} \tag{4}$$

Thus, we see that there is an intricate connection between the EVD and SVD. From here, we see that if we can compute the SVD of a matrix quickly, then we can quickly compute the Principle Components of a dataset, helping us efficiently decribe real-world data sets using a small number of components.

The main challenge then is computing the Singular Value Decomposition. It turns out that this is a computational expensive task. We assume the matrix $X \in \mathbb{R}^{m\times n}$, so that it has m trials and n features. We usually assume that $m \gg n$. In this case, the cost of computing the SVD is approximately $4m^2n$. The main cost in this algorithm is first reducing X to bidiagonal form, so that

$$X = U_BBV_B^T, \tag{5}$$

where U_B and V_B orthogonal matrices and B is an upper bidiagonal matrix. From here, it takes $O\left(n^2\right)$ operations to compute the SVD of B. Putting this all together gives us the SVD of X.

Instead of directly computing the SVD of X, we could first compute the QR factorization of X, where

$$X = QR, \tag{6}$$

where we have Q is an orthogonal matrix and R is an upper triangular matrix. Now, as stated R has the same dimensions of the original matrix X, but in practice an "economical" QR factorization is used, so that R is a square matrix and Q has linearly independent columns. If we first compute

$$X = QR \tag{7}$$

and then

$$R = U\sigma V^T, \tag{8}$$

we will obtain

$$X = (QU)\,\Sigma V^T. \tag{9}$$

This is the Singular Value Decomposition of X. It turns out that it takes $2mn^2$ operations to perform the QR factorization. As we can see, this results in far fewer flops, especially when there are many more trials than features, which is often the case in practice. AFter the QR factorization, the overall process for computing SVD of R is only $4n^3$ operations, which will often be negligible with regards to the total number of flops.

2.2 Parallelization of PCA

From the previously is Sec. 2.1, the primary computational cost in Principle Component Analysis comes from the QR factorization and the Singular Value Decomposition. To do this, we look at performing these computations on the Nvidia Cuda GPU on the Comet cluster at the San Diego Supercomputing Center running CuSolver 7.0. Due to the ubiquitous nature of these factorizations, the main problem is finding libraries that can perform these factorizations quickly. Using GPUs, the main difficulty is splitting up the different tasks efficiently and having the entire unit work together. The more effectively libraries are able to parallelize these factorizations, then the faster these computations can be performed and the more quicly inferences can be made between data sets.

In our particular algorithm, we first zero (shift) the data so that each feature has zero mean. From here, we perform QR factorization on X and then the Singular Value Decomposition on $R = U\Sigma V^T$. This implies that we have

$$\frac{1}{n-1}X^TX = V\left(\frac{1}{n-1}\Sigma^2\right)V^T. \tag{10}$$

In our computation, Cuda forces us to store Q, R, U, Σ, and V^T. This is due to library requirements. Hopefully, future versions of this library would allow one to not store Q and U. This would decrease on some of the computing cost. The need for parallelization may not be apparent for small data sets, but any cost reduction means that larger data sets can be analyzed.

Different areas of data mining have different tools for investigating data. The advantage that PCA has is that the main methods used are matrix factorizations, common in numerical linear algebra. Thus, this means that PCA users do not necessarily have to know the intricate details of the various implementations, as others have and will put much time into this area.

When running one hundred trials Singular Value Decompositions (first computing $X = QR$ followed by $R = U\Sigma V^T$), on ten thousand by three hundred random matrix, we found that 1.28 seconds was spent on performing the QR factorization while 0.04 seconds were spend on the SVD. So, 97% of the computational cost comes from computing the QR factorization. Thus, it is valuable for us to parallelized the QR factorization for an overall speedup. Of course, parallelizing the SVD portion is useful as well, but the focus should be on parallelizing QR.

3 Literature survey

In [1], Seshadri *et al* discuss their efforts to parallelize the formation of the covariance matrix and its eigenvalue decomposition using a PowerXCell 8i processor and SIMD vectorization. They were able to see a speedup of between 16x to 40x, with smaller speedup coming from a smaller data set. These results are expected.

On the matrix computations side, *Matrix Computations* by Golub and Van Loan [2] is necessary for anyone who wants to study numerical linear algebra. This book contains information about QR factorization and the Singular Value Decomposition and describes the standard techniques. In addition, the book also has extensive references to journal articles, important when trying to get the latest information. All algorithm complexity results in this report come from this book.

4 Results

We used 2 datasets to evaluate our code. Firstly, we used the 10k US Adult Faces Database to work on. The data contains a matrix for the Psychological study associates attributes to photos with 33,431 tests and 20 attributes. The sample attributes include Atypical, Boring, Cold, Common, Confident, Trustworthy, Unfriendly, etc. We then used random matrices to test the program behavior at various matrix size.

We now compare the results between the MKL library and using Nvidia GPU. First, the most noticeable feature is that the MKL library always has a higher flop rate than the GPU. Initially this was both troubling and confusing, but we ran across a online discussion at [3] and were able to make more sense of the matter. It turns out that Nvidia CuSolver 7.0 was the first time an SVD solver was a routine. Furthermore, the routine was not optimized; rather, it was there so that it could be used. Future releases would focus on increasing efficiency. This can be seen in the results, especially as we consider that the MKL is optimized. From the above online forum, it is clear that processing power alone does not give high performance; parallelizable code is needed as well. Now, we do note that the flop rate for the QR factorization on the GPU is high, but this makes sense given that it has been optimized.

There was some difficulty trying to run our algorithm using only two or three GPUs on Comet, so we focused on using one GPU or four GPUs. We ran a number of different iterations in order to see the flop rate. Table 1 shows the results of performing QR factorization once followed by multiple iterations of factorizing R. From here, we see that the flop rate is large, although it is an order of magnitude slower than the results for the MKL, found in Table 3. We also show the results for the second algorithm, where one

5 Conclusion

To summarize, Principle Component Analysis is one method to gather information from large data sets. The large computational requirements can still lead to less overall time by parallelizing the algorithm. From our results, we know that optimized libraries can make the difference. Our results suggest that graphical processing units may be helpful in parallelizing algorithm assuming that good numerical libraries exist.

References

[1] Gautam Seshadri, Ramnik Jain and Ankush Mittal. *Parallelization of Principal Component Analysis (using Eigen Value Decomposition) on Scalable Multi-core Architecture* Advance Computing Conference (IACC), 2010 IEEE 2nd International.

[2] Gene H. Golub and Charles F. Van Loan *Matrix Computations* 4th Edition.

[3] NVIDIA Cuda Zone Forums
`https://devtalk.nvidia.com/default/topic/830894/cusolverdncgesvd-performance-vs-mkl/`

Number of Iterations	GPU size	Kernel time(ms)	MFlops/s
5	1	959.719849	562.664
	4	342.210083	1577.978
10	1	2800.846924	385.598
	4	962.197754	1122.43
100	1	N/A	N/A
	4	10051.44922	1074.472
1000	1	89426.70313	1207.693
	4	88738.96094	1217.053

Table 1: Algorithm: QR once followed by SVD performed in multiple iterations.

Iterations	GPU size	Kernel time(ms)	MFlops/s (overall)	MFlops/s (QR)	MFlops/s (SVD)
5	1	8617.817383	487.521	99882.652	62.661
	4	2011.598267	1627.467	107160.68	268.443
10	1	16270.14648	508.099	90154.579	66.379
	4	2949.37793	1896.733	92595.732	366.179
100	1	N/A	N/A	N/A	N/A
	4	24623.41406	2060.314	88128.938	438.607

Table 2: Algorithm: Multiple iterations of QR followed by SVD.

Iterations	Kernel time	MFlops/s
5	4.300468	1.40E+04
10	9.731463	1.23E+04
100	578.454099	2.07E+04
1000	606.856085	1.98E+04

Table 3: Algorithm: Standard BLAS3 implementation of QR factorization followed by SVD using MKL.