ALS Implicit Collaborative Filtering with GPUs

Lucas Tata
Department of Electrical and Computer Engineering
University of Massachusetts - Lowell
Lowell MA, USA
Email: lucas_tata@student.uml.edu

Salvatore Amico
Department of Electrical and Computer Engineering
University of Massachusetts - Lowell
Lowell MA, USA
Email: salvatore_amico@student.uml.edu

Abstract

Alternating Least Squares (ALS) implicit collaborative filtering is type of recommender system. Our goal in this research is to first implement this algorithm on a CPU and improve the computation time of the algorithm with CUDA GPU speedup using various techniques. We show that speedup on the GPU is attainable, and has significant improvement over CPU computation.

I. Introduction

ALS implicit collaborative filtering is type of recommender system. This algorithm takes in a dataset of users and the artists that they like, compare it to the user in question, and make a recommendation based on these factors. For instance running the algorithm for user zero will have an output of five names of five different artist that the algorithm predicts user zero will like based on their previous listens. The data used for this algorithm is implicit, or data which is gathered passively from the users without their input or opinion in it, simply raw data of their usage. By using this meta data, more accurate results can be obtained and therefore useful trends can be established and implemented.

Our goal in this research is to first implement this algorithm on a CPU, and improve the computation time of the algorithm with CUDA GPU speedup. The dataset we will use for the proof of concept for this recommendation system is the LastFM dataset, which is filled with random user and artist data for music recommendations.

II. Background

Our CPU implementation is programmed in C. It is segmented off into three different parts: The I/O reading of the dataset and manipulating the data, running the ALS filtering, and finally using the filtered data to give recommendations.

Our I/O of the data set involves going through the data set file, which is a tab-separated file, and adding the user/artist data to our string matrices that we use to hold the names of users and artists. We make sure to check that the users and artists do not exist in our matrices before we add them. We prepare our sparse matrix of data by adding the number of plays for that particular user and artist combination to the matrix for each entry, and then set the rest of the matrix to 0. We now have our sparse matrix, and our matrices that map each user and artist id to indexes in the sparse matrix.

We now start our ALS implicit collaborative filtering algorithm. We separate our sparse data matrix into two separate matrices, one for the user space and one for the artist space. We use these matrices and initialize them to random values. We take these matrices and aim to minimize this loss function as shown in Equation 1:

$$\min_{y_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \| x_u \|^2 + \sum_i \| y_i \|^2 \right) \tag{1}$$

The loss equation can be derived down to the following two equations to minimize the loss of both the users and the artists as follows in Equations 2 and 3:

$$x_u = (Y^T Y + Y^T (C^u - I) Y + \lambda I)^{-1} Y^T C^u p(u) \tag{2}$$

$$y_i = (X^T X + X^T (C^i - I) X + \lambda I)^{-1} X^T C^i p(i) \tag{3}$$

After the loss equation is derived Y transpose, Cu and Y are multiplied in the fashion shown below in Equation 4.

$$Y^T C_u Y = Y^T Y + Y^T (C_u - I) Y \tag{4}$$

The final step in conditioning the data is to redefine x_u and y_i to acquire a more efficient final form for the actual finding of the recommendations as shown in Equations 5 and 6.

$$x_u = (Y^T Y + Y^T (C^u - I) Y + \lambda I)^{-1} Y^T C^u p(u) \tag{5}$$

$$y_i = (X^T X + X^T (C^i - I) X + \lambda I)^{-1} X^T C^i p(i) \tag{6}$$

The final step in our code is the actual finding of our answer. To make a recommendation the greatest values for each artist of a certain user are analyzed. These values are then ordered greatest to least and the top 5 artists are given as recommendations for that given user.

III. Implementation/Techniques

A. GPU Implementation

Once we had our CPU implementation ironed out, we pursued our implementation using CUDA kernels, with intent to speedup our computation time. Our starter code for our GPU implementation was our CPU implementation. Because we already had the algorithm working correctly with C code on the CPU, we needed to isolate parts of our code that were blaringly parallelizable.

In our ALS implicit function, we have numerous matrix-matrix numerical operations (multiplications, divisions, additions, subtractions), matrix transposes, and matrix initializations. We started CUDA parallelization by creating kernel calls for these three areas. Once we had basic implementation on the GPU for these areas, we explored other areas that could increase our speed inside that ALS function. Another largely parallelizable area of our CPU implementation is the fact that for every single user and every single artist, these matrix operations compute over and over. There were still loose ends that were not parallelized in our algorithm, and we continued to add kernel calls for anything that seemed applicable.

B. Technique I

Our most time consuming individual kernel was matrix multiplication. We first started with a generic shared memory block implementation of this kernel, but were not seeing much of a speed increase for that kernel. Looking more in depth to the data we were inputting to this kernel, we found that since all of our matrices inside this kernel had one dimension of the matrix in terms of the number of features chosen, we were not fully utilizing the GPU. Our number of features chosen was 10, and using a kernel call of 256 blocks and 256 threads per block, we were allocating 256 threads to only 10 entries, which is wildly inefficient. We were able to improve upon this and allow one block to take control of 25 rows, in which we saw a much better speed increase. We combined our shared memory implementation with some atomic addition calls, and we saw a very small speed increase as well.

We were seeing much faster computation time, but not the type of speeds we were hoping for. Upon analysis of computation time of certain portions of our code, we found that one of the most time consuming tasks of our code was allocating and copying memory to the GPU. Because we were still allocating memory, copying memory, and calling multiple kernels numerous times for each and every user and artist in our data, the memory allocation was a large bottleneck.

We changed and we were able to allocate the memory on the GPU at the very beginning of our function call, and only allow for memory copying of the data that was being altered mid algorithm. We found a major speed increase after making this change.

C. Technique II

Upon improving with this method, another bottleneck is that we still serially call our kernels numerous times for each user and each artist. Our first inclination was to parallelize this method. We started taking portions of our code outside of the artist and user loops, and calling kernels outside of it to increase speed. We were successful with these changes, but we still were capped at around a 10x speed increase.

Our final major improvement attempt was combining all of our kernel calls into one call, which would calculate the entire ALS algorithm in one call. We were able to do this by figuring out all of our allocation and memory copying needed for the previous kernel calls, and pushing the kernel code together for each kernel call. This improvement was our largest increase, bringing our overall speedup from 10x to 600x from the CPU to the GPU version.

IV. EVALUATION

Our CPU implementation successfully was able to successfully recommend a number of artists for a specific user based on their current listening preferences. We ran our CPU implementation and the average runtime was 20 minutes, 1200 seconds.

With our first GPU implementation, namely the one that used numerous kernel calls, our runtime was 2 minutes 30 seconds, 150 seconds. This is a speedup of 8x. We were able to see the same recommendations, which verifies that the algorithms are equivalent.

With our second GPU implementation, namely the one that used one kernel call for our ALS algorithm, our runtime was a little under 2 seconds. This is a total speedup of 600x. We were able to see the same recommendations, which verifies that the algorithms are also equivalent.

V. RELATED WORK

There are a number of related articles in this field of collaborative filtering and recommendation systems. The majority of the field falls into one of two buckets. The first bucket is a standard of ALS implicit collaborative filtering and modifications on that algorithm, and the second bucket is the use of another algorithm for recommendation systems.

The most important related field of work is a Medium article by Victor Kohler named ALS Implicit Collaborative Filtering. This article goes into detail on the ALS algorithm, and walks through a Python implementation of the algorithm. This was a very good starting basis for our project. [1]

Like our algorithm the team combination and AT&T Labs and Yahoo! Research, "relies only on past user behavior without requiring the creation of explicit profiles. This approach is known as Collaborative Filtering." Essentially their algorithm also only uses implicit data as ours does in contrast to methods that use as they describe a "content based approach." Their method is also similar to ours in that it uses many matrix transforms and transposes showing their data is stored in a way similar to ours. Their confidence level however had a much higher threshold their ours did. Ours was simply based off a proportional amount of listens while their algorithm was based on if you watched over 30 minutes of a TV show. Their algorithm had a problem however that ours did not that it very often recommended very similar shows to everyone. This may be due to their dataset but approximately 40% of the recommendations were very similar for all users. [2]

In, "An Implicit Data Structure Supporting Insertion, Deletion, and Search in O(log' n) Time*" a much more efficient method than ours is shown. This method while much faster may not have the same accuracy of results. The main focus in their paper is the speed of the algorithm not necessarily the usage missing out on things such as confidence and not recommending already "liked" items. [3]

"Matrix Factorization Techniques for Recommender Systems" is a paper by Yehuda Koren of Yahoo Research and is on a more efficient content filtering approach. Their paper focuses on matrix factorization and how speedups in this sector can create faster and more efficient algorithms. Their paper moves to prove that, "matrix factorization models are superior to classic nearest neighbor techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels." This method is similar to ours in spirit as we use implicit feedback and confidence levels however we do not use temporal effects. In going through their paper it is clear while our focus was on creating efficient GPU algorithms their focus was more on the math and writing more efficient "math scripts" that speed up the process intrinsically by being less compute intensive. [4]

In "Proceedings of the 2016 SIAM International Conference on Data Mining" in the SIAM Journal, they talk about using matrix factorization techniques to aid collaborative filtering. This approach increases accuracy of the collaborative filtering method, by exploring multiple ways to factorize the matrices. We believe this is important work because it is closely related to increasing the accuracy of recommendation models. [5]

In, "Fast ALS-Based Tensor Factorization for Context-Aware Recommendation from Implicit Feedback" by Balázs Hidasi and Domonkos Tikk they focus on speeding the algorithm using tensor-based factorization. Hidasi says, "ALS-based tensor factorization learning method that scales linearly with the number of non-zero elements in the tensor. The method also allows us to incorporate various contextual information into the model while maintaining its computational efficiency. " In essence their method is one that attempts to scale linearly by using tensor. Our algorithm also scales very well linearly as ours does do to the GPU resources being able to be allocated as they are in our algorithm. Their method however does introduce another element of incorporating more contextual information while ours solely relies on the implicit data. [6]

One-Class Collaborative Filtering is a paper in which a method is described for attempting to solve the "one-class" problem through collaborative filtering. The "one-class" problem is the fact that in implicit data someone simply not "bookmarking" a site doesn't mean they don't like it but if you are looking at a dataset of bookmarks it may appear that way. Their paper is similar to ours in that it uses collaborative filtering and confidence levels to try to make more accurate predictions. Overall our methods our similar with theirs being slightly more robust in attempting to filter out false positives and in their case also false negatives. [7]

In "Improving Recommendation Quality by Merging Collaborative Filtering and Social Relationships", they talk about matrix factorization techniques that can improve recommendation quality. The main method they use to improve the recommendations is to merge collaborative filtering techniques with added data about social relationships. This is an interesting concept to adapt collaborative filtering techniques, and use them as a basis instead of the end result of recommendation systems. We believe this idea of building on these filtering techniques will allow for better user recommendations. [8]

In, "Algorithmic Acceleration of Parallel ALS for Collaborative Filtering: Speeding up Distributed Big Data Recommendation in Spark" by Manda Winlaw of the Dept. of Appl. Math., Univ. of Waterloo they propose to, "accelerate the convergence of parallel ALS-based optimization methods for collaborative filtering using a nonlinear conjugate gradient (NCG) wrapper around the ALS iterations." The paper also looks into "a parallel implementation of the accelerated ALS-NCG algorithm in the Apache Spark distributed data processing environment" Their algorithm while being in the same spirit as ours for an ALF iterative method is in fact much more complex. They're approach is simply much more math based than ours is and seems to be extremely efficient. With a starting biases and their math I believe GPU acceleration could do a lot to make their algorithm even better. [9]

"Expectation-Maximization Collaborative Filtering with Explicit and Implicit Feedback" is a paper by Bin Wang, Mohammadreza Rahimi, Dequan Zhou and Xin Wang on collaborative filtering with their method of, "Expectation-Maximization Collaborative Filtering (EMCF), based on matrix factorization." Essentially their method is an approach that combines implicit and explicit feedback together to attempt to make more accurate results. Due to the type of dataset we use for our project this would not be something that's possible for our specific implementation but I believe in the proper dataset their algorithm would be more accurate. With more data types and data points you can really hone in on what each user wants and with that give the best recommendations. We believe in this case more data is better and that their method would provide even better results than ours. [10]

VI. Conclusion

Our goal in this research was to implement the ALS algorithm on a CPU, and improve the computation time of the algorithm with CUDA GPU speedup. We were able to successfully implement this algorithm on the CPU and GPU, with the GPU version having significant speedup. We did not have time to directly contribute to another path of current research in the field, but we believe this was a very fulfilling and interesting experience for us to further our careers. This was our first attempt at a large-scale CUDA project, but will not be the last.

Acknowledgment

We would like to acknowledge our university, University of Massachusetts – Lowell, for providing a beneficial Electrical and Computer Engineering program. We would also like to acknowledge our High Performance Computing for GPUs professor, Professor Hang Liu, for pushing us to pursue such an interesting project topic.

References

[1] Köhler, Victor. “ALS Implicit Collaborative Filtering – Rn Engineering – Medium.” *Medium*, Augmenting Humanity, 23 Aug. 2017, medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe.

[2] Hu, Yifan. “Collaborative Filtering for Implicit Feedback Datasets.”

[3] Munro, Ian. An Implicit Data Structure Supporting Insertion, Deletion, and Search in Olog2n)

[4] Koren, Yehuda. “Matrix Factorization Techniques for Recommender Systems - IEEE Journals & Magazine.” IEEE Conference Publication, Wiley-IEEE Press, ieeexplore.ieee.org/document/5197422/#full-text-section.

[5] Liu, Xinyue. “Proceedings of the 2016 SIAM International Conference on Data Mining.” SIAM Journal on Computing, epubs.siam.org/doi/abs/10.1137/1.9781611974348.43.

[6] Hidasi, Balázs, and Domonkos Tikk. “Fast ALS-Based Tensor Factorization for Context-Aware Recommendation from Implicit Feedback.” SpringerLink, Springer, Dordrecht, 24 Sept. 2012, link.springer.com/chapter/10.1007/978-3-642-33486-3_5.

[7] Pan, Rong. “One-Class Collaborative Filtering - IEEE Conference Publication.” IEEE Conference Publication, Wiley-IEEE Press, ieeexplore.ieee.org/document/4781145/#full-text-section.

[8] Meo, Pasquale De. Improving Recommendation Quality by Merging Collaborative Filtering and Social Relationships - IEEE Conference Publication. Wiley-IEEE Press, ieeexplore.ieee.org/document/6121719/#full-text-section.

[9] Winlaw, Manda. Algorithmic Acceleration of Parallel ALS for Collaborative Filtering: Speeding up Distributed Big Data Recommendation in Spark - IEEE Conference Publication. Wiley-IEEE Press, ieeexplore.ieee.org/document/7384354/#full-text-section.

[10] Wang, Bin, et al. “Expectation-Maximization Collaborative Filtering with Explicit and Implicit Feedback.” SpringerLink, Springer, Dordrecht, 29 May 2012, link.springer.com/chapter/10.1007/978-3-642-30217-6_50.