Parallelize Matrix factorization using OpenMP

Arnav Kansal **Ojas Deshpande**
{ak6806, oad230}@nyu.edu

1 Introduction

Recommender systems have been in wide use in the e-commerce, music and entertainment industry lately. After the Netflix Prize and the work by Koren et.al.2009 [1] matrix factorization methods have been a major component in the recommendation system toolboxes. As the name suggests, the application is an effective method to factorize a matrix into the product of 2 matrices. In this course project we wish to implement matrix factorization using Stochastic gradient descent methods in OPENMP for large matrices.

1.1 Mathematical formulation

Decompose $\mathbf{R} \in \mathrm{R}^{mxn}$ into product of 2 matrices **X, Y** $\in \mathrm{R}^{mxk}$ and R^{kxn} to compute **XY** $\approx$ **R**.

R = XY

Here **R** may have missing (unknown) entries and thus has wide applications in the domain of recommendation systems for instance, finding the missing entries of **R**. We will not talk about the origin of such matrices in detail. But just as an example i-j th element of **R** may represent i th user's preference for j th element of an e-commerce website.

Different mathematical formulations are used for this problem, and the most general optimization problem that it represents is:

$$arg\min_{X,Y} f_R(X,Y)$$

An example of such a function f_R is:

$$\min_{X,Y} ||\mathbf{R} - \mathbf{XY}||_F^2 + \lambda||\mathbf{X}||_F^2 + \mu||\mathbf{Y}||_F^2$$

One may be tempted to simply apply the k-rank approximation of **R** obtained using SVD, but doing so is not possible because many entries of **R** may be missing.

2 Dataset

We intend to use MovieLens [2], Netflix [3] and Yahoo-music dataset [4]. Also we plan to create a random dataset of custom size by sampling from a random number generator to produce matrices of varying size to experiment and observe speedup.

3 Survey of related work

The problem of matrix factorization has been solved primarily via three classes of algorithms. The three algorithm classes employed are Alternating Least Squares(ALS) , Stochastic gradient descent(SGD) and Coordinate Descent (CD). The current state of the art methods involve these algorithms as the base idea and evolve on the shortcomings of the degree of parallelization of these algorithms to get better performance on real life datasets. None of these methods can be applied to

Course project : *Multicore Processors: Architecture & Programming*, Spring 2018 at New York University.

the problem setting in a vanilla formulation as all of these algorithms have inherent sequential update rules.

ALS is a very simple idea to understand and has excellent theoretical bounds for convergence rates. ALS relies on optimizing for X, given a fixed Y and vice versa alternatively to converge to a perfect pair of matrix factors. Thus only after finding one perfect X leads to optimization for Y and so on. This sequence is inherently sequential and has dependencies between each iteration of the algorithm.

Coordinate descent is another popular algorithm in the field of optimization. Yu, Hsiang-Fu, et al.2012 [5] have applied Coordinate Descent approaches to update the matrix **X** and **Y** in an alternating fashion. So they have utilised both ALS and coordinate descent to formulate their problem.

SGD is a very well known algorithm used in many applications of machine learning because of the minimum memory footprint of the algorithm. Also it is very robust against noise in the data and guarantees fast learning rates. But inherent sequential behaviour of the algorithm limits its use. Recht, Benjamin, et al.2011 [6] have come up with a lock free approach to parallelize SGD and have shown that usually the optimization problem is sparse and that most gradient updates only modify small parts of the decision variable and thus can be parallelized by modifying the sampling scheme of the underlying data. This idea has been further utilized by Chin, Wei-Sheng, et al.2015 [7] for shared memory systems. They have reduced the cache-miss rate and adjusted for the dynamic load balancing of the threads to achieve a state of the art matrix factorization library LIBMF.

4 Proposed Approach

We will be mostly be following the approach used by the libMF paper, i.e., using stochastic gradient descent (SGD) to solve matrix factorization. Our loss function would be a non-convex loss function $f_R(X, Y)$ as mentioned above (optionally with some additional regularization terms that minimize the norms of the matrix factors produced).

The main method of parallelizing matrix factorization is based on the assumption that given a matrix **R** to be factorized, different blocks (non-overlapping submatrices that don't share a column or row) of this matrix are independent from each other. If this is true, then different threads can simultaneously update the variables for these blocks in parallel.

The choice of selecting which thread would work on which block is important, and usually the decisive factor in determining the efficiency of this approach. As in libMF, we'll follow the technique of assigning the next free thread the least recently updated free block (no other thread is working on updating this block right now, and this way all blocks get updated periodically). It is easy to show that with this approach, using t threads, we need atleast to break **R** into $(s + 1) * (s + 1)$ blocks.

There's another issue that needs addressing which is how we access the entries in a given block. When a thread accesses the entires of a block discontinuously, it creates a lot of cache misses, which again makes our program less efficient. This is addressed by instead accessing the entries in a fixed order (either row-wise or column-wise), so that one of the matrices **X** or **Y** can now be accessed continuously.

Combining both the above methods, the overall algorithm is as follows:

Figure 1: Pseudocode for the mentioned approach

1: randomly shuffle R
2: grid R into a set B with at least $(s + 1) \times (s + 1)$ blocks
3: sort each block by user (or item) identities
4: construct a scheduler
5: launch s working threads
6: wait until the total number of updates reaches a user-defined value

References

[1] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[2] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[3] James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[4] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The yahoo! music dataset and kdd-cup'11. In *Proceedings of KDD Cup 2011*, pages 3–18, 2012.

[5] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 765–774. IEEE, 2012.

[6] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[7] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(1):2, 2015.