# An Efficient Implementation of the ALS-WR Algorithm on x86 CPUs

Maosen Chen[1], Tun Chen[2,⋆], and Qianyun Chen[3]

[1] Qihoo 360 Technology Co. Ltd.
chenmaosen@360.cn
[2] SKL of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
chentun@ict.ac.cn
[3] College of Computing, Georgia Institute of Technology
qchen336@gatech.edu

**Abstract.** With the continuous development of computers and big data technology, more recommendation systems are applied in the fields of online music, online movies, games, online shopping, and so on, to solve information redundancy and effectively to recommend interesting products for users. In this paper, we implement and accelerate the Alternating-Least-Squares with Weighted-$\lambda$-Regularization (ALS-WR) by adopting a two-level parallel strategies on the x86-64 Zen-based CPUs. As one of the most widely used recommendation algorithms, the ALS-WR algorithm is based on matrix factorization. In the mathematical discipline of linear algebra, a matrix decomposition or matrix factorization is a dimensionality reduction technique that factorizes a matrix into a product of matrices. Therefore, vector and matrix operations are the computational core of the ALS-WR algorithm, accelerating these computational kernels can effectively improve the overall performance of the ALS-WR algorithm. The experimental results show that our high-performance ALS-WR implementation can achieve 185.09 seconds (with 100 features and 30 iterations) on the MovieLens 20M dataset.

**Keywords:** ALS-WR · Matrix Factorization · Matrix Multiplcation · Recommendation Algorithm.

## 1 Introduction

Recommender systems are utilized in a variety of areas and are most commonly recognized as multimedia recommenders like Netflix, YouTube, and Spotify, product recommenders such as Amazon, Taobao in Alibaba, or content recommenders for social media platforms such as Facebook and Twitter [7]. These systems aim to provide users with personalized online product or service recommendations by predicting the rating score or preference that a user would give to an item based on users' buying or browsing behaviors. In this way, the user can

⋆ Corresponding author

handle the increasing online information overload problem and improve customer relationship management. Given the predicted rating score, the business will then recommend preferable new items to the user for further purchases. There are three main kind of recommendation systems: content-based, collaborative filtering-based and deep learning based. Collaborative filtering method is simple and effective and widely used in lots of internet companies like Amazon [16], Google News [2], and other academic contests or related evaluations like AIBench [5] [6], Edge AIBench[8], Netfix, HPC AI500 [13], AIoT Bench [17] and so on. AIBench is open source http://www.benchcouncil.org/benchhub/AIBench/. A major appeal of collaborative filtering is that it is domain free, yet it can address data aspects that are often elusive and difficult to profile using content filtering, while generally more accurate than content-based techniques. Collaborative filtering relies solely on the rating scores that a user gave to the item, where the features of the user (such as age and gender) or the item (such as perishable or not) itself do not play an important role in the algorithm.

BenchCouncil organizes a series of International AI challenges in 2019. It contains four challenges tracks: 1) International AI System Challenge based on RISC-V Subject [11]; 2) International AI System Challenge based on Cambricon Chip [14]; 3) International 3D Face Recognition Algorithm Challenge [23]; and 4) International AI System Challenge based on X86 Platform. Our work is based on the track 4. In this paper, we focus on the Alternating Least Squares with Weighted-$\lambda$-Regularization (ALS-WR) algorithm [24], one of the most commonly used approaches for performing matrix factorization on recommender systems on the MovieLens dataset [10] and update the two low-rank matrices' weight alternately which save a lot of time and operate in a clever way without hurting the performance. The ALS-WR algorithm was developed for the Netflix Prize competition, which also involved a sparse matrix of reviewers by items being reviewed. It has the advantage over the Stochastic Gradient Descent (SGD) [1] and Restricted Boltzmann Machines (RBM) [20] algorithms that require fewer features to be specified and has been previously been shown to have great efficiency potential on CPUs [18].

Assuming that we have a user-item rating matrix $R = \{r_{ij}\}_{u \times m}$. Each element $r_{ij}$ represents a rating score of movie $j$ rated by user $i$ with its value neither being a real number or being missing, $u$ indicates the number of users and $m$ indicates the number of movies. A high-quality recommendation system can estimate some of the missing values based on the existing values.

In real-world problems, the user-item matrix $R$ is normally large, so matrix factorization is introduced to decompose matrix $R$ into products of smaller matrices. As presented in Fig.1 [19], the user-item matrix $R$ is decomposed by the product of two smaller matrices $U = [u_i]$ and $M = [m_j]$. In the factorization process, we assume each row of the user matrix $U$ represents one user with $d$ features. Similarly, each column of the item matrix $M$ represents an analogous set of $d$ features. In order to generate a low-rank approximation of the user-item matrix $R$, all we have to do is to perform a matrix multiplication of the user matrix $U$ and the item matrix $M$.

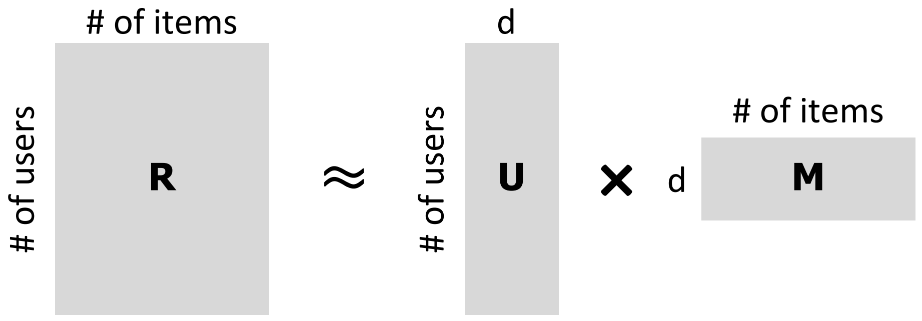

**Fig. 1.** Matrix factorization process.

First, we denote $d$ features with each user by representing each user $i$ has a $d$-dimensional vector ${x_i}^T$, which is normally referred to as the user latent vectors. Similarly, a movie $j$ also has a $d$-dimensional vector $y_j$, and the rating score that we predict user $i$ will give for movie $j$ is the dot product of the two vectors, as presented in Eq.1.

$$\hat{r_{ij}} = x_i {y_j}^T = \sum_d x_{id} y_{dj} \tag{1}$$

where $\hat{r_{ij}}$ represents the approximate pridction of the ground truth rating score $r_{ij}$.

Then we define the objective function to minimize the square of the difference between all rating scores in the dataset $S$ and our predictions, as presented in Eq.2.

$$L = \sum_{i,j \in S} (r_{ij} - x_i {y_j}^T)^2 + \lambda(\sum_i \|x_i\|^2 + \sum_j \|y_j\|^2) \tag{2}$$

where $\lambda$ is a hyperparameter used for preventing overfitting of the user and item vectors.

## 2 Alternating Least Squares with Weighted-λ-Regularization

In order to solve this low-rank approximation problem, the iterative algorithm ALS-WR [24] is adopted. Many optimization [3, 9] approaches are adopted for ALS-WR algorithm. ALS-WR starts by treating one set of latent vectors as constant. For this example, it picks the item vectors $y_j$, then takes the derivative of the loss function with respect to the user vectors $x_i$, and solve for the user vectors, as presented in Eq.3.

$$\begin{aligned} \frac{\partial L}{\partial x_i} &= -2\sum_j (r_{ij} - x_i {y_j}^T) y_j + 2\lambda x_i = 0 \\ &= x_i = r_i Y (Y^T Y + \lambda I)^{-1} \end{aligned} \tag{3}$$

where vector $r_i$ represents the $i^t h$ user from the rating score matrix with all scores for all movies; $Y$ is a $m \times d$ representing all movies row vectors vertically stacked together; and $I$ is the identity matrix with dimension $d \times d$.

Next, after updating the user vectors $x_i$, we take them as constant and alternatively update the item vectors $y_j$ in a similar way, as presented in Eq.4. Then we alternate back and forth and carry out these two steps until convergence.

$$\frac{\partial L}{\partial y_j} = y_j = r_j X (X^T X + \lambda I)^{-1} \tag{4}$$

## 2.1 The ALS-WR Process

In general, the ALS algorithm can be summarized as the following four steps:

1. Initialize matrix $M$ by assigning the average rating for that movie as the first row, and small random numbers for the remaining entries.
2. Fix $M$, Solve $U$ by minimizing the objective function (the sum of squared errors).
3. Fix $U$, solve $M$ by minimizing the objective function similarly.
4. Repeat Steps 2 and 3 until a stopping criterion is satisfied.

Here we use the observed root-mean-square error (RMSE) as the stopping criterion on the MovieLens dataset. Each round we will update both matrices $U$ and $M$, if the criterion is obtained, then we use the obtained matrices $U$ and $M$ to make final predictions on the test dataset.

## 2.2 Two-level Parallelization

In each round of the ALS-WR algorithm, we need to update matrices $U$ and $M$, respectively. In order to accelerate each iteration, we implement a two-level parallel method.

1. **High-level Parallelism.** In order to update each user's feature vector $x_i$, we need to take the rating score matrix $R$ and the movie matrix $M$ as inputs. This process contains native parallelism: we can parallel the computational operations of multiple users and update their feature vectors simultaneously. Similarly, while updating the movie matrix $M$, the same parallel strategy can be adopted.

**Table 1.** Experimental Environment

| **CPU** | Arch. | # of threads | L1d cache | OpenBLAS |
|---|---|---|---|---|
| **AMD Zen** | Zen | 64 | 32 KB | 0.3.7 |

2. **Low-level Parallelism.** The core operations of the ALS-WR algorithm are updating the user matrix $U$ and the movie matrix $M$, as presented in Eq.3 and Eq.4, their operations can be simplified to some basic vector and matrix operations, such as matrix multiplication, matrix-vector multiplication, matrix transposition, and so on. These basic routines are provided by most of BLAS (Basic Linear Algebra Subprograms) libraries. Existing BLAS libraries, such as OpenBLAS [22] and Intel MKL [12] , are highly optimized and tuned by researchers and vendors. These implementations have undergone extensive architecture-dependent tuning on specific microarchitecture features to pursue peak system performance. Most important of all, these BLAS libraries also provide high-performance multi-threaded implementations for users to make full use of their underlying computing resources. As for large matrix multiplication, we can replace it by the FFT algorithm. Efficient FFT librarlies [12, 15, 4] can accelerate this process. In our ALS-WR (Zen-ALSWR), we adopt OpenBLAS to carry out these vector and matrix operations.

## 3 Performance Evaluation

This section evaluates the performance of Zen-ALSWR on server-grade Zen x86-64 (AMD Zen microarchitecture [21]) CPUs. The experimental conditions are listed in Table 1. Considering OpenBLAS 0.3.7 version has been deeply optimized for its kernels for Zen Dhyana zen CPUs, we take this version as our underlying BLAS library to accelerate the performance of vector and matrix operations.

**Table 2.** MovieLens ml-20m dataset

| **Dataset** | # Ratings | # Users | # Movies | Sparsity | RMSE Loss |
|---|---|---|---|---|---|
| **All** | 20,000,263 | 138,493 | 25,809 | 0.540% | Null |
| **Train** | 16,003,852 | 138,493 | 25,809 | 0.448% | 2.882 |
| **Test** | 3,996,411 | 138,493 | 25,809 | 0.112% | 2.473 |

Our experiments are based on the MovieLens ml-20m dataset. We randomly take 80% of the ml-20m dataset according to ratings as training data and take the remaining 20% of the data as the test dataset which is listed in Table 2. Moreover, we set the number of features as 100 and the number of iterations as 30, then the performance will be evaluated by recording the training time (wall clock time).

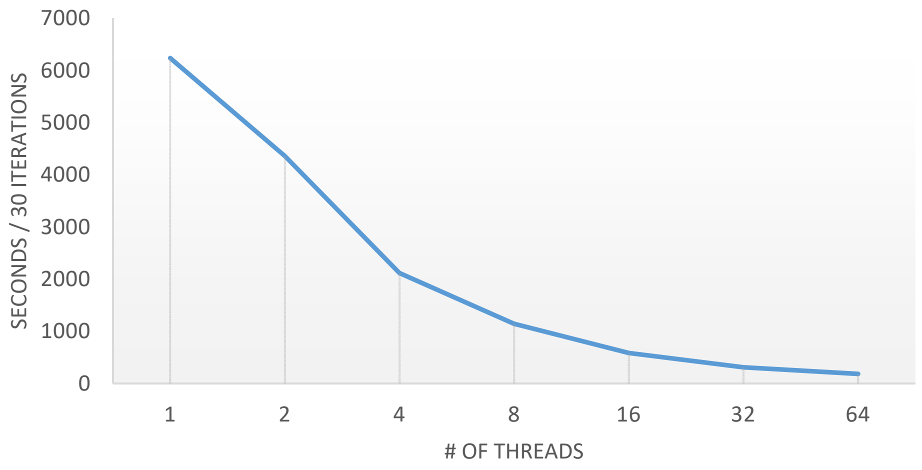

**Fig. 2.** The performance of the multithreaded Zen-ALSWR on MovieLens ml-20m dataset.

Fig.2 represents the performance of our Zen-ALSWR on AMD Zen CPUs. As the number of threads increases, the training time is greatly reduced. For the single-threaded Zen-ALSWR, it costs about 6237.28 seconds to complete the whole training process. When we adopt 64 threads within a node to parallel the training progress, the training time decreases into 185.09 seconds. Fig.2 shows that Zen-ALSWR seems to be a scalable and efficient implementation for large-scale collaborative filtering.

## 4 Conclusion

This paper introduces a high-performance ALS-WR implementation named Zen-ALSWR on AMD Zen CPUs. Because the parallel strategies are straightforward for the ALS-WR implementation, the Zen-ALSWR is designed to be scalable and efficient to very large datasets on server-grade CPUs.

## References

1. Bottou, L., Bousquet, O.: The tradeoffs of large scale learning. In: Advances in neural information processing systems. pp. 161–168 (2008)
2. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: Proceedings of the 16th international conference on World Wide Web. pp. 271–280. ACM (2007)
3. Deng, W., Wang, P., Wang, J., Li, C., Guo, M.: Psl: Exploiting parallelism, sparsity and locality to accelerate matrix factorization on x86 platforms. In: International Symposium on Benchmarking, Measuring and Optimization (Bench19). Springer (2019)

4. Frigo, M., Johnson, S.G.: Fftw: An adaptive software architecture for the fft. In: Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181). vol. 3, pp. 1381–1384. IEEE (1998)
5. Gao, W., Luo, C., Wang, L., Xiong, X., Chen, J., Hao, T., Jiang, Z., Fan, F., Du, M., Huang, Y., Zhang, F., Wen, X., Zheng, C., He, X., Dai, J., Ye, H., Cao, Z., Jia, Z., Zhan, K., Tang, H., Zheng, D., Xie, B., Li, W., Wang, X., Zhan, J.: Aibench: towards scalable and comprehensive datacenter ai benchmarking. In: International Symposium on Benchmarking, Measuring and Optimization. pp. 3–9. Springer (2018)
6. Gao, W., Tang, F., Wang, L., Zhan, J., Lan, C., Luo, C., Huang, Y., Zheng, C., Dai, J., Cao, Z., et al.: Aibench: an industry standard internet service ai benchmark suite. arXiv preprint arXiv:1908.08998 (2019)
7. Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D., Zadeh, R.B.: Wtf: The who-to-follow system at twitter. In: Proceedings of the 22nd international conference on World Wide Web WWW (2013)
8. Hao, T., Huang, Y., Wen, X., Gao, W., Zhang, F., Zheng, C., Wang, L., Ye, H., Hwang, K., Ren, Z., et al.: Edge aibench: towards comprehensive end-to-end edge computing benchmarking. In: International Symposium on Benchmarking, Measuring and Optimization. pp. 23–30. Springer (2018)
9. Hao, T., Zheng, Z.: The implementation and optimization of matrix decomposition based collaborative filtering task on x86 platform. In: International Symposium on Benchmarking, Measuring and Optimization (Bench19). Springer (2019)
10. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. Acm transactions on interactive intelligent systems (tiis) **5**(4), 19 (2016)
11. Hou, P., Yu, J., Miao, Y., Tai, Y., Wu, Y., Zhao, C.: Rvtensor: A light-weight neural network inference framework based on the risc-v architecture. In: International Symposium on Benchmarking, Measuring and Optimization (Bench19). Springer (2019)
12. Intel: Intel math kernel library (intel mkl) 2019 update 4. https://software.intel.com/en-us/mkl (2019)
13. Jiang, Z., Gao, W., Wang, L., Xiong, X., Zhang, Y., Wen, X., Luo, C., Ye, H., Lu, X., Zhang, Y., Feng, S., Li, K., Xu, W., Zhan, J.: Hpc ai500: A benchmark suite for hpc ai systems. 2018 BenchCouncil International Symposium on Benchmarking, Measuring and Optimizing (Bench18) (2018)
14. Li, G., Wang, X., Ma, X., Liu, L., Feng, X.: Xdn: Towards efficient inference of residual neural networks on cambricon chips. In: International Symposium on Benchmarking, Measuring and Optimization (Bench19). Springer (2019)
15. Li, Z., Jia, H., Zhang, Y., Chen, T., Yuan, L., Cao, L., Wang, X.: Autofft: a template-based fft codes auto-generation framework for arm and x86 cpus. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. p. 25. ACM (2019)
16. Linden, G., Smith, B., York, J.: Amazon. com recommendations: Item-to-item collaborative filtering. IEEE Internet computing (1), 76–80 (2003)
17. Luo, C., Zhang, F., Huang, C., Xiong, X., Chen, J., Wang, L., Gao, W., Ye, H., Wu, T., Zhou, R., Zhan, J.: Aiot bench: Towards comprehensive benchmarking mobile and embedded device intelligence. 2018 BenchCouncil International Symposium on Benchmarking, Measuring and Optimizing (Bench18) (2018)
18. Makari Manshadi, F.: Scalable optimization algorithms for recommender systems (2014)

19. Ortega, F., Hernando, A., Bobadilla, J., Kang, J.H.: Recommending items to group of users using matrix factorization based collaborative filtering. Information Sciences **345**, 313–324 (2016)
20. Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted boltzmann machines for collaborative filtering. In: Proceedings of the 24th international conference on Machine learning. pp. 791–798. ACM (2007)
21. Singh, T., Rangarajan, S., John, D., Henrion, C., Southard, S., McIntyre, H., Novak, A., Kosonocky, S., Jotwani, R., Schaefer, A., et al.: Zen: A next-generation high-performance$\times$ 86 core. In: 2017 IEEE International Solid-State Circuits Conference (ISSCC). pp. 52–53. IEEE (2017)
22. Xianyi, Z., Qian, W., Chothia, Z.: Openblas: an optimized blas library. https://github.com/xianyi/OpenBLAS (2019)
23. Xiong, X., Wen, X., Huang, C.: Improving rgb-d face recognition via transfer learning from a pretrained 2d network. In: International Symposium on Benchmarking, Measuring and Optimization (Bench19). Springer (2019)
24. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: International conference on algorithmic applications in management. pp. 337–348. Springer (2008)