A High-Performance Implementation of Bayesian Matrix Factorization with Limited Communication

Tom Vander Aa[1], Xiangju Qin[2,3], Paul Blomstedt[2,4],
Roel Wuyts[1], Wilfried Verachtert[1], and Samuel Kaski[2]

[1] ExaScience Life Lab at imec, Leuven, Belgium
tom.vanderaa@imec.be
[2] Helsinki Institute for Information Technology (HIIT),
Department of Computer Science,
Aalto University, Finland
[3] University of Helsinki, Helsinki, Finland
[4] F-Secure, Helsinki, Finland

Abstract. Matrix factorization is a very common machine learning technique in recommender systems. Bayesian Matrix Factorization (BMF) algorithms would be attractive because of their ability to quantify uncertainty in their predictions and avoid over-fitting, combined with high prediction accuracy. However, they have not been widely used on large-scale data because of their prohibitive computational cost. In recent work, efforts have been made to reduce the cost, both by improving the scalability of the BMF algorithm as well as its implementation, but so far mainly separately. In this paper we show that the state-of-the-art of both approaches to scalability can be combined. We combine the recent highly-scalable Posterior Propagation algorithm for BMF, which parallelizes computation of blocks of the matrix, with a distributed BMF implementation that users asynchronous communication within each block. We show that the combination of the two methods gives substantial improvements in the scalability of BMF on web-scale datasets, when the goal is to reduce the wall-clock time.

1 Introduction

Matrix Factorization (MF) is a core machine learning technique for applications of collaborative filtering, such as recommender systems or drug discovery, where a data matrix $\mathbf{R}$ is factorized into a product of two matrices, such that $\mathbf{R} \approx \mathbf{U}\mathbf{V}^\top$. The main task in such applications is to predict unobserved elements of a partially observed data matrix. In recommender systems, the elements of $\mathbf{R}$ are often ratings given by users to items, while in drug discovery they typically represent bioactivities between chemical compounds and protein targets or cell lines.

Bayesian Matrix Factorization (BMF) [2,13], formulates the matrix factorization task as a probabilistic model, with Bayesian inference conducted on the

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50433-5_1

unknown matrices $\mathbf{U}$ and $\mathbf{V}$. Advantages often associated with BMF include robustness to over-fitting and improved predictive accuracy, as well as flexible utilization of prior knowledge and side-data. Finally, for application domains such as drug discovery, the ability of the Bayesian approach to quantify uncertainty in predictions is of crucial importance [9].

Despite the appeal and many advantages of BMF, scaling up the posterior inference for industry-scale problems has proven difficult. Scaling up to this level requires both data and computations to be distributed over many workers, and so far only very few distributed implementations of BMF have been presented in the literature. In [16], a high-performance computing implementation of BMF using Gibbs sampling for distributed systems was proposed. The authors considered three different distributed programming models: Message Passing Interface (MPI), Global Address Space Programming Interface (GASPI) and ExaSHARK. In a different line of work, [1] proposed to use a distributed version of the minibatch-based Stochastic Gradient Langevin Dynamics algorithm for posterior inference in BMF models. While a key factor in devising efficient distributed solutions is to be able to minimize communication between worker nodes, both of the above solutions require some degree of communication in between iterations.

A recent promising proposal, aiming at minimizing communication and thus reaching a solution in a faster way, is to use a hierarchical embarrassingly parallel MCMC strategy [12]. This technique, called BMF with Posterior Propagation (BMF-PP), enhances regular embarrassingly parallel MCMC (e.g. [10,17]), which does not work well for matrix factorization [12] because of identifiability issues. BMF-PP introduces communication at predetermined limited phases in the algorithm to make the problem identifiable, effectively building one model for all the parallelized data subsets, while in previous works multiple independent solutions were found per subset.

The current paper is based on a realization that the approaches of [16] and [12] are compatible, and in fact synergistic. BMF-PP will be able to parallelize a massive matrix but will be the more accurate the larger the parallelized blocks are. Now replacing the earlier serial processing of the blocks by the distributed BMF in [16] will allow making the blocks larger up to the scalability limit of the distributed BMF, and hence decrease the wall-clock time by engaging more processors.

The main contributions of this work are:

- We combine both approaches, allowing for parallelization both at the algorithmic and at the implementation level.
- We analyze what is the best way to subdivide the original matrix into subsets, taking into account both compute performance *and* model quality.
- We examine several web-scale datasets and show that datasets with different properties (like the number of non-zero items per row) require different parallelization strategies.

The rest of this paper is organized as follows. In Section 2 we present the existing Posterior Propagation algorithm and distributed BMF implementation and we explain how to combine both. Section 3 is the main section of this paper, where we document the experimental setup and used dataset, we compare with related MF methods, and present the results both from a machine learning point of view, as from a high-performance compute point of view. In Section 4 we draw conclusions and propose future work.

2 Distributed Bayesian Matrix Factorization with Posterior Propagation

In this section, we first briefly review the BMF model and then describe the individual aspects of distributed computation and Posterior Propagation and how to combine them.

2.1 Bayesian Matrix Factorization

In matrix factorization, a (typically very sparsely observed) data matrix $\mathbf{R} \in \mathbb{R}^{N\times D}$ is factorized into a product of two matrices $\mathbf{U} \in \mathbb{R}^{N\times K} = (\mathbf{u}_1, \dots, \mathbf{u}_N)^\top$ and $\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_D)^\top \in \mathbb{R}^{D\times K}$. In the context of recommender systems, $\mathbf{R}$ is a rating matrix, with N the number of users and D the number of rated items.

In Bayesian matrix factorization [2,13], the data are modelled as

$$p(\mathbf{R}|\mathbf{U},\mathbf{V}) = \prod_{n=1}^{N}\prod_{d=1}^{D}\left[\mathcal{N}\left(r_{nd}|\mathbf{u}_n^\top \mathbf{v}_d, \tau^{-1}\right)\right]^{I_{nd}} \tag{1}$$

where I_{nd} denotes an indicator which equals 1 if the element r_{nd} is observed and 0 otherwise, and τ denotes the residual noise precision. The two parameter matrices $\mathbf{U}$ and $\mathbf{V}$ are assigned Gaussian priors. Our goal is then to compute the joint posterior density $p(\mathbf{U},\mathbf{V}|\mathbf{R}) \propto p(\mathbf{U})p(\mathbf{V})p(\mathbf{R}|\mathbf{U},\mathbf{V})$, conditional on the observed data. Posterior inference is typically done using Gibbs sampling, see [13] for details.

2.2 Bayesian Matrix Factorization with Posterior Propagation

In the *Posterior Propagation* (PP) framework [12], we start by partitioning $\mathbf{R}$ with respect to both rows and columns into $I \times J$ subsets $\mathbf{R}^{(i,j)}$, $i = 1,\dots,I$, $j = 1,\dots,J$. The parameter matrices $\mathbf{U}$ and $\mathbf{V}$ are correspondingly partitioned into I and J submatrices, respectively. The basic idea of PP is to process each subset using a hierarchical embarrassingly parallel MCMC scheme in three phases, where the posteriors from each phase are propagated forwards and used as priors in the following phase, thus introducing dependencies between the subsets. The approach proceeds as follows (for an illustration, see Figure 1):

Phase (a): Joint inference for submatrices $(\mathbf{U}^{(1)}, \mathbf{V}^{(1)})$, conditional on data subset $\mathbf{R}^{(1,1)}$:

$$p\left(\mathbf{U}^{(1)},\mathbf{V}^{(1)}|\mathbf{R}^{(1,1)}\right) \propto p\left(\mathbf{U}^{(1)}\right)p\left(\mathbf{V}^{(1)}\right)p\left(\mathbf{R}^{(1,1)}|\mathbf{U}^{(1)},\mathbf{V}^{(1)}\right).$$

Phase (b): Joint inference in parallel for submatrices $(\mathbf{U}^{(i)}, \mathbf{V}^{(1)})$, $i = 2, \ldots, I$, and $(\mathbf{U}^{(1)}, \mathbf{V}^{(j)})$, $j = 2, \ldots, J$, conditional on data subsets which share columns or rows with $\mathbf{R}^{(1,1)}$, and using posterior marginals from phase (a) as priors:

$$p\left(\mathbf{U}^{(i)}, \mathbf{V}^{(1)} | \mathbf{R}^{(1,1)}, \mathbf{R}^{(i,1)}\right) \propto p\left(\mathbf{V}^{(1)} | \mathbf{R}^{(1,1)}\right) p\left(\mathbf{U}^{(i)}\right) p\left(\mathbf{R}^{(i,1)} | \mathbf{U}^{(i)}, \mathbf{V}^{(1)}\right),$$
$$p\left(\mathbf{U}^{(1)}, \mathbf{V}^{(j)} | \mathbf{R}^{(1,1)}, \mathbf{R}^{(1,j)}\right) \propto p\left(\mathbf{U}^{(1)} | \mathbf{R}^{(1,1)}\right) p\left(\mathbf{V}^{(j)}\right) p\left(\mathbf{R}^{(1,j)} | \mathbf{U}^{(1)}, \mathbf{V}^{(j)}\right).$$

Phase (c): Joint inference in parallel for submatrices $(\mathbf{U}^{(i)}, \mathbf{V}^{(j)})$, $i = 2, \ldots, I$, $j = 2, \ldots, J$, conditional on the remaining data subsets, and using posterior marginals propagated from phase (b) as priors:

$$p\left(\mathbf{U}^{(i)}, \mathbf{V}^{(j)} | \mathbf{R}^{(1,1)}, \mathbf{R}^{(i,1)}, \mathbf{R}^{(1,j)}, \mathbf{R}^{(i,j)}\right)$$
$$\propto p\left(\mathbf{U}^{(i)} | \mathbf{R}^{(1,1)}, \mathbf{R}^{(i,1)}\right) p\left(\mathbf{V}^{(j)} | \mathbf{R}^{(1,1)}, \mathbf{R}^{(1,j)}\right) p\left(\mathbf{R}^{(i,j)} | \mathbf{U}^{(i)}, \mathbf{V}^{(j)}\right).$$

Finally, the aggregated posterior is obtained by combining the posteriors obtained in phases (a)-(c) and dividing away the multiply-counted propagated posterior marginals; see [12] for details.

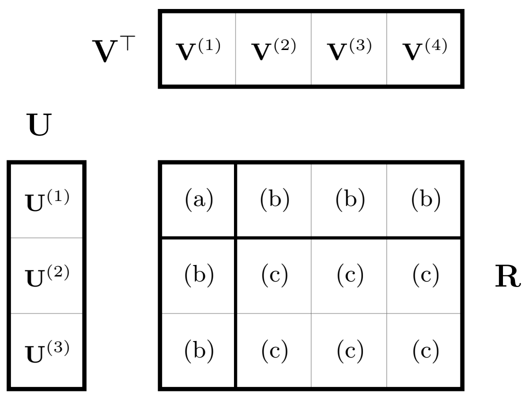


Fig. 1: Illustration of Posterior Propagation (PP) for a data matrix $\mathbf{R}$ partitioned into 3×4 subsets. Subset inferences for the matrices $\mathbf{U}$ and $\mathbf{V}$ proceed in three successive phases, with posteriors obtained in one phase being propagated as priors to the next one. The letters (a), (b), (c) in the matrix $\mathbf{R}$ refer to the phase in which the particular data subset is processed. Within each phase, the subsets are processed in parallel with no communication. Figure adapted from [12].

2.3 Distributed Bayesian Matrix Factorization

In [16] a distributed parallel implementation of BMF is proposed. In that paper an implementation the BMF algorithm [13] is proposed that distributes the rows and columns of **R** across different nodes of a supercomputing system.

Since Gibbs sampling is used, rows of **U** and rows of **V** are independent and can be sampled in parallel, on different nodes. However, there is a dependency between samples of **U** and **V**. The communication pattern between **U** and **V** is shown in Figure 2.

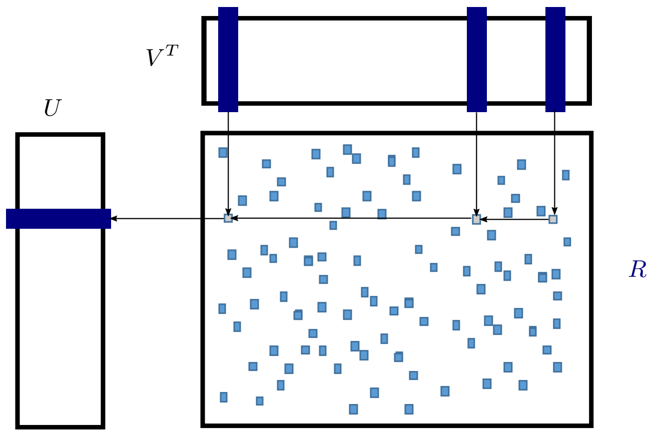


Fig. 2: Communication pattern in the distributed implementation of BMF

The main contribution of this implementation is how to distribute **U** and **V** to make sure the computational load is distributed as equally as possible and the amount of data communication is minimized. The authors of [16] optimize the distributions by analysing the sparsity structure of **R**.

As presented in the paper, distributed BMF provides a reasonable speed-up for compute systems up to 128 nodes. After this, speed-up in the strong scaling case is limited by the increase in communication, while at the same computations for each node decrease.

2.4 Distributed Bayesian Matrix Factorization with Posterior Propagation

By combining the distributed BMF implementation with PP, we can exploit the parallelism at both levels. The different subsets of **R** in the parallel phases of PP are independent and can thus be computed on in parallel on different resources. Inside each subset we exploit parallelism by using the distributed BMF implementation with MPI communication.

3 Experiments

In this section we evaluate the distributed BMF implementation with Posterior Propagation (D-BMF+PP) and competitors for benchmark datasets. We first have a look at the datasets we will use for the experiments. Next, we compare our proposed D-BMF+PP implementation with other related matrix factorization methods in terms of accuracy and runtime, with a special emphasis on the benefits of Bayesian methods. Finally we look at the strong scaling of D-BMF+PP and what is a good way to split the matrix in to blocks for Posterior Propagation.

3.1 Datasets

Table 1: Statistics about benchmark datasets.

Name	Movielens [5]	Netflix [4]	Yahoo [3]	Amazon [8]
Scale	1-5	1-5	0-100	1-5
# Rows	138.5K	480.2K	1.0M	21.2M
# Columns	27.3K	17.8K	625.0K	9.7M
# Ratings	20.0M	100.5M	262.8M	82.5M
Sparsity	189	85	2.4K	2.4M
Ratings/Row	144	209	263	4
#Rows/#Cols	5.1	27.0	1.6	2.2
K	10	100	100	10
rows/sec ($\times$1000)	416	15	27	911
ratings/sec ($\times 10^6$)	70	5.5	5.2	3.8

Table 1 shows the different webscale datasets we used for our experiments. The table shows we have a wide diversity of properties: from the relatively small Movielens dataset with 20M ratings, to the 262M ratings Yahoo dataset. Sparsity is expressed as the ratio of the total number of elements in the matrix (#rows $\times$ #columns) to the number of filled-in ratings (#ratings). This metric also varies: especially the Amazon dataset is very sparsely filled.

The scale of the ratings (either 1 to 5 or 0 to 100) is only important when looking at prediction error values, which we will do in the next section.

The final three lines of the table are not data properties, but rather properties of the matrix factorization methods. K is the number of latent dimensions used, which is different per dataset but chosen to be common to all matrix factorization methods. The last two lines are two compute performance metrics, which will be discussed in Section 3.4.

3.2 Related Methods

In this section we compare the proposed BMF+PP method to other matrix factorization (MF) methods, in terms of accuracy (Root-Mean-Square Error or RMSE on the test set), and in terms of compute performance (wall-clock time).

Stochastic gradient decent (SGD [15]), Coordinate Gradient Descent (CGD [18]) and Alternating Least Squares (ALS [14]) are the three most-used algorithms for non-Bayesian MF.

CGD- and ALS-based algorithms update along one dimension at a time while the other dimension of the matrix remains fixed. Many variants of ALS and CGD exist that improve the convergence [6], or the parallelization degree [11], or are optimized for non-sparse rating matrices [7]. In this paper we limit ourselves to comparing to methods, which divide the rating matrix into blocks, which is necessary to support very large matrices. Of the previous methods, this includes the SGD-based ones.

FPSGD [15] is a very efficient SGD-based library for matrix factorization on multi-cores. While FPSGD is a single-machine implementation, which limits its capability to solve large-scale problems, it has outperformed all other methods on a machine with up to 16 cores. NOMAD [19] extends the idea of block partitioning, adding the capability to release a portion of a block to another thread before its full completion. It performs similarly to FPSGD on a single machine, and can scale out to a 64-node HPC cluster.

Table 2: RMSE of different matrix factorization methods on benchmark datasets.

Dataset	BMF+PP	NOMAD	FPSGD
Movielens	0.76	0.77	0.77
Netflix	0.90	0.91	0.92
Yahoo	21.79	21.91	21.78
Amazon	1.13	1.20	1.15

Table 2 compares the RMSE values on the test sets of the four selected datasets. For all methods, we used the K provided in Table 1. For competitor methods, we used the default hyperparameters suggested by the authors for the different data sets. As already concluded in [12], BMF with Posterior Propagation results in equally good RMSE compared to the original BMF. We also see from the table that on average the Bayesian method produces only slightly better results, in terms of RMSE. However we do know that Bayesian methods have significant statistical advantages [9] that stimulate their use.

For many applications, this advantage outweighs the much higher computational cost. Indeed, as can be seem from Table 3, BMF is significantly more expensive, than NOMAD and FPSGD, even when taking in to account the speed-ups thanks to using PP. NOMAD is the fastest method, thanks to the aforementioned improvements compared to FPSGD.

Table 3: Wall-clock time (hh:mm) of different matrix factorization methods on benchmark datasets, running on single-node system with 16 cores.

Dataset	BMF+PP	BMF	NOMAD	FPSGD
Movielens	0:07	0:14	0:08	0:09
Netflix	2:02	4:39	0:08	1:04
Yahoo	2:13	12:22	0:10	2:41
Amazon	4:15	13:02	0:40	2:28

3.3 Block Size

The combined method will achieve some of the parallelization with the PP algorithm, and some with the distributed BMF within each block. We next compare the performance as the share between the two is varied by varying the block size. We find that blocks should be approximately square, meaning the number of rows and columns inside the each block should be more or less equal. This implies the number of blocks across the rows of the **R** matrix will be less if the **R** matrix has fewer rows and vice versa.

Figure 3 shows optimizing the block size is crucial to achieve a good speed up with BMF+PP and avoid compromising the quality of the model (as measured with the RMSE). The block size in the figure, listed as $I \times J$, means the R matrix is split in I blocks (with equal amount of rows) in the vertical direction and J blocks (with equal amount of columns) in the horizontal direction. The figure explores different block sizes for the Netflix dataset, which has significantly more rows than columns (27×) as can be seen from Table 1. This is reflected by the fact that the data point with the smallest bubble area (20×3) provides the best trade-off between wall-clock time and RMSE.

We stipulate that the reason is the underlying trade-off between the amount of information in the block and the amount of compute per block. Both the amount of information and the amount of compute can optimized (maximized and minimized respectively) by making the blocks approximately squared, since both are proportionate to the ratio of the area versus the circumference of the block.

We will come back to this trade-off when we look at performance of BMF+PP when scaling to multiple nodes for the different datasets in Section 3.4.

3.4 Scaling

In this section we look at how the added parallelization of Posterior Propagation increases the strong scaling behavior of BMF+PP. Strong scaling means we look at the speed-up obtained by increasing the amount of compute nodes, while keeping the dataset constant.

The graphs in this section are displayed with a logarithmic scale on both axes. This has the effect that linear scaling (i.e. doubling of the amount of resources results in a halving of the runtime) is shown as a straight line . Additionally,

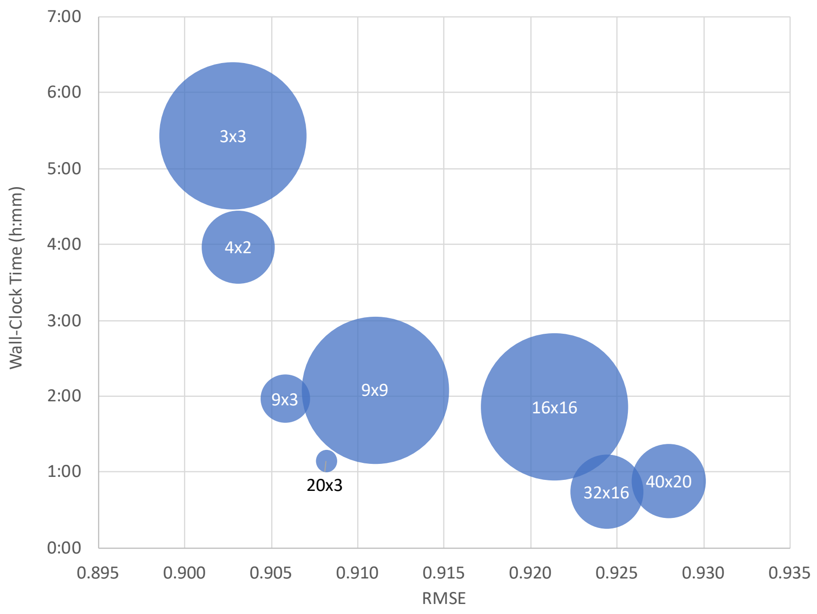


Fig. 3: Block size exploration: RMSE on test set and wall-clock time (hh:mm) for different block size on Netflix. Each bubble is the result for a different block size indicated as the number of blocks across the rows and the columns inside the bubble. The size of bubbles is an indication for the aspect ratio of the blocks inside PP. Smaller bubbles indicate the blocks are more square.

we indicate Pareto optimal solutions with a blue dot. For these solutions one cannot reduce the execution time without increasing the resources.

We performed experiments with different block sizes, indicated with different colors in the graphs. For example, the yellow line labeled 16×8 in Figure Figure 4 means we performed PP with 16 blocks for the rows of $\mathbf{R}$ and 8 blocks for the columns of $\mathbf{R}$.

We have explored scaling on a system with up to 128K nodes, and use this multi-node parallelism in two ways:

1. Parallelisation inside a block using the distributed version of BMF [16].
2. Parallelism across blocks. In phase (b) we can exploit parallelism across the blocks in the first row and first column, using up to $I+J$ nodes where I and J is the number of blocks in the vertical, respectively horizontal direction of the matrix. In phase (c) we can use up to $I \times J$ nodes.

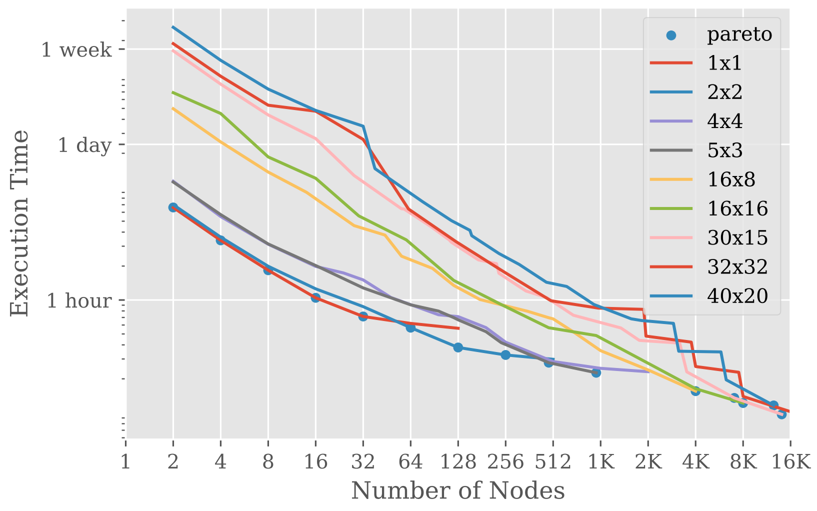
General Trends When looking at the four graphs (Figure 4 and 5), we observe that for the same amount of nodes using more blocks for posterior propagation, increases the wall-clock time. The main reason is that we do significantly more compute for this solution, because we take the same amount of samples for each sub-block. This means for a partitioning of 32×32 we take $1024\times$ more samples than 1×1.

On the other hand, we do get significant speedup, with up to $68\times$ improvement for the Netflix dataset. However the amount of resources we need for this is clearly prohibitively large (16K nodes). To reduce execution time, we plan to investigate reducing the amount of Gibbs samples and the effect this has on RMSE.

Netflix and Yahoo Runs on the Netflix and Yahoo datasets (Figure 4) have been performed with 100 latent dimensions (K=100). Since computational intensity (the amount of compute per row/column of $\mathbf{R}$), is $O(K^3)$, the amount of compute versus compute for these experiments is high, leading to good scalability for a single block (1×1) for Netflix, and almost linear scalability up to 16 or even 64 nodes. The Yahoo dataset is too large to run with a 1×1 block size, but we see a similar trend for 2×2.

Movielens and Amazon For Movielens and Amazon (Figure 5), we used $K = 10$ for our experiments. This implies a much smaller amount of compute for the same communication cost inside a block. Hence scaling for 1×1 is mostly flat. We do get significant performance improvements with more but smaller blocks where the Amazon dataset running on 2048 nodes, with 32×32 blocks is $20\times$ faster than the best block size (1×1) on a single node.

When the nodes in the experiment align with the parallelism across of blocks (either $I + J$ or $I \times J$ as explained above), we see a significant drop in the run time. For example for the Amazon dataset on 32×32 blocks going from 1024 to 2048 nodes.



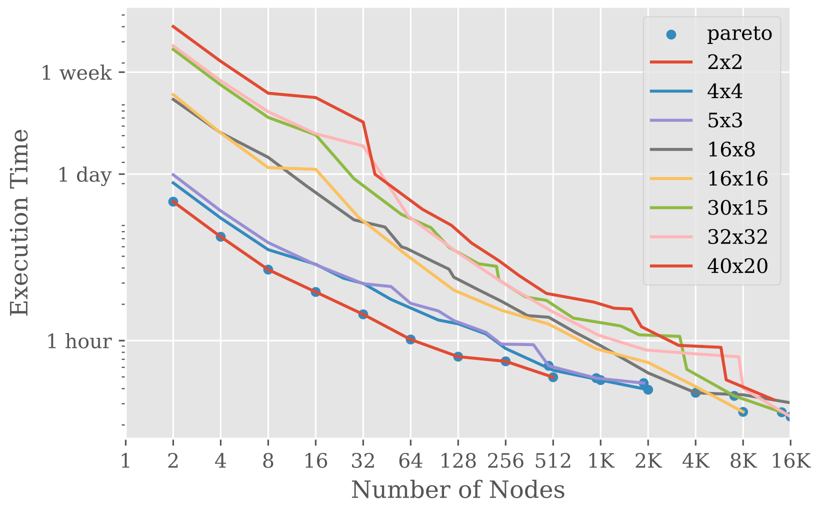
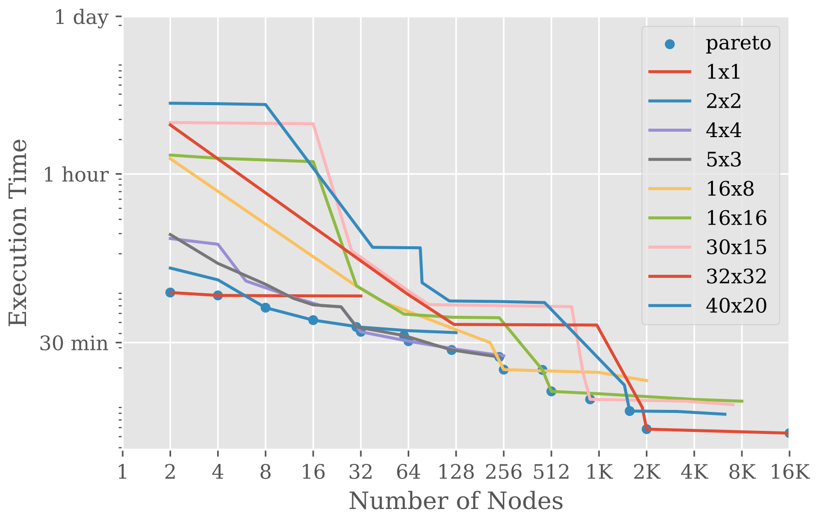


Fig. 4: Strong scaling results for the Netflix (top) and Yahoo (bottom) datasets. X-axis indicates the amount of compute resources (#nodes). Y-axis is wall-clock time. The different series correspond to different block sizes.



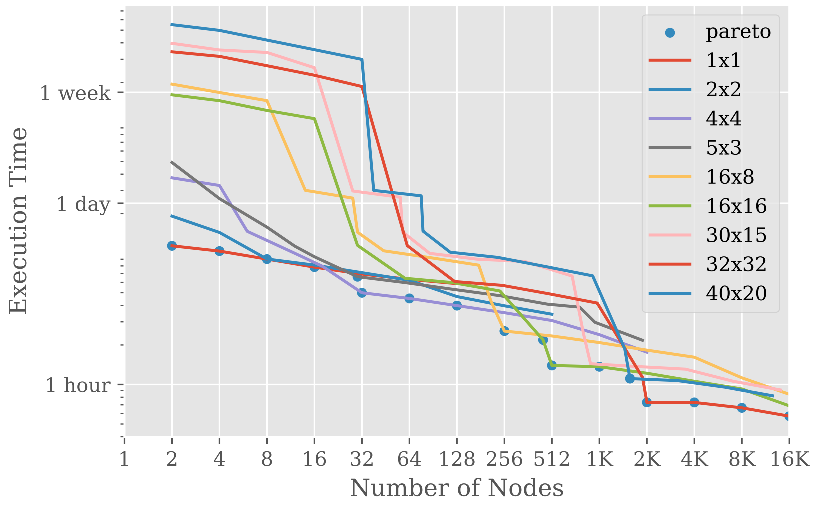


Fig. 5: Strong scaling results for the Movielens (top) and Amazon (bottom) datasets. X-axis indicates the amount of compute resources (#nodes). Y-axis is wall-clock time. The different series correspond to different block sizes.

4 Conclusions and Further Work

In this paper we presented a scalable, distributed implementation of Bayesian Probabilistic Matrix Factorization, using asynchronous communication. We evaluated both the machine learning and the compute performance on several web-scale datasets.

While we do get significant speed-ups, resource requirements for these speed-ups are extermely large. Hence, in future work, we plan to investigate the effect of the following measures on execution time, resource usage and RMSE:

- Reduce the number of samples for sub-blocks in phase (b) and phase (c).
- Use the posterior from phase (b) blocks to make predictions for phase (c) blocks.
- Use the GASPI implementation of [16], instead of the MPI implementation.

Acknowledgments

The research leading to these results has received funding from the European Union's Horizon2020 research and innovation programme under the EPEEC project, grant agreement No 801051. The work was also supported by the Academy of Finland (Flagship programme: Finnish Center for Artificial Intelligence, FCAI; grants 319264, 292334). We acknowledge PRACE for awarding us access to Hazel Hen at GCS@HLRS, Germany.

References

1. Ahn, S., Korattikara, A., Liu, N., Rajan, S., Welling, M.: Large-scale distributed Bayesian matrix factorization using stochastic gradient MCMC. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 9–18 (2015). https://doi.org/10.1145/2783258.2783373
2. Bhattacharya, A., Dunson, D.B.: Sparse Bayesian infinite factor models. Biometrika **98**, 291–306 (2011)
3. Dror, G., Koenigstein, N., Koren, Y., Weimer, M.: The Yahoo! music dataset and KDD-Cup'11. In: Dror, G., Koren, Y., Weimer, M. (eds.) Proceedings of KDD Cup 2011. Proceedings of Machine Learning Research, vol. 18, pp. 3–18. PMLR (2012), http://proceedings.mlr.press/v18/dror12a.html
4. Gomez-Uribe, C.A., Hunt, N.: The netflix recommender system: Algorithms, business value, and innovation. ACM Trans. Manage. Inf. Syst. **6**(4), 13:1–13:19 (Dec 2015). https://doi.org/10.1145/2843948, http://doi.acm.org/10.1145/2843948
5. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. **5**(4), 19:1–19:19 (Dec 2015). https://doi.org/10.1145/2827872, http://doi.acm.org/10.1145/2827872
6. Hsieh, C.J., Dhillon, I.S.: Fast coordinate descent methods with variable selection for non-negative matrix factorization. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (2011)

7. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (Aug 2009). https://doi.org/10.1109/MC.2009.263
8. Lab, S.S.: Web data: Amazon reviews. https://snap.stanford.edu/data/web-Amazon.html, [Online; accessed 18-Aug-2018]
9. Labelle, C., Marinier, A., Lemieux, S.: Enhancing the drug discovery process: Bayesian inference for the analysis and comparison of dose–response experiments. Bioinformatics **35**(14), i464–i473 (07 2019). https://doi.org/10.1093/bioinformatics/btz335, https://doi.org/10.1093/bioinformatics/btz335
10. Neiswanger, W., Wang, C., Xing, E.P.: Asymptotically exact, embarrassingly parallel MCMC. In: Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence. pp. 623–632. UAI'14, AUAI Press, Arlington, Virginia, United States (2014), http://dl.acm.org/citation.cfm?id=3020751.3020816
11. Pilászy, I., Zibriczky, D., Tikk, D.: Fast als-based matrix factorization for explicit and implicit feedback datasets. In: Proceedings of the fourth ACM conference on Recommender systems. pp. 71–78. ACM (2010)
12. Qin, X., Blomstedt, P., Leppäaho, E., Parviainen, P., Kaski, S.: Distributed bayesian matrix factorization with limited communication. Machine Learning **108**(10), 1805–1830 (2019). https://doi.org/10.1007/s10994-019-05778-2, https://doi.org/10.1007/s10994-019-05778-2
13. Salakhutdinov, R., Mnih, A.: Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In: Proceedings of the 25th International Conference on Machine Learning. pp. 880–887. ACM (2008). https://doi.org/10.1145/1390156.1390267
14. Tan, W., Cao, L., Fong, L.: Faster and cheaper: Parallelizing large-scale matrix factorization on gpus. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC'16). pp. 219–230. ACM (2016). https://doi.org/10.1145/2907294.2907297
15. Teflioudi, C., Makari, F., Gemulla, R.: Distributed matrix completion. In: Proceedings of the 2012 IEEE 12th International Conference on Data Mining (ICDM '12). pp. 655–664 (2012). https://doi.org/10.1109/ICDM.2012.120
16. Vander Aa, T., Chakroun, I., Haber, T.: Distributed Bayesian probabilistic matrix factorization. Procedia Computer Science **108**, 1030 – 1039 (2017), international Conference on Computational Science, ICCS 2017
17. Wang, X., Guo, F., Heller, K.A., Dunson, D.B.: Parallelizing MCMC with random partition trees. In: Advances in Neural Information Processing Systems 28, pp. 451–459. Curran Associates, Inc. (2015)
18. Yu, H.F., Hsieh, C.J., Si, S., Dhillon, I.: Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In: Proceedings of the 2012 IEEE 12th International Conference on Data Mining (ICDM '12). pp. 765–774. IEEE Computer Society (2012). https://doi.org/10.1109/ICDM.2012.168, http://dx.doi.org/10.1109/ICDM.2012.168
19. Yun, H., Yu, H.F., Hsieh, C.J., Vishwanathan, S.V.N., Dhillon, I.: Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. Proc. VLDB Endow. **7**(11), 975–986 (2014). https://doi.org/10.14778/2732967.2732973, http://dx.doi.org/10.14778/2732967.2732973