

# Lawrence Berkeley National Laboratory

## LBL Publications

### Title

Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion

### Permalink

<https://escholarship.org/uc/item/9bd3d9rj>

### Journal

IEEE Transactions on Knowledge and Data Engineering, PP(99)

### ISSN

1041-4347

### Authors

Abubaker, Nabil  
Karsavuran, M Ozan  
Aykanat, Cevdet

### Publication Date

2023

### DOI

10.1109/tkde.2023.3253791

### Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NonCommercial License, available at <https://creativecommons.org/licenses/by-nc/4.0/>

Peer reviewed

# Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion

Nabil Abubaker, M. Ozan Karsavuran, and Cevdet Aykanat

**Abstract**—Stratified SGD (SSGD) is the primary approach for achieving serializable parallel SGD for matrix completion. State-of-the-art parallelizations of SSGD fail to scale due to large communication overhead. During an SGD epoch, these methods send data proportional to one of the dimensions of the rating matrix. We propose a framework for scalable SSGD through significantly reducing the communication overhead via exchanging point-to-point messages utilizing the sparsity of the rating matrix. We provide formulas to represent the essential communication for correctly performing parallel SSGD and we propose a dynamic programming algorithm for efficiently computing them to establish the point-to-point message schedules. This scheme, however, significantly increases the number of messages sent by a processor per epoch from  $\mathcal{O}(K)$  to  $\mathcal{O}(K^2)$  for a  $K$ -processor system which might limit the scalability. To remedy this, we propose a Hold-and-Combine strategy to limit the upper-bound on the number of messages sent per processor to  $\mathcal{O}(K \lg K)$ . We also propose a hypergraph partitioning model that correctly encapsulates reducing the communication volume. Experimental results show that the framework successfully achieves a scalable distributed SSGD through significantly reducing the communication overhead. Our code is publicly available at: [github.com/nfabubaker/CESSGD](https://github.com/nfabubaker/CESSGD)

**Index Terms**—Recommender Systems, Collaborative Filtering, Matrix Completion, SGD, HPC, Combinatorial algorithms, Hypergraph partitioning, Communication cost minimization, Latency cost, Bandwidth cost.

## 1 INTRODUCTION

RECOMMENDER systems are omnipresent in e-commerce as well as social, professional and academic networks. These systems help businesses improve profit by targeted advertisements to interested parties, facilitate the recruitment process by matching more relevant candidates to jobs, and help academics explore cross-disciplinary research works as well as expand their collaboration networks. Recommender systems can involve one or more techniques, among which Collaborative Filtering (CF) is the most widely used.

CF approaches recommend an *item* to a target *user* by using other users' ratings given that those other users and the target user have rated some other items similarly. The rating data produced nowadays, whether by social networks or e-commerce, is rather huge and change very often. Recommender systems for such huge data are usually implemented on distributed-memory systems that might involve multiple data centers. Therefore, the CF component should be performant and scalable to utilize the provided computational resources as well as the high-speed networks.

Low-rank matrix factorization have been successfully used in CF via revealing feature vectors that represent the users and the items (latent factors). In matrix-factorization-based CF methods, the known ratings are stored as a sparse matrix, rows of which represent users and columns of which represent items. The sparse matrix is factorized into two dense matrices representing the feature vectors of items and users, and these dense matrices are then used to

predict missing ratings in the original rating matrix. This use of matrix factorization is commonly referred to as *matrix completion*. The matrix factorization can be computed with different methods, including stochastic gradient descent (SGD), alternating least squares (ALS), cyclic coordinate descent (CCD) and more.

SGD is very efficient and usually achieves high completion accuracy compared to other methods [1]. However, given its sequential nature it has been a challenge to efficiently parallelize while maintaining accuracy and convergence guarantee. For this reason, serializable parallel SGD algorithms are most desired. Serializability of parallel SGD refers to the existence of an equivalent serially-executed SGD algorithm with the same update order. Serializability guarantees the convergence and assures that no two processors update the same feature vector at the same time (race condition) thus leading to faster convergence [2]. Stratified SGD [3] is the de-facto algorithm for achieving a serializable parallel SGD.

The state-of-the-art methods implementing SSGD (such as DSGD [3], DSGD++ [4], and NOMAD [5]) achieve the inter-processor communication necessary for the correctness of the SSGD through sending/receiving feature vectors with sizes proportional to one of the dimensions of the input rating matrix. In other words, these methods perform dense communications without exploiting the sparse nature of the rating matrix, leading to a huge amount of unnecessary communication especially when the nonzero density of the rating matrix is low. The extra communication did not pose a concern because these methods are tested on a relatively small number of processors (up to 64) in distributed setting. At such small scale, the SGD runtime is expected to be dominated by computation; and investing in improving the communication component does not drastically affect the overall running time.

- Nabil and Cevdet are with the Department of Computer Engineering, Bilkent University, Turkey. M. Ozan is with Lawrence Berkeley National Laboratory, CA, USA. E-mails: [nabil.abubaker@bilkent.edu.tr](mailto:nabil.abubaker@bilkent.edu.tr), [mokarsavuran@lbl.gov](mailto:mokarsavuran@lbl.gov), [aykanat@cs.bilkent.edu.tr](mailto:aykanat@cs.bilkent.edu.tr)
- This work was done when M. Ozan was affiliated with Bilkent University.

On large scale (hundreds and thousands of processors), the communication component becomes dominant, and therefore reducing the communication overhead becomes essential for the scalability of the SSGD algorithm. The dense communications of the state-of-the-art methods implementing SSGD prohibit their scalability. This is empirically confirmed by us (Section 6) as well as by another work [6] that runs DSGD on thousands of processors (cf. Fig. 8 in [6]).

In this work, we propose a communication-efficient framework for the SSGD algorithm. Our framework starts with exploiting the sparsity of the rating matrix by performing sparse communications instead of dense communications. This is achieved with efficiently finding the essential feature vectors to be communicated between processors and communicating them through point-to-point (P2P) messages. This approach, although invaluable for reducing the volume of communication performed, has the down side of increasing the number of messages sent per processor from  $\mathcal{O}(K)$  (as in DSGD) to  $\mathcal{O}(K^2)$ .

Inter-processor communication cost ideally consists of latency term and bandwidth term. The latency term is proportional to the number of messages sent, whereas the bandwidth term is proportional to the volume of data transferred. If the number of messages is high, the latency cost might dominate the overall communication component since each message’s startup time might be higher than that of sending a few kilobytes of data [7]. The  $\mathcal{O}(K^2)$  bound of the new sparse communication method has the potential to increase the latency overhead and possibly affecting the scalability as  $K$  increases, which makes it *latency-unsafe*. To remedy this, we propose a novel approach called hold and combine that reduces the upper bound on the number of messages from  $\mathcal{O}(K^2)$  to  $\mathcal{O}(K \lg K)$  which renders the new sparse communication method *latency-safe*.

The volume of the sparse communication in parallel SSGD is also affected by how the ratings are distributed to different processors. This property indicates that there is a room for reducing the volume of communication combinatorially via intelligent partitioning methods. We propose a partitioning method utilizing a hypergraph partitioning model that correctly encapsulates the total volume of communication between processors. In this method, the objective of reducing the cutsize of the hypergraph model partition also corresponds to reducing the total volume of communication in an SSGD epoch.

The rest of the paper is organized as follows: Section 2 gives the essentials of using and parallelizing SGD for matrix completion. In Section 3, the communication requirement in parallel SSGD is studied in detail. In Section 4, the proposed framework for scaling P2P SSGD, including the hold-and-combine scheme, is presented. In Section 5, the proposed hypergraph partitioning (HP) method is presented. Section 6 contains the experiments conducted on an HPC system along with the results and discussions. Related works are discussed in Section 7 and the paper is concluded in Section 8.

## 2 BACKGROUND

### 2.1 Matrix Completion with SGD

We define the matrix completion problem in the context of collaborative filtering as follows: Given a set  $\mathcal{U}$  of  $N$  users, a

set  $\mathcal{I}$  of  $M$  items, and a set  $\Omega$  of ratings as the known entries of a sparse rating matrix  $\mathbf{R} \in \mathbb{R}^{N \times M}$ . The problem is to find two dense factor matrices  $\mathbf{W} \in \mathbb{R}^{N \times F}$  and  $\mathbf{H} \in \mathbb{R}^{M \times F}$  such that a low-rank approximation  $\mathbf{R} \approx \mathbf{W}\mathbf{H}^\top$  is achieved. Here,  $F \ll M, N$  is called the dimension or the rank of the factorization. The approximation  $\hat{r}_{ij}$  of rating  $r_{ij}$  can then be calculated as

$$\hat{r}_{ij} = \mathbf{w}_i \mathbf{h}_j^\top, \quad (1)$$

where  $\mathbf{w}_i$  and  $\mathbf{h}_j$  respectively denote the  $i$ th row of  $\mathbf{W}$  and the  $j$ th row of  $\mathbf{H}$ . The quality of the approximation is usually measured by an application-dependent loss function  $\mathcal{L}$ , thus the problem becomes  $\operatorname{argmin}_{\mathbf{W}, \mathbf{H}} \mathcal{L}(\mathbf{R}, \mathbf{W}, \mathbf{H})$ . For collaborative filtering,  $\mathcal{L}$  is usually the euclidean distance and thus the problem becomes

$$\operatorname{argmin}_{\mathbf{W}, \mathbf{H}} \sum_{(i,j) \ni r_{ij} \in \Omega} ((r_{ij} - \hat{r}_{ij})^2 + \gamma(\|\mathbf{w}_i\|^2 + \|\mathbf{h}_j\|^2)), \quad (2)$$

where  $\gamma$  is a regularization parameter to avoid over-fitting, and  $\hat{r}_{ij}$  is computed with (1).

Since the minimization problem in (2) has two unknowns  $\mathbf{W}$  and  $\mathbf{H}$ ,  $\mathcal{L}$  is a non-convex function [8]. SGD has been widely used to optimize (minimize) such functions due to its ability to escape local minimas. At an SGD epoch, each rating  $r_{ij} \in \Omega$  is used to update the objective function’s parameters. The gradient of the objective function at point  $r_{ij}$  is calculated ( $\nabla_{r_{ij}} \mathcal{L}_{r_{ij}}(\mathbf{R}, \mathbf{W}, \mathbf{H})$ ) and the corresponding  $\mathbf{w}_i$  and  $\mathbf{h}_j$  rows are updated as

$$\mathbf{w}_i = \mathbf{w}_i - \epsilon[(r_{ij} - \hat{r}_{ij})\mathbf{h}_j + \gamma\mathbf{w}_i], \quad (3)$$

$$\mathbf{h}_j = \mathbf{h}_j - \epsilon[(r_{ij} - \hat{r}_{ij})\mathbf{w}_i + \gamma\mathbf{h}_j] \quad (4)$$

where  $\epsilon$  is the step size.

It is clear from (3) and (4) that SGD is sequential in nature, thus parallelizing it requires communicating up-to-date  $\mathbf{W}$ - and  $\mathbf{H}$ -matrix rows. Trivially, the up-to-date  $\mathbf{W}$ - and  $\mathbf{H}$ -matrix rows should be communicated after each SGD update which enforces very high communication and synchronization overheads. Otherwise, some SGD updates will be performed on stale versions of  $\mathbf{W}$ - and  $\mathbf{H}$ -matrix rows which may drastically affect the learning process and the convergence guarantee. The parallel SGD methods that allow updating on stale  $\mathbf{W}$ - and  $\mathbf{H}$ -matrix rows (i.e., allow staleness) are called *asynchronous*. These methods are usually non-serializable. Simple parallelizations of the SGD-based matrix completion, such as row-wise or column-wise partitioning of the rating matrix, are examples of asynchronous SGD (see Fig. 1).

## 2.2 Stratified SGD (SSGD) and Its Parallelization

### 2.2.1 SSGD

The SSGD method is proposed by Gemulla et al. [3] in order to mitigate the staleness problem. In SSGD, the rating matrix is divided into  $K^2$  2D blocks using  $K$ -way mutually exclusive and exhaustive partitions on the rows  $\Pi^R = \{R_1, \dots, R_K\}$  and columns  $\Pi^C = \{C_1, \dots, C_K\}$  of  $\mathbf{R}$ . The rows of the dense matrices  $\mathbf{W}$  and  $\mathbf{H}$  are partitioned conformably with  $\Pi^R$  and  $\Pi^C$ , respectively. We denote the row blocks of  $\mathbf{W}$  and  $\mathbf{H}$  that respectively conform with  $R_\alpha$  and  $C_\beta$  as  $\mathbf{W}_\alpha$  and  $\mathbf{H}_\beta$ . We denote a block of  $\mathbf{R}$  with rows in  $R_\alpha$  and columns in  $C_\beta$  as  $\mathbf{R}_{\alpha\beta}$ .

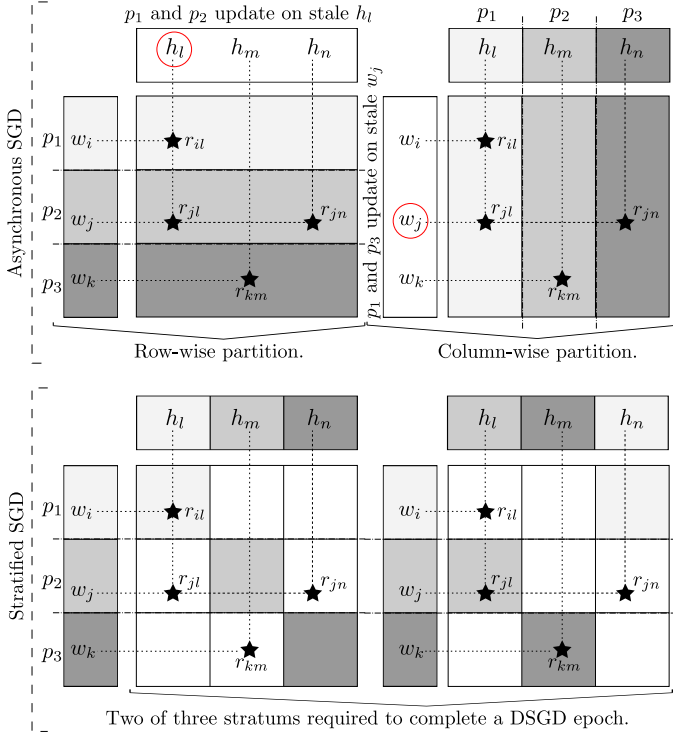


Fig. 1: Stale updates in simple row- or column-wise partitions (upper part) versus stale-free DSGD (bottom). In the row-wise partition of  $\mathbf{R}$ , the rows of  $\mathbf{W}$  are partitioned conformably and thus each  $\mathbf{W}$ -matrix row is accessed by one processor. However, this is not the case for  $\mathbf{H}$ -matrix rows. For instance, ratings  $r_{il}$  and  $r_{jl}$  are respectively assigned to  $p_1$  and  $p_2$  and both used to update  $h_l$  possibly at the same time thus either  $p_1$  or  $p_2$  will update on a stale  $h_l$ . A similar discussion holds for column-wise partition in a dual manner regarding  $r_{jl}$ ,  $r_{jn}$  and  $w_j$ . Black stars are known ratings.

In SSGD, a set of  $K$  2D non-overlapping sub-matrix blocks are called a *stratum* (denoted by  $\mathcal{S}$  hereafter). Two 2D sub-matrix blocks are said to be non-overlapping if they do not share any row or column. A set of  $K$  strata  $\mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_K\}$  that exhausts all of the  $K^2$  sub-matrix blocks is called correct strata. Fig. 2 shows the strata  $\mathcal{S} = \langle \mathcal{S}_1 = \{\mathbf{R}_{1,1}, \mathbf{R}_{2,2}, \dots, \mathbf{R}_{8,8}\}, \mathcal{S}_2 = \{\mathbf{R}_{1,2}, \mathbf{R}_{2,3}, \dots, \mathbf{R}_{8,1}\}, \dots, \mathcal{S}_8 = \{\mathbf{R}_{1,8}, \mathbf{R}_{2,1}, \dots, \mathbf{R}_{8,7}\} \rangle$ .

Given correct strata to be used in an SSGD epoch, each stratum is processed in a separate mini epoch (called sub-epoch), and the order in which these sub-epochs are executed can be random. Although the SSGD algorithm is serial, its distinguishing property is that no ratings in different blocks of a stratum can update the same row of the factor matrices  $\mathbf{W}$  and  $\mathbf{H}$ , which makes it suitable for stale-free parallelization.

## 2.2.2 Parallel SSGD

In [3], the parallel algorithm that utilizes SSGD is called the Distributed Stochastic Gradient Descent (DSGD) algorithm. In DSGD, each stratum is executed in parallel in one sub-epoch, where the  $\mathbf{W}$ - and  $\mathbf{H}$ -matrix rows are updated with the ratings in the stratum according to (3) and (4). Then, inter-processor communications are performed

to synchronize all updated rows of factor matrices. If a row-parallel execution is chosen, that is the  $\mathbf{R}$  matrix is partitioned row-wise such that each row block is executed by a single processor, then communication is restricted to the  $\mathbf{H}$ -matrix rows. Row-parallel execution is usually preferred because the number of users which means the amount of data to be communicated ( $\mathbf{H}$ ) is small compared to  $\mathbf{W}$ . In row-parallel execution, we abuse the stratum notation  $\mathcal{S}$  to also be viewed as a mapping function  $\mathcal{S} : [K] \rightarrow [K]$  (where  $[K]$  is used to denote the set  $\{1, \dots, K\}$  hereafter) from a processor  $p_k$  to the index  $\beta$  of a column block  $C_\beta$ . For instance,  $\mathcal{S}_2(p_4) = 5$  means that during sub-epoch 2, processor  $p_4$  will exclusively update the rows of the  $\mathbf{H}_5$  sub-matrix. We also use  $\mathcal{S}_\beta^{-1}(p_x)$  to retrieve the sub-epoch at which  $p_x$  updates  $\mathbf{H}_\beta$ . As mentioned previously in the introduction, DSGD performs dense communications. We will utilize the parallelization style of DSGD in our methods while changing how the communication is performed. Hereafter, we will refer to the parallelization style of DSGD as “parallel SSGD”, and we will use the name “DSGD” to distinguish the algorithm that performs dense communication.

## 2.2.3 Generating Correct Strata

There are several ways to generate a correct strata that covers the whole dataset and schedule the strata to sub-epochs. For simplicity, we consider a simple form of scheduling as follows: at sub-epoch 1, processor  $p_x$ , for  $x = 1, 2, \dots, K$ , processes the ratings in  $\mathbf{R}_{xx}$  to update the rows in  $\mathbf{W}_x$  and  $\mathbf{H}_x$ ; at sub-epoch  $k$ , processor  $p_x$  processes the ratings in  $\mathbf{R}_{x\beta}$  to update the rows in  $\mathbf{W}_x$  and  $\mathbf{H}_\beta$ , where  $\beta = 1 + (x + k - 2) \bmod K$ . We refer to this scheduling as “ring scheduling” or “ring strata” hereafter. A general form of the ring scheduling consists of a *seed*, where  $1 \leq \text{seed} \leq K$ . At sub-epoch  $k$ , the processor  $p_{\text{seed}}$  processes the ratings in  $\mathbf{R}_{\text{seed},k}$  to update the rows in  $\mathbf{W}_{\text{seed}}$  and  $\mathbf{H}_k$ . At sub-epoch  $k$ , processor  $p_x$  processes the ratings in  $\mathbf{R}_{x\beta}$  to update the rows in  $\mathbf{W}_x$  and  $\mathbf{H}_\beta$ , where

$$\beta = 1 + (k + (\text{seed} + x - 1) \bmod K - 2) \bmod K. \quad (5)$$

## 3 COMMUNICATION IN PARALLEL SSGD

In this section, we analyze the communication requirement of parallel SSGD. We define the essential required communication in an SSGD epoch that utilizes the data sparsity, and compare it with the dense communication of DSGD.

Given strata  $\mathcal{S}$  where each stratum is to be processed in a sub-epoch in row-parallel execution. For an  $\mathbf{H}$ -matrix row block  $\mathbf{H}_\beta$ , we define the sequence

$$\Upsilon_\beta = \langle p_{i_1}, p_{i_2}, \dots, p_{i_K} \rangle$$

of processors that compute the gradient using ratings in the column block  $C_\beta$  according to  $\mathcal{S}$ . That is,  $p_{i_1}$  updates the rows of  $\mathbf{H}_\beta$  in the first sub-epoch,  $p_{i_2}$  in the second sub-epoch and so forth. Furthermore, we define a distance metric  $d_\beta^{xy}$  between two processors  $p_x$  and  $p_y$  updating  $\mathbf{H}_\beta$  as

$$d_\beta^{xy} = \mathcal{S}_\beta^{-1}(p_x) - \mathcal{S}_\beta^{-1}(p_y). \quad (6)$$

This distance quantifies the number of sub-epochs elapsed after  $p_x$  updates rows in  $\mathbf{H}_\beta$  and before  $p_y$  does so.

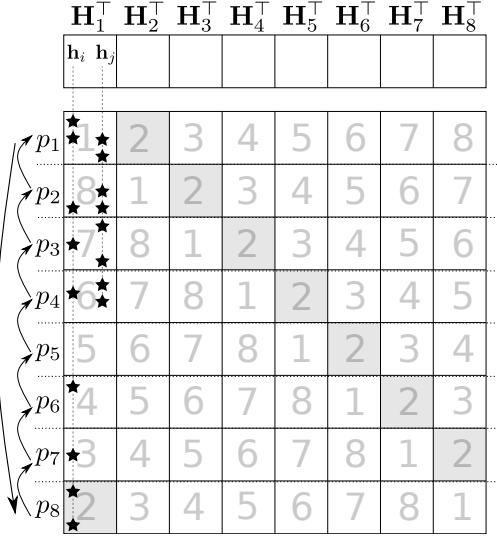


Fig. 2: The numbers identify the sub-matrix blocks that constitute a stratum in a ring strata with  $seed=1$ . Stratum  $S_2$  is highlighted. Side arrows show the processor update order of  $\mathbf{h}_i$  and  $\mathbf{h}_j$  in  $\mathbf{H}_1^\top$ .

### 3.1 Defining essential required communication

The communication of  $\mathbf{H}$ -matrix rows required for correctly executing SSGD in a distributed fashion is described according to the following definition:

**Definition 1** (*d-gap rows*). During parallel SSGD, if a row  $\mathbf{h}_j \in \mathbf{H}_\beta$  is updated by both  $p_{i_x}$  and  $p_{i_{x+1}}$ , then  $\mathbf{h}_j$  is called a zero-gap row. If  $\mathbf{h}_j$  is updated by both  $p_{i_x}$  and  $p_{i_{x+2}}$  but not  $p_{i_{x+1}}$ , then  $\mathbf{h}_j$  is called a one-gap row. Then for the general case, consider two nonadjacent processors in  $\Upsilon_\beta$ :  $p_{i_x}$  and  $p_{i_y}$  such that  $x < y$ .  $\mathbf{h}_j$  is called a  $d$ -gap row if it is updated by both  $p_{i_x}$  and  $p_{i_y}$  but not any of the  $d = d_\beta^{x,y}$  processors in-between (that is, in  $\langle p_{i_{x+1}}, \dots, p_{i_{y-1}} \rangle$ ). The set of all such  $d$ -gap rows between  $p_{i_x}$  and  $p_{i_y}$  in  $\mathbf{H}_\beta$  is given by

$$\mathbf{H}_\beta^{i_x i_y} = \{\mathbf{h}_j \mid (\exists r_{ij})[r_{ij} \in \mathbf{R}_{x\beta} \cap \mathbf{R}_{y\beta} \wedge r_{ij} \notin \mathbf{R}_{(x+1)\beta} \cup \dots \cup \mathbf{R}_{(y-1)\beta}]\}. \quad (7)$$

Communicating  $\mathbf{H}_\beta^{i_x i_y}$  from  $p_{i_x}$  to  $p_{i_y}$  after  $p_{i_x}$  processes the ratings in 2D block  $\mathbf{R}_{i_x\beta}$  and before  $p_{i_y}$  starts processing the ratings in  $\mathbf{R}_{i_y\beta}$  guarantees a correct distributed row-parallel SSGD execution.

### 3.2 Communication in DSGD

In the original DSGD algorithm [3], after processor  $p_x$  updates row block  $\mathbf{H}_\beta$  in sub-epoch  $k$ , it sends the rows in  $\mathbf{H}_\beta$  to the processor that will update  $\mathbf{H}_\beta$  in sub-epoch  $k+1$ . Therefore, at each sub-epoch, each processor sends a whole row block of  $\mathbf{H}$  to exactly one processor. For instance, assuming the DSGD is executed according to the ring strata given in Fig. 2, after sub-epoch 1 is completed  $p_1$  sends  $\mathbf{H}_1$  to  $p_8$ ,  $p_8$  sends  $\mathbf{H}_8$  to  $p_7$  and so forth.

The communication scheme of DSGD guarantees the correctness of the SSGD algorithm since up-to-date  $\mathbf{h}_j \in \mathbf{H}_\beta^{i_x i_y}$  will eventually reach  $p_{i_y}$  from  $p_{i_x}$ , assuming  $x < y$  in  $\Upsilon_\beta$ , via forwarding through  $\langle p_{i_{x+1}}, \dots, p_{i_{y-1}} \rangle$ . Furthermore, the

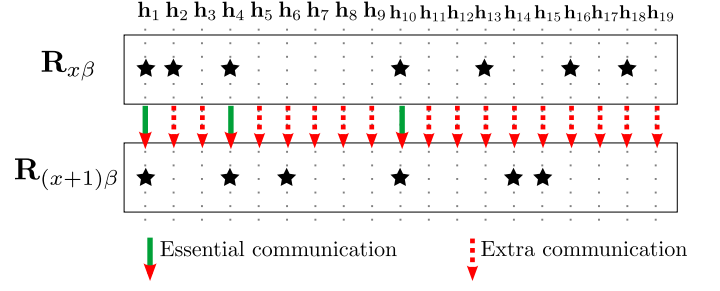


Fig. 3: Illustrating the extra communication of DSGD. The figure shows two blocks of  $\mathbf{R}$  that belong to processors  $p_x$  and  $p_{x+1}$  such that  $p_{x+1}$  updates column block  $\mathbf{H}_\beta$  right after  $p_x$ . A row  $\mathbf{h}_j \in \mathbf{H}_\beta$  has to be sent from  $p_x$  to  $p_{x+1}$  if both processors contain a nonzero with column index equal to  $\mathbf{h}_j$  because  $p_{x+1}$  has to know the up-to-date version of  $\mathbf{h}_j$  after  $p_x$  updates it. When either only one of  $p_x$  or  $p_{x+1}$  has such nonzero, or neither of them, the communication of  $\mathbf{h}_j$  at this stage is considered extra and can be avoided.

communication scheme of the DSGD has the nice property of very low latency overhead since it restricts the number of messages sent by any processor at any sub-epoch to one. However, this scheme suffers from increasing the bandwidth overhead (communication volume) due to forwarding the  $\mathbf{H}$ -matrix rows. For each epoch, the communication volume sent by all processors is equal to  $F \times M \times K$  words as each processor sends approximately  $M/K$  dense  $\mathbf{H}$ -matrix rows each of size  $F$  words during each of the  $K$  sub-epochs. Especially for highly sparse rating matrices, it is clear that the volume of communication performed is much more than the required, and the increased bandwidth overhead due to forwarding can be prohibitive as  $K$  increases, see Fig. 3.

In Fig. 2, the update sequence for row block  $\mathbf{H}_1$  is  $\Upsilon_1 = \langle p_1, p_8, p_7, p_6, p_5, p_4, p_3, p_2 \rangle$ . The communication of  $\mathbf{h}_i \in \mathbf{H}_1$  through the subsequence/subchain  $p_1 \rightarrow p_8 \rightarrow p_7 \rightarrow p_6$  does not incur any extra volume since each of these processors update  $\mathbf{h}_i$ . However,  $p_5$  does not update  $\mathbf{h}_i$  but still  $p_5$  needs to receive the up-to-date  $\mathbf{h}_i$  from  $p_6$  and forward it to  $p_4$  in the next sub-epoch. In this case,  $\mathbf{h}_i$  incurs  $F$  words of forwarding overhead. In the case of  $\mathbf{h}_j \in \mathbf{H}_1$ , the first processor to update it after  $p_1$  is  $p_4$ . Therefore, four forwarding communications, each of size  $F$ , are incurred due to  $\mathbf{h}_j$  in  $p_1 \rightarrow p_8 \rightarrow p_7 \rightarrow p_6 \rightarrow p_5$ .

Let  $\lambda(\mathbf{h}_j)$  denote the number of processors that update  $\mathbf{h}_j \in \mathbf{H}_\beta$ , then the amount of forwarding overhead of  $\mathbf{h}_j$  in DSGD is  $F(K - \lambda(\mathbf{h}_j))$ . The total amount of forwarding overhead per epoch then becomes  $F(MK - \sum_{\mathbf{h}_j \in \mathbf{H}} \lambda(\mathbf{h}_j))$ . The clear difference between the communication in DSGD and the essential required communication is that the former is a direct factor of  $K$  and  $M$ , whereas the latter is upper bounded by the number of nonzeros ( $nnz$ ) of the rating matrix. This can be shown as follows: at sub-epoch  $k$ , processor  $p_x$  sends  $nnz(\mathbf{R}_{x, S_k(p_x)})$   $\mathbf{H}$ -matrix rows at the worst case. This means that, at worst case, the total volume of communication sent by all processors per an SSGD epoch is equal to  $F \sum_{x \in [K]} \sum_{k \in [K]} nnz(\mathbf{R}_{x, S_k(p_x)}) = F \times nnz(\mathbf{R})$ .

## 4 A FRAMEWORK FOR SCALING SSGD

### 4.1 Communicating $d$ -gap rows through P2P messages

We propose to avoid the forwarding overhead by sending an updated  $\mathbf{H}$ -matrix row to the processor that updates it next directly through P2P communications. At the beginning of sub-epoch  $k$ , processor  $p_x$  sends P2P messages to a set of processors  $SendSet^k(p_x)$  and receives from  $RecvSet^k(p_x)$ . These two sets can be respectively constructed as

$$SendSet^k(p_x) = \{p_y \mid \mathbf{H}_{S_{k-1}(p_x)}^{xy} \neq \emptyset\}, \quad (8)$$

$$RecvSet^k(p_x) = \{p_y \mid \mathbf{H}_{S_k(p_x)}^{yx} \neq \emptyset\}. \quad (9)$$

For example, in Fig. 2, at the beginning of the second sub-epoch  $p_1$  sends  $\mathbf{h}_i$  to  $p_8$  and  $\mathbf{h}_j$  to  $p_4$ .

---

#### Algorithm 1 Point-to-Point parallel SSGD on processor $p_x$

---

**Require:** Rating matrix  $\mathbf{R}$ , Processor count  $K$

- 1: Initialize local factor matrices  $\mathbf{W}$  and  $\mathbf{H}$  randomly
- 2: **repeat**
- 3:   Receive strata  $S$  from  $p_1$  through  $B_{\text{cast}}$ .
- 4:   Construct P2P communication according to  $S$
- 5:   **for**  $k = 1$  to  $K$  **do** ▷ For each sub-epoch
- 6:      $\beta_{\text{prev}} \leftarrow S_{k-1}(p_x)$
- 7:      $\beta_{\text{curr}} \leftarrow S_k(p_x)$ ;
- 8:     **for** each  $p_y \in SendSet^k(p_x)$  **do**
- 9:       Send  $\mathbf{H}_{\beta_{\text{prev}}}^{xy}$  to  $p_y$
- 10:    **end for**
- 11:    **for** each  $p_z \in RecvSet^k(p_x)$  **do**
- 12:      Receive  $\mathbf{H}_{\beta_{\text{curr}}}^{zx}$  from  $p_z$
- 13:    **end for**
- 14:    **for** each  $r_{ij} \in \mathbf{R}_{x\beta_{\text{curr}}}$  **do**
- 15:       $\mathbf{w}_i = \mathbf{w}_i - \epsilon[(r_{ij} - \hat{r}_{ij})\mathbf{h}_j + \gamma\mathbf{w}_i]$
- 16:       $\mathbf{h}_j = \mathbf{h}_j - \epsilon[(r_{ij} - \hat{r}_{ij})\mathbf{w}_i + \gamma\mathbf{h}_j]$
- 17:    **end for**
- 18:    **end for**
- 19: **until** convergence or max. number of epochs reached

---

Algorithm 1 presents the P2P-based parallel SSGD algorithm for processor  $p_x$ . At line 3, processor  $p_1$  picks strata  $S$  and broadcasts to all other processors. At line 4,  $p_x$  determines the communication requirement according to (7) and constructs the send/receive information of the P2P messages according to (8) and (9). Then, the up-to-date rows required in the current sub-epoch are communicated at lines 8-13 through P2P messages. The SGD updates are performed at lines 15 and 16 respectively according to (3) and (4).

### 4.2 Efficiently constructing $d$ -gap row sets

Computing  $d$ -gap  $\mathbf{H}$ -matrix rows using (7) has recurring computations for different instances. For example, computing  $\mathbf{H}_{\beta}^{i_x i_y}$  and  $\mathbf{H}_{\beta}^{i_x i_{y+1}}$  would require computing the same  $\mathbf{R}_{(x+1)\beta} \cup \dots \cup \mathbf{R}_{(y-1)\beta}$  term twice. For an efficient computation, we devise an algorithm that utilizes a dynamic programming formulation leveraging efficient bulk bit-wise operations.

Consider a binary string  $B_{\beta}^{i_x} \in \{0, 1\}^{n_{\beta}}$  of length  $n_{\beta} = |\mathbf{H}_{\beta}|$ , such that the  $b$ th entry of  $B_{\beta}^{i_x}$  is set to ‘1’ if  $p_{i_x}$  updates the  $b$ th row in  $\mathbf{H}_{\beta}$ , and set to ‘0’ otherwise. Then,

---

#### Algorithm 2 Find $d$ -gap $\mathbf{H}$ -matrix rows on processor $p_x$

---

**Require:** Rating matrix  $\mathbf{R}$ , Processor count  $K$ , Strata  $S$

- 1: **for** each  $\mathbf{H}_{\beta} \in \mathbf{H}$  **do**
- 2:   Compute  $\Upsilon_{\beta}^{p_x}$
- 3:    $mask \leftarrow B_{\beta}^{\Upsilon_{\beta}^{p_x}[2]}$
- 4:   **for**  $k = 2$  to  $K$  **do**
- 5:      $p_y \leftarrow \Upsilon_{\beta}^{p_x}[k]$
- 6:      $\Psi_{\beta}^{p_x, p_y} \leftarrow B_{\beta}^x \wedge B_{\beta}^y \oplus (B_{\beta}^x \wedge B_{\beta}^y \wedge mask)$
- 7:      $\mathbf{H}_{\beta}^{xy} \leftarrow \{\mathbf{h}_i \in \mathbf{H}_{\beta} \mid \Psi_{\beta}^{p_x, p_y}[i] = 1\}$
- 8:      $mask \leftarrow mask \vee B_{\beta}^y$
- 9:   **end for**
- 10: **end for**

---

the indices of the rows to be communicated between  $p_{i_x}$  and  $p_{i_y}$  are the indices of the 1-bits in

$$\Psi_{\beta}^{i_x, i_y} = B_{\beta}^{i_x} \wedge B_{\beta}^{i_y} \oplus (B_{\beta}^{i_x} \wedge B_{\beta}^{i_y} \wedge (B_{\beta}^{i_{x+1}} \vee \dots \vee B_{\beta}^{i_{y-1}})), \quad (10)$$

where  $\oplus$ ,  $\wedge$  and  $\vee$  respectively denote logical exclusive OR (XOR), logical AND and logical OR operations. The term  $(B_{\beta}^{i_{x+1}} \vee \dots \vee B_{\beta}^{i_{y-1}})$  in (10) can be computed incrementally thanks to the associativity property of the  $\vee$  operation.

Given  $\mathbf{H}_{\beta}$  and  $\Upsilon_{\beta}$ , we define  $\Upsilon_{\beta}^{p_x}$  as the sequence of processors updating  $\mathbf{H}_{\beta}$  starting from  $p_x$ .  $\Upsilon_{\beta}^{p_x}$  can be obtained from  $\Upsilon_{\beta}$  by left-rotating the sequence until  $p_x$  is at the first index. Algorithm 2 presents the efficient dynamic-programming-based computation of the  $d$ -gap  $\mathbf{H}$ -matrix rows between  $p_x$  and the other  $K - 1$  processors. For each  $\mathbf{H}_{\beta}$ , the order of processors updating  $\mathbf{H}_{\beta}$  starting from  $p_x$  according to strata  $S$  is maintained in  $\Upsilon_{\beta}^{p_x}$  (line 3). Then, in lines 4-9,  $p_x$  constructs the  $d$ -gap rows one by one according to this order leveraging the bottom-up construction of the term  $(B_{\beta}^{i_{x+1}} \vee \dots \vee B_{\beta}^{i_{y-1}})$ .

### 4.3 Hold & Combine strategy for reducing latency

Using P2P messages to communicate the updated rows without forwarding is indispensable for reducing the bandwidth overhead of the communication. However, it has a high potential of increasing the latency overhead via increasing the number of messages performed per epoch compared to DSGD. In DSGD, a processor sends  $K$  messages per epoch (one message to one processor at each sub-epoch), whereas using the P2P requires sending at most  $K \times (K - 1)$  messages per epoch (up to  $K - 1$  messages from each of the  $K$  processors at each sub-epoch). We propose the hold and combine (H&C) strategy to reduce the upper-bound on the number of messages sent per epoch to  $\mathcal{O}(K \lg K)$ .

**Definition 2.** Fixed-distance strata is any strata that satisfies

$$d_{\alpha}^{xy} = d_{\beta}^{xy} \quad \text{for any pair of } \mathbf{H}\text{-matrix rows } \alpha \text{ and } \beta. \quad (11)$$

That is, the fixed-distance strata have the property of constant distance between any two processors regardless of the  $\mathbf{H}$ -matrix row block they are updating. We refer to the distance between two processors  $p_x$  and  $p_y$  in a fixed-distance strata as  $d^{xy}$ . Any ring strata scheduled with (5) is a fixed-distance strata.

During an SGD epoch, the communication of  $\mathbf{H}_{\beta}^{xy}$  should be performed after  $p_x$  updates  $\mathbf{H}_{\beta}$  in sub-epoch

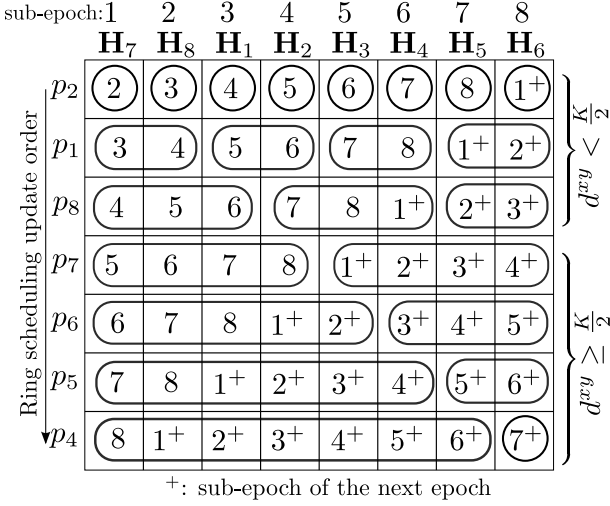


Fig. 4: An example TSMS for  $p_3$ . The rows are the processors that  $p_3$  communicates with sorted according to their distance from  $p_3$ . The columns represent both the sub-epochs and the  $\mathbf{H}$ -matrix blocks to be updated at each sub-epoch. An entry  $(p_y, \mathbf{H}_\beta)$  gives the sub-epoch at which  $p_y$  updates  $\mathbf{H}_\beta$  after  $p_3$  does (note that this sub-epoch might be in the next epoch). The circles show the messages that can be combined.

$k$  and before  $p_y$  starts updating  $\mathbf{H}_\beta$ . This means that  $\mathbf{H}_\beta^{xy}$  can be sent at the beginning of any sub-epoch between  $k+1$  and  $k+d^{xy}$ . Now consider the communication of  $\mathbf{H}_\beta^{xy}$  at sub-epoch  $k$  in a fixed-distance strata. Observe that when sub-epoch  $k+d^{xy}$  is reached, all the rows in  $\mathbf{H}_{S_{k+1}(p_x)}^{xy}, \mathbf{H}_{S_{k+2}(p_x)}^{xy}, \dots, \mathbf{H}_{S_{k+d^{xy}-1}(p_x)}^{xy}$  are already updated by  $p_x$  and ready to be sent to  $p_y$ . So, these rows can be held by  $p_x$  and sent all at once in one message to  $p_y$  in sub-epoch  $k+d^{xy}$  along with  $\mathbf{H}_\beta^{xy}$ .

Utilizing fixed-distance strata, we propose to hold P2P messages and combine them as follows: If  $d^{xy} \geq K/2$ , then the messages between  $p_x$  and  $p_y$  in an epoch can be combined into two or more P2P messages. This is because if  $d^{xy} = K-1$  then one message is needed for  $K-1$   $\mathbf{H}$ -matrix row blocks and another message needed for the last block. Otherwise if  $d^{xy} < K/2$ , then the messages between  $p_x$  and  $p_y$  can be combined in  $\lceil K/d^{xy} \rceil$  P2P messages. Therefore, the number of messages sent per processor per epoch can be computed by

$$\sum_{i=1}^{\frac{K}{2}} 2 + \sum_{i=1}^{\frac{K-1}{2}} \frac{K}{i}.$$

The second summation is a harmonic series which can be approximated by  $\ln(K-1)/2 + 1$ , thus

$$\sum_{i=1}^{\frac{K}{2}} 2 + \sum_{i=1}^{\frac{K-1}{2}} \frac{K}{i} \approx K + K \ln(K-1)/2 \leq K \lg K \quad (12)$$

is the upper bound on the number of messages sent per processor per epoch.

To facilitate the presentation of the H&C strategy, we assume that each processor constructs a tabular-shaped message schedule (TSMS). In the TSMS of  $p_x$ , rows are the  $K-1$  processors that  $p_x$  communicates with during

an epoch, and columns represent sub-epochs as well as the corresponding  $\mathbf{H}$ -matrix row blocks updated by  $p_x$ . Each table entry  $\text{TSMS}(p_y, \mathbf{H}_\beta)$  represents the sub-epoch  $S_\beta^{-1}(p_y)$ .

---

**Algorithm 3** Message combining strategy on processor  $p_x$

---

**Require:**  $\text{SendSet}^k(p_x), \mathbf{H}^{xy} \forall k, y \in [K] \wedge y \neq x, S$

```

1: for  $k = 1$  to  $K$  do
2:   for each  $p_y \in \text{SendSet}^k(p_x)$  do
3:      $m^{\text{id}} \leftarrow \lceil \frac{k}{d^{xy}} \rceil$  ▷ get the message ID
4:     // get the  $\mathbf{H}$ -matrix block that  $p_x$  updates at SE  $k$ 
5:      $\beta \leftarrow S_k(p_x)$ 
6:     // add the  $d$ -gap rows  $\mathbf{H}_\beta^{xy}$  to Msg  $m^{\text{id}}$ 
7:      $M_{m^{\text{id}}}^{xy} \leftarrow M_{m^{\text{id}}}^{xy} \cup \mathbf{H}_\beta^{xy}$ 
8:     // When does  $p_x$  update the first block of  $m^{\text{id}}$ ?
9:      $t \leftarrow (m^{\text{id}} - 1) * d^{xy} + 1$ 
10:    // get the  $\mathbf{H}$ -matrix block that  $p_x$  updates at SE  $t$ 
11:     $\eta \leftarrow S_t(p_x)$ 
12:    // get the sub-epoch  $s$  at which  $p_y$  updates  $\mathbf{H}_\eta$ 
13:     $s \leftarrow S_\eta^{-1}(p_y)$ 
14:     $c\text{SendSet}^s(p_x) \leftarrow c\text{SendSet}^s(p_x) \cup \{p_y\}$ 
15:  end for
16: end for
```

---

Fig. 4 shows a TSMS for  $p_3$  using strata with  $\text{seed} = 5$ . In the figure, the circled TSMS entries denote the messages ( $\mathbf{H}$ -matrix row blocks) that can be combined. For instance, the communication requirement between  $p_3$  and  $p_7$  during an SGD epoch can be done with two messages. The first message, required at the beginning of sub-epoch 5, consists of  $\mathbf{H}_7^{3,7} \cup \mathbf{H}_8^{3,7} \cup \mathbf{H}_1^{3,7} \cup \mathbf{H}_2^{3,7}$ . The second message, required at the beginning of sub-epoch 1 of the next SGD epoch, consists of  $\mathbf{H}_3^{3,7} \cup \mathbf{H}_4^{3,7} \cup \mathbf{H}_5^{3,7} \cup \mathbf{H}_6^{3,7}$ . Observe that the sub-epoch at which the combined message should be sent is decided by the first  $\mathbf{H}$ -matrix block of the combined message. For instance, the first message to  $p_7$  must arrive before  $p_7$  starts updating rows in  $\mathbf{H}_7$  which is sub-epoch 5.

Algorithm 3 shows the procedure to construct combined messages from P2P messages at  $p_x$ . Given the  $\text{SendSet}^k(p_x) \forall k \in \{1, \dots, K\}$  and the  $d$ -gap rows between  $p_x$  and  $\{p_y \mid y \neq x\}$ , the combined messages are constructed as follows: There are  $\lceil K/d^{xy} \rceil$  possible messages to  $p_y$  each of which is identified by  $m^{\text{id}}$ . For each  $p_y \in \text{SendSet}^k(p_x)$  the rows in  $\mathbf{H}_\beta^{xy}$  are assigned to a combined message  $M_{m^{\text{id}}}^{xy}$  (lines 3 and 4). Then,  $p_y$  is added to the new send set of the sub-epoch at which message  $m^{\text{id}}$  is sent (lines 5-9).

Algorithm 1 can be modified to accommodate the H&C strategy as follows: After constructing the P2P communication (line 4), Algorithm 3 is used to combine the messages. Then, lines 8-13 can be replaced with the sending/receiving of combined messages; for each  $p_y$  in  $c\text{SendSet}^k(p_x)$  a combined message is identified using  $m^{\text{id}} = \lceil k/d^{xy} \rceil$  and sent to  $p_y$ , and similarly so for receiving from each  $p_z$  in  $c\text{RecvSet}^k(p_x)$ .

It is important to make sure that the  $K \lg K$  messages sent per epoch are uniformly distributed over  $K$  sub-epochs. Otherwise, some sub-epochs will constitute a performance bottleneck due to high number of messages. We show that utilizing Algorithm 3 for combining the messages has the nice property of limiting the expected number of messages sent by each processor at each sub-epoch to  $\mathcal{O}(\lg K)$ .



**Theorem 1.** Using the H&C strategy, the expected number of messages sent by each processor at each sub-epoch is  $\mathcal{O}(\lg K)$ .

*Proof.* Consider a set  $F^{xy}$  that consists of all sub-epochs wherein a message is sent from  $p_x$  to  $p_y$ . For each sub-epoch  $k$ , the function

$$\sigma(k, p_x, p_y) = \begin{cases} 1 & k \in F^{xy} \\ 0 & \text{otherwise} \end{cases}$$

defines if there is a message to be sent from  $p_x$  to  $p_y$  in  $k$ .

We can prove that  $\mathcal{O}(\lg K)$  messages are sent by each processor at each sub-epoch as follows. The number of messages sent by  $p_x$  per sub-epoch is equal to the number of occurrences of that sub-epoch in  $\biguplus_{y \in [K] \wedge y \neq x} F^{xy}$ . For each processor  $p_y$  with distance  $d^{xy}$ , the probability that  $k$  is one of the sub-epochs in which a message is sent to  $p_y$  is equal to  $1/d^{xy}$ . In other words, given  $K$  sub-epochs, the probability that sub-epoch  $k$  will be used to send one of the  $\lceil K/d^{xy} \rceil$  messages is  $1/d^{xy}$ . Then, the expected number of messages from  $p_x$  to  $p_y$  at sub-epoch  $k$  is

$$\begin{aligned} \mathbf{E}[\sigma(k, p_x, p_y)] &= \Pr(\sigma(k, p_x, p_y) = 1) \times 1 + \\ &\Pr(\sigma(k, p_x, p_y) = 0) \times 0 = \frac{1}{d^{xy}}. \end{aligned}$$

Using linearity of expectation, the expected total number of messages sent by  $p_x$  at sub-epoch  $k$  is

$$\mathbf{E}\left[\sum_{p_y \neq p_x} \sigma(k, p_x, p_y)\right] = \sum_{i=1}^K \frac{1}{i} \approx \ln K + 1 \leq \lg K + 1. \quad (13)$$

□

## 5 HP MODEL FOR REDUCING BANDWIDTH COST

There exists two hypergraph models for 1D partitioning of sparse matrices for SpMV-like kernels; namely the column-net model for rowwise partitioning and the row-net model for columnwise partitioning [9]. In these models, the ‘‘connectivity–1’’ metric [9] is utilized for partitioning objective of reducing the communication volume in SpMV-like kernels, whereas the partitioning constraint is maintaining computational balance among processors. As mentioned earlier, rating matrices usually have larger number of rows than columns, hence we mainly focus on rowwise partitioning of rating matrix  $\mathbf{R}$ . The hypergraph model discussed here is topologically similar to the column-net model, however the cutsize metric utilized in the partitioning objective is different.

In the hypergraph model  $\mathcal{H}_{\mathbf{R}} = (\mathcal{V}, \mathcal{N})$ , there exists a vertex  $v_i \in \mathcal{V}$  for each row  $\mathbf{r}_i$  of  $\mathbf{R}$  and a net (hyperedge)  $n_j \in \mathcal{N}$  for each column  $\mathbf{c}_j$  of  $\mathbf{R}$ . Each net  $n_j$  connects the vertices corresponding to the  $\mathbf{R}$ -matrix rows that contain nonzeros in column  $\mathbf{c}_j$ . That is,  $\text{Pins}(n_j) = \{v_i \in \mathcal{V} \mid r_{ij} \neq 0\}$ . Each vertex  $v_i$  is associated with a weight equals to the number of nonzeros in row  $\mathbf{r}_i$ . Each net is associated with a cost  $F$ .

A  $K$ -way partition  $\Pi(\mathcal{H}_{\mathbf{R}}) = \{\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_K\}$  is decoded as a  $K$ -way rowwise partition of  $\mathbf{R}$ , where the rows corresponding to the vertices in part  $\mathcal{V}_\alpha$  constitute the row block  $\mathbf{R}_\alpha$ , for  $\alpha = 1, 2, \dots, K$ . Without loss of generality, row block  $\mathbf{R}_\alpha$  is assigned to processor  $p_\alpha$  for  $\alpha = 1, 2, \dots, K$ .

The  $\mathbf{W}$ -matrix rows are partitioned conformably with the  $\mathbf{R}$ -matrix row partition. That is,  $\mathbf{W}$ -matrix rows in  $\mathbf{W}_\alpha$  correspond to the  $\mathbf{R}$ -matrix rows in  $\mathbf{R}_\alpha$ .

In partition  $\Pi(\mathcal{H}_{\mathbf{R}})$ , the weight of each part is equal to the sum of the weights of the vertices in that part. Hence, the partitioning constraint of maintaining balance on the part weights encodes maintaining balance on the nonzero counts of the  $\mathbf{R}$ -matrix row blocks. This in turn corresponds to maintaining balance on the computational loads of the processors.

In partition  $\Pi(\mathcal{H}_{\mathbf{R}})$ , a net  $n_j$  is said to connect a part  $\mathcal{V}_\alpha$  if it connects at least one vertex in part  $\mathcal{V}_\alpha$ , that is,  $\text{Pins}(n_j) \cap \mathcal{V}_\alpha \neq \emptyset$ . The connectivity set  $\Lambda(n_j)$  of a net  $n_j$  is defined as the set of the parts that net  $n_j$  connects, whereas the connectivity  $\lambda(n_j)$  denotes the number of parts connected by net  $n_j$ , that is  $\lambda(n_j) = |\Lambda(n_j)|$ . A net  $n_j$  is said to be cut if  $\lambda(n_j) > 1$  and uncut otherwise. The partitioning objective is to minimize the cutsize which is defined over the cut nets.

In this model,  $\Lambda(n_j)$  also represents the set of  $\mathbf{R}$ -matrix row blocks that has at least one nonzero in column  $\mathbf{c}_j$  of  $\mathbf{R}$ . Hence, the connectivity set of net  $n_j$  denotes the set of processors that update the  $\mathbf{H}$ -matrix row  $\mathbf{h}_j$ . Consider the  $\mathbf{H}$ -matrix row  $\mathbf{h}_j$  corresponding to a cut net  $n_j$  in the P2P communication scheme. Also consider  $\mathbf{h}_j$  update sequence defined using the connectivity set and strata. For each epoch, each processor except the last processor in the sequence should send its updated  $\mathbf{h}_j$  value once to the next processor in the sequence. The last processor sends its updated  $\mathbf{h}_j$  value to the first processor for the next iteration. Hence, each cut net  $n_j$  incurs a communication volume of  $F\lambda(n_j)$ . On the other hand, uncut nets incurs no communication. Therefore, cutsize which encapsulates the total communication volume during an SSGD epoch can be computed as

$$\sum_{n_j \ni \lambda(n_j) > 1} F\lambda(n_j). \quad (14)$$

Among the various cutsize metrics in the literature, cutsize (14) is called as the sum of external degrees (SOED) [10].

There exists several successful hypergraph partitioning tools that utilize multilevel recursive bipartitioning (RB) algorithms. Among these partitioning tools, to our knowledge, only *hMETIS* [10] supports the SOED metric via direct multi-way partitioning [11]. In fact Karypis and Kumar [11] clearly indicates that RB framework does not allow directly optimizing the SOED metric. Here, we propose an RB framework that encodes the minimization of the SOED metric correctly.

In the RB framework, a given hypergraph  $\mathcal{H}$  is recursively bipartitioned until  $K$  parts are obtained, assuming  $K$  is a power of two without loss of generality. At each RB step, a bipartition  $\Pi_2 = \{\mathcal{V}_L, \mathcal{V}_R\}$  on the current hypergraph forms two vertex-induced subhypergraphs  $\mathcal{H}_L = (\mathcal{V}_L, \mathcal{N}_L)$  and  $\mathcal{H}_R = (\mathcal{V}_R, \mathcal{N}_R)$ . Here,  $\mathcal{V}_L$  and  $\mathcal{V}_R$  are respectively used to refer to the left and right parts of the bipartition. The net sets  $\mathcal{N}_L$  and  $\mathcal{N}_R$  are constructed through cut net splitting method [9] as follows: Internal nets of  $\mathcal{V}_L$  and  $\mathcal{V}_R$  are respectively included in  $\mathcal{N}_L$  and  $\mathcal{N}_R$ . A cut net  $n_j$  in  $\Pi_2$  is split into two subnets  $n'_j$  and  $n''_j$ , where  $\text{Pins}(n'_j) = \text{Pins}(n_j) \cap \mathcal{V}_L$  and  $\text{Pins}(n''_j) = \text{Pins}(n_j) \cap \mathcal{V}_R$ .

In order to encode the SOED metric (14), we propose the following strategy during the RB framework. We assign a



cost of  $2F$  to each net of the initial hypergraph. Then, after each RB step, internal nets inherit their cost, whereas splitted nets are assigned a cost of  $F$ . That is, a net holds its cost of  $2F$  until it becomes cut for the first time, then a cost of  $F$  is assigned to each of split subnets and they inherit their cost of  $F$  through the further RB steps until the end of the partitioning. Hence, when a net becomes cut for the first time it incurs  $2F$  to the cutsize, then whenever its subnets become cut they incur  $F$  to the cutsize. In this way, the sum of all cut net costs encountered during the overall RB algorithm becomes equal to the SOED metric (14).

## 6 EXPERIMENTAL EVALUATIONS

### 6.1 Experimental framework

We evaluate the contributions proposed in this work through comparing three methods implementing parallel SSGD using six real-world rating matrices. The first method, DSGD, is the algorithm proposed in the original work of Gemulla et al. [3]. DSGD performs block-wise communication of  $\mathbf{H}$ -matrix row blocks in each sub-epoch. The second method, P2P, uses P2P messages as in Algorithm 1. The third, H&C, uses combined P2P messages (Algorithm 3) for communication.

In all three methods, column-to-stratum assignments are done randomly in such a way that the number of columns per stratum differs by at most one. Row-to-processor assignments are obtained either randomly in a way similar to that of column-to-stratum assignments, or using the HP method discussed in Section 5. Whenever the former is used, the method will be prefixed by RAND, whereas if the latter is used the method will be prefixed by HP. The HP method is implemented according to the RB framework described in Section 5 to encapsulate the SOED metric. In order to obtain two-way partitions on the (sub)hypergraphs at each RB level, we use the HP tool PaToH [9] with default parameters in SPEED mode.

We implemented the parallel SSGD code that includes DSGD, P2P and H&C in C and used MPI for inter-process communications. We perform our experiments on an HPC system with AMD EPYC 7742 processors and a high-speed HDR InfiniBand network with 200Gb/s bandwidth.

We compare the three methods in terms of communication cost metrics as well as SGD iteration time. The communication cost metrics consist of bandwidth-oriented metrics:  $sum-max\ vol$  and  $tot\ vol$ , and latency-oriented metrics:  $sum-max\ msgs$  and  $tot\ msgs$ .  $sum-max\ msgs$  is calculated as follows: at each sub-epoch, the number of messages sent by the bottleneck processor (the processor that sends highest number of messages) is obtained. Then, the summation is taken over all  $K$  sub-epochs. That is,

$$sum-max\ msgs = \sum_{k=1}^K \max_{x \in [K]} (|SendSet^k(p_x)|).$$

In a similar way,  $sum-max\ vol$  is computed as

$$sum-max\ vol = \sum_{k=1}^K \max_{x \in [K]} (SendVol^k(p_x)).$$

$tot\ msgs$  and  $tot\ vol$  are respectively computed as

$$tot\ msgs = \sum_{k=1}^K \sum_{x \in [K]} (|SendSet^k(p_x)|),$$

$$tot\ vol = \sum_{k=1}^K \sum_{x \in [K]} (SendVol^k(p_x)).$$

Here,  $SendVol^k(p_x) = |\mathbf{H}_{S_{k-1}(p_x)}|$  if DSGD is used, and  $SendVol^k(p_x) = \sum_{p_y} |\mathbf{H}_{S_{k-1}(p_x)}^{xy}|$  if P2P or H&C are used. Whenever the values for the volume of communication are presented, these values are normalized with respect to  $F$ . This uncoupling of  $F$  from the volume values helps evaluate the proposed methods and model for any  $F$  value.

TABLE 1: Properties of matrices in the dataset

Matrix	#rows	#cols	#nnz	density
Amz Items	21.177M	9.874M	82.677M	3.95E-07
Amz Books	8.026M	2.330M	22.50M	1.20E-06
Amz Clothing & Jewelry	3.117M	1.136M	5.75M	1.62E-06
Goodreads Reviews	0.465M	2.080M	15.740M	1.63E-05
Google Reviews	5.055M	3.117M	11.454M	7.27E-07
Twitch	15.524M	6.162M	474.677M	4.96E-06

Table 1 shows the real-world matrices used to evaluate the proposed methods and their properties. Amz Items contains product reviews from Amazon between May 1996 - July 2014 [12] with aggressive duplicate removal. The other two amazon datasets, Books and Clothing, are category-based subsets of the original comprehensive reviews. Goodread Reviews contains user ratings of books from the Goodreads website [13]. Google Reviews contains user ratings/reviews of local businesses from the Google Maps website [14], [15]. Twitch contains ratings relative to how much time a user spent on a stream in the Twitch streaming website [16]. The original data does not contain any explicit ratings. We modified the dataset to represent  $(user, stream, rating)$  such that the rating value is proportional to the amount of time the user spent in the specific stream.

### 6.2 Evaluations with Communication Cost Metrics

Figs. 5a, 5b and 5c compare DSGD, P2P and H&C in terms of communication cost metrics  $tot\ vol$ ,  $sum-max\ vol$  and  $tot\ msgs$  on  $K = 1024$  processors. In the figures, the red bars denote RAND-based methods whereas light blue bars denote HP-based methods. HP does not affect DSGD's communication which is why HP is not applicable for DSGD and hence DSGD has only red bars. Comparison in terms of  $sum-max\ msgs$  will be discussed in Fig. 6.

#### 6.2.1 Bandwidth-oriented Communication Cost Metrics

As seen in Fig. 5a, both P2P and H&C incur the essential amount of communication volume as defined in (10), without any forwarding overhead. Compared to DSGD, both RAND- and HP-based P2P and H&C methods incur significantly reduced amount of communication volume per epoch (more than 10x). Compared to RAND, the HP-based P2P and H&C methods incur significantly reduced volume (between 1.4x and 5x).

Fig. 5b shows that in all matrix instances P2P and H&C have a significantly reduced  $sum-max\ vol$  compared to DSGD (more than 10x). H&C has slightly higher  $sum-max\ vol$  compared to P2P. This is because combining the messages disturbs the random volume balancing of P2P. As expected, HP-based P2P incurs less  $sum-max\ vol$  compared to RAND-based

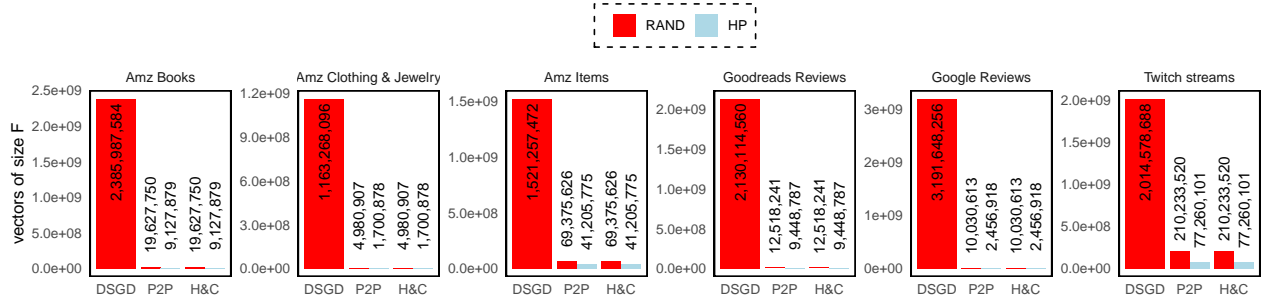
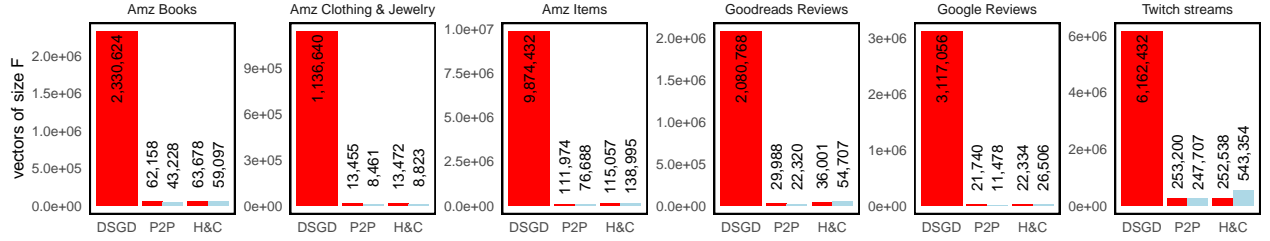
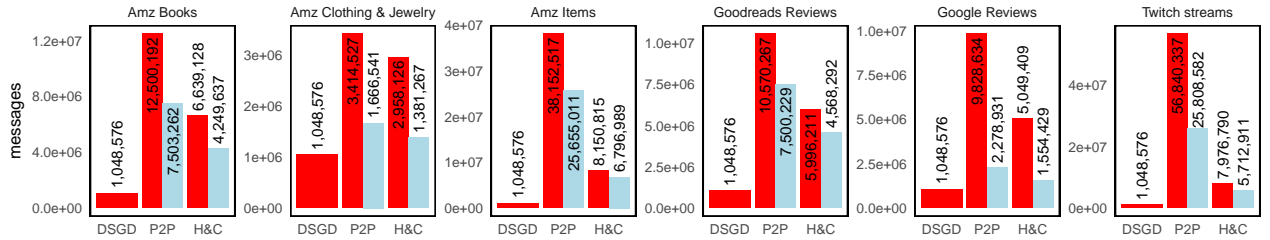
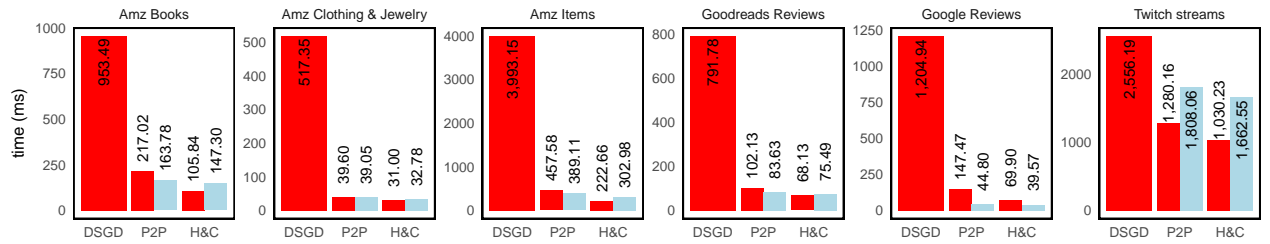
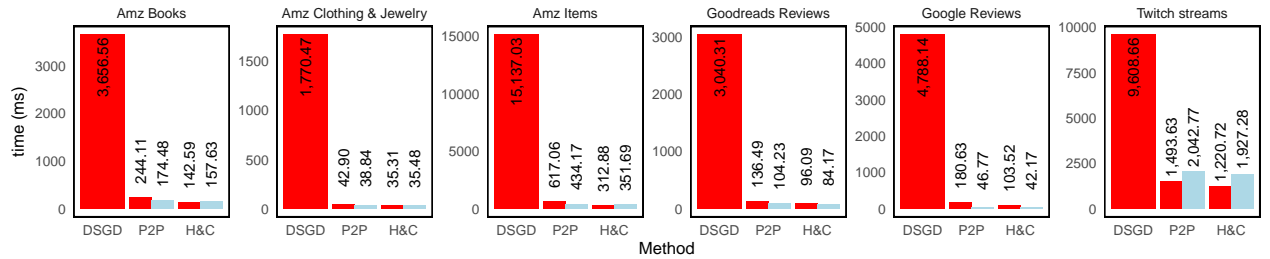
(a) *tot vol* in an SGD epoch(b) *sum-max vol* in an SGD epoch(c) *tot msgs* in an SGD epoch(d) SGD iteration times when  $F = 16$ (e) SGD iteration times when  $F = 64$ 

Fig. 5: Comparing RAND- and HP-based P2P and H&C methods against RAND-based DSGD using communication cost metrics (a to c) and SGD iteration time (d and e) using all dataset matrices on  $K = 1024$  processors.

P2P. HP-based H&C shows a decrease in *sum-max vol* on two matrices (Amz Books and Amz Clothing & Jewelry), and an increase in other four matrices. This is because the HP method, when used for H&C, does not encapsulate reducing the *sum-max vol* metric.

### 6.2.2 Latency-oriented Communication Cost Metrics

Fig. 5c shows that the H&C method significantly reduces *tot msgs* on all dataset matrices. DSGD always incurs a constant number of messages for each  $K$  value, thus *tot msgs* is always equal to  $K^2 = (1024)^2 = 1048576$ . *tot msgs* of P2P can go up to  $K^2 \times (K-1)$ . On the other hand, H&C keeps *tot msgs* limited to  $\mathcal{O}(K^2 \lg K)$ . Depending on the sparsity pattern of the matrix, *tot msgs* of P2P can be very high (e.g., Amz Books, Amz Items and Twitch) or relatively close to the lower bound (e.g., Amz Clothing & Jewelry). The H&C method successfully controls the fluctuation in the number of messages thanks to the  $\lg K$  factor. The significant reduction in *tot vol* of HP-based P2P and H&C methods compared to those of RAND-based is expected to reflect on the total number of messages, which is the case as shown in the figure.

Fig. 6 showcases the H&C method’s regularization of messages sent per epoch over  $K$  sub-epochs. In order to experimentally verify the  $\mathcal{O}(\lg K)$  bound given in Theorem 1, we introduce the *max-max msgs* metric as the maximum number of messages sent per sub-epoch among all sub-epochs. That is,

$$\text{max-max msgs} = \max\{\max_{x \in [K]}(|\text{SendSet}^k(p_x)|) \mid k \in [K]\}.$$

As seen in Fig. 6a, using H&C, *max-max msgs* is empirically found to be  $\approx 3 \times \lg K$ , which is very close to the expected  $\lg K$  bound on the number of messages per sub-epoch given in (13). The figure shows that P2P incurs high *max-max msgs* on  $K = 256$ , and then the *max-max msgs* values start to decrease as  $K$  increases. We believe this is attributed to the ability of random partitioning to balance P2P message counts and volume. In Fig. 6b, the *sum-max msgs* metric is shown for all matrices in the dataset using P2P and H&C on  $K = 64, \dots, 1024$  processors. The figure shows the success of H&C in keeping the number of messages under the  $K \lg K$  theoretical bound. Since P2P’s *sum-max msgs* do not decrease as  $K$  increases, this means maximum number of messages per sub-epoch are almost equal among all sub-epochs, especially when  $K \geq 512$ . On the other hand, although the H&C’s *max-max msgs* come very close to those of P2P on some instances such as Goodreads Reviews and Google Reviews, *sum-max msgs* stay significantly less than those of P2P. This means that although the maximum number of messages sent per sub-epoch can reach  $3 \lg K$  in very few sub-epochs, it is still equal to or less than the expected  $\lg K$  messages.

### 6.3 Evaluations with SGD Iteration Time

Figs. 5d and 5e compare the methods in terms of SGD iteration time on  $K = 1024$  processors respectively using  $F = 16$  and  $F = 64$  values. The figure shows that the P2P improvement over DSGD is significant (more than 4x on all matrices, except for Twitch which is 1.4x) when  $F = 16$ . The improvement grows further as  $F$  increases to 64. It

TABLE 2: Normalized cost metrics of P2P-HP with respect to P2P-RAND on  $K = 1024$  processors. A value  $v < 1$  means P2P-HP outperforms P2P-RAND by  $(1 - v) \times 100\%$ .

Matrix	total messages	total volume	SGD time	
			$F=16$	$F=64$
Amz Items	0.67	0.59	0.85	0.70
Amz Books	0.60	0.47	0.75	0.71
Amz Clothing & Jewelry	0.49	0.34	0.99	0.91
Goodreads Reviews	0.71	0.75	0.82	0.76
Google Reviews	0.23	0.24	0.30	0.26
Twitch streams	0.45	0.37	1.41	1.37
<b>Avg. (Geometric mean)</b>	<b>0.50</b>	<b>0.43</b>	<b>0.78</b>	<b>0.71</b>

becomes more than 15x on all matrices except Twitch, and on Twitch the improvement becomes at least 4.7x.

Using RAND, the H&C improvement over P2P is also significant. When  $F = 16$ , H&C improves the iteration runtime over P2P by 2x, 1.2x, 2x, 1.5x, 2.15x, and 1.25x respectively on Amz Books, Amz Clothing & Jewelry, Amz Items, Goodreads Reviews, Google Reviews and Twitch. When  $F = 64$ , the respective values become 1.7x, 1.2x, 2x, 1.4x, 1.74x, and 1.22x.

Using HP improves the P2P runtime by 1.3x, 1.17x, 1.22x and 3.35x on Amz Books, Amz Items, Goodreads Reviews and Google Reviews, respectively, when  $F = 16$ . On Amz Clothing & Jewelry there is no significant improvement and on Twitch there is deterioration by 1.4x. When  $F = 64$ , HP improves the P2P runtime by 1.4x, 1.42x, 1.3x and 3.9x respectively on Amz Books, Amz Items, Goodreads Reviews and Google Reviews. Table 2 also shows the normalized P2P-HP cost metrics with respect to those of P2P-RAND. On average, HP improves (reduces) the SGD iteration time by 22% when  $F = 16$  and 29% when  $F = 64$ . The increase in the gap between HP and RAND in terms of P2P runtime when  $F$  grows from 16 to 64 is expected since the HP method aims at reducing the total volume, effect of which is seen more with higher  $F$  values. We observed that the HP method improves the H&C runtime compared to RAND only on Goodreads Reviews and Google Reviews.

Fig. 7 shows the strong scaling curves of RAND-based DSGD, P2P and H&C using two different  $F$  values on  $K = \{64, 128, 256, 512, 1024\}$  processors. As seen in the figure, P2P and H&C show superior scaling compared to DSGD. Furthermore, H&C performs significantly better than P2P, especially with smaller  $F$  values.

Increasing the  $F$  value is expected to render the SGD communication as bandwidth-bound. Therefore, the effect of the methods that reduce the bandwidth (volume of communication) becomes conspicuous. This is observed in two different cases when moving from  $F = 16$  in Fig. 7a to  $F = 64$  in Fig. 7b: (i) the performance gap between P2P/H&C and DSGD increases as  $F$  becomes larger as a result of the huge reduction in communication volume when using P2P/H&C, and (ii) the difference in performance between P2P and H&C slightly reduces due to the communication overhead leaning towards bandwidth.

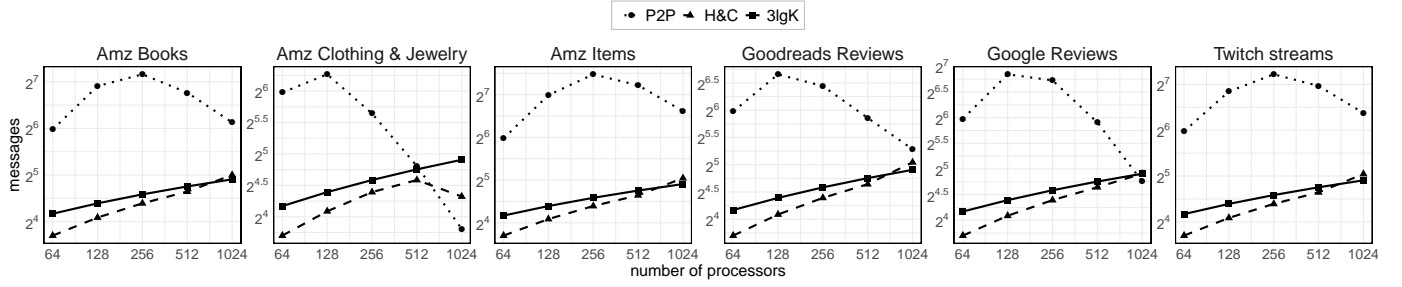
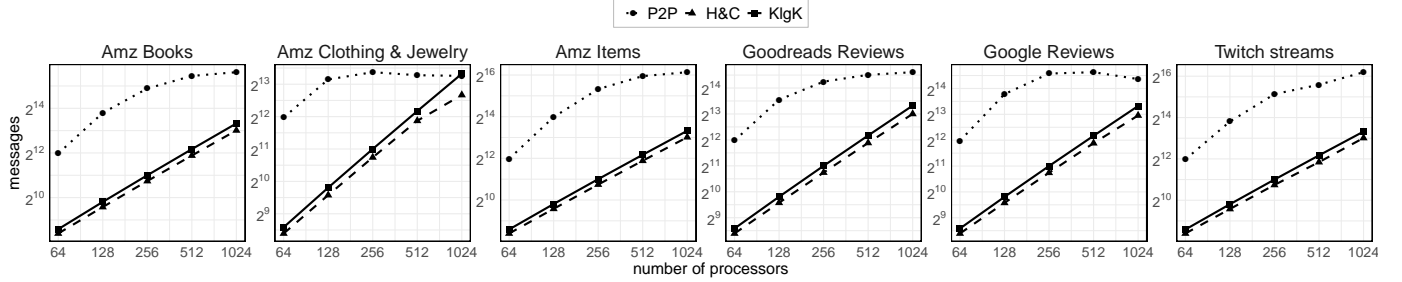
(a) *max-max msgs* in an SGD epoch ( $\log_2\text{-}\log_2$ )(b) *sum-max msgs* in an SGD epoch ( $\log_2\text{-}\log_2$ )

Fig. 6: Showcasing the upper bound of the max-max messages and sum-max messages sent per sub-epoch using the H&C method compared to P2P on  $K = \{64, \dots, 1024\}$  processors.

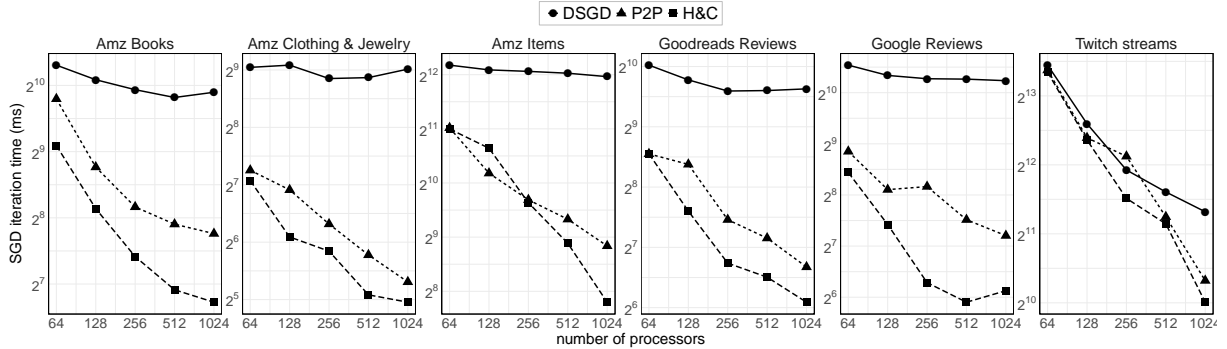
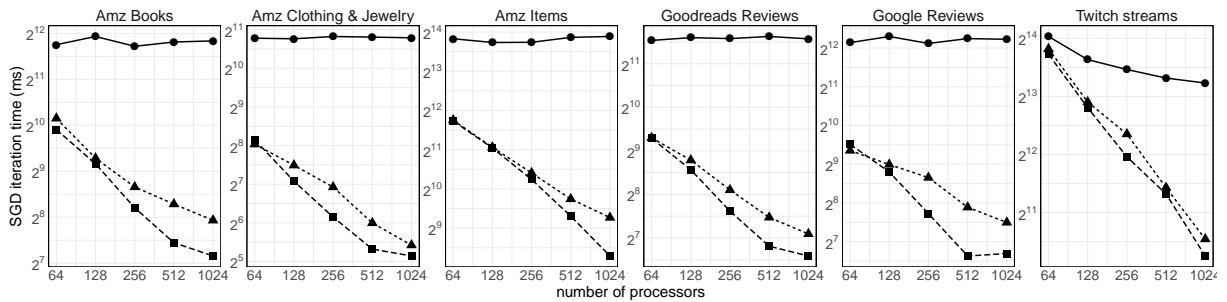
(a) SGD iteration times when  $F = 16$  ( $\log_2\text{-}\log_2$ )(b) SGD iteration times when  $F = 64$  ( $\log_2\text{-}\log_2$ )

Fig. 7: Strong scaling curves of DSGD, P2P and H&C on  $K = \{64, 128, 256, 512, 1024\}$  processors using all dataset matrices with two  $F$  values.

#### 6.4 Evaluations with Loss Values

Since all the methods discussed in this work follow the stratified SGD algorithm, their loss values per iteration is expected to be very similar regardless of the communication strategy used or number of processors. We demonstrate

this using Fig. 8a. The figure shows the loss value ( $y$ -axis) following each SGD iteration ( $x$ -axis) of Amz Books and Goodreads Reviews using the RAND-based DSGD, P2P, H&C methods on  $K = \{64, 256, 1024\}$  processors. The loss values are very close as expected thus the curves appear to be on top of each other.

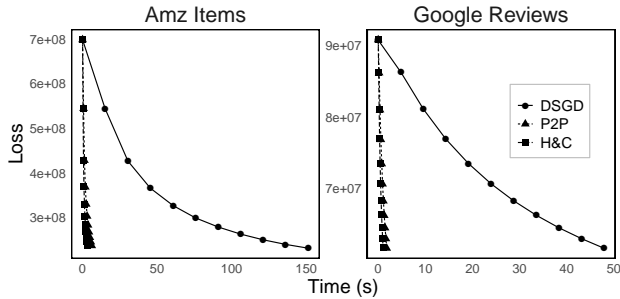
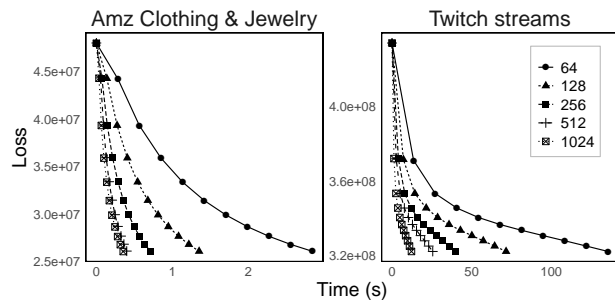
(a) SGD iteration vs. loss for all methods on different  $K$  values.(b) Time vs. loss for all methods on  $K=1024$  processors.(c) Time vs. loss for H&C on different  $K$  values.

Fig. 8: Loss curves.

Fig. 8b shows the amount of time ( $x$ -axis) required to reach a certain loss value ( $y$ -axis) of Amz Items and Google Reviews using the RAND-based DSGD, P2P, H&C methods on 1024 processors. The figure shows that DSGD requires significantly more time to reach a certain loss value compared to P2P and H&C.

Fig. 8c shows the scaling behavior of the RAND-based H&C method with Amz Clothes & Jewelry and Twitch in terms of loss value as the time increases.

## 7 RELATED WORK

There exist several works in the literature that adopt the SSGD for parallel matrix completion for shared-memory systems [17], [18], [19] and distributed-memory systems [3], [4], [5]. Here, we focus on the works that involve distributed-memory implementations. The work of Gemulla et al. [3] proposed the SSGD approach as well as the parallel DSGD algorithm discussed in Sections 2.2 and 3.2. Teflioudi et al. [4] proposed DSGD++, an improved DSGD framework for better performance. They use computation and communication overlaying through dividing the input matrix into

$K \times 2K$  blocks, and in each of the  $K$  sub-epochs DSGD++ performs computation on  $K$  blocks while simultaneously communicating the other  $K$  blocks. They report up to 2.3x improvement over DSGD++ in terms of runtime. Yun et al. [5] extend the idea of DSGD++ in their framework, NOMAD, and divide the input matrix into  $K \times M$  blocks. Each of the  $K$  processors dedicates  $\ell$  threads to update  $\ell$   $\mathbf{H}$ -matrix rows, and  $M - \ell$  other threads for communication. Once processor  $p_x$  updates an  $\mathbf{H}$ -matrix row, or a set of rows, it sends it/them to another processor  $p_y$  that has idle computation threads. DSGD, DSGD++ and NOMAD have the same total communication volume during an SGD epoch which is equals to  $F \times M \times K$  as discussed in Section 3.2. The number of messages sent per processor during an epoch of DSGD and DSGD++ has an upper bound of  $\mathcal{O}(K)$ , whereas NOMAD may send up to  $\mathcal{O}(M)$  messages. Guo et al. [6] proposed a novel framework, BaPa, for improving the nonzero load balance of DSGD through a novel algorithm for balancing per-processor and per-epoch ratings. Their BaPa-based DSGD shows a significant runtime improvement on small number of processors ( $< 16$ ). However, their results show that both the original DSGD as well as the BaPa-based DSGD stop scaling after 256 processors.

There are several asynchronous-SGD-based parallel matrix completion algorithms in the literature. ASGD [4] (shown in the upper part of Fig. 1) is the simplest example of such algorithm. During ASGD, it is possible that several processors update the same  $\mathbf{H}$ -matrix row  $\mathbf{h}_j$  at the same time (i.e., stale updates). This results in each processor having a different copy of  $\mathbf{h}_j$ . These copies are coordinated by sending them to a processor responsible for  $\mathbf{h}_j$ . This processor takes their average and then sends the up-to-date version of  $\mathbf{h}_j$  back to the same set of processors. This type of coordination is done once or more during an SGD epoch [4], [20]. GASGD [20] extends ASGD by utilizing intelligent partitioning for balancing computational loads, reducing communication between processors, and reducing staleness. The authors utilize a bipartite graph model and propose a partitioning method based on the *balanced  $K$ -way vertex-cut* problem [21] to achieve the partitioning goals. Luo et al. [22] proposed a different strategy to facilitate asynchronously computing SGD in parallel which is called alternating SGD. In alternating SGD, each epoch is divided into two sub-epochs where in each sub-epochs one factor matrix is fixed and the other is updated. This approach enables limiting the feature vector updates that use stale data to one of the two factor matrices during a sub-epoch. Recently, Shi et al. [23] proposed a distributed algorithm based on alternating SGD with data-aware partitioning.

## 8 CONCLUSIONS

We proposed a framework for scaling stratified SGD through significantly reducing the communication overhead. The framework targets at reducing the bandwidth overhead by efficiently finding the required communication during an SGD epoch, using P2P messages to perform it, and an HP-based method to further reduce the P2P communication volume. The framework targets at reducing the increase in latency overhead through the novel H&C strategy to limit the number of messages sent by a processor per epoch

to  $\mathcal{O}(K \lg K)$ . Our proposed framework achieves scalable distributed SGD, on up to  $K = 1024$  processors, without any compromise on convergence rate or any update on stale factors. The proposed framework achieves up to 15x runtime improvement over the state of the art DSGD method, on 1024 processors, using six real-world rating matrices.

## ACKNOWLEDGMENTS

This work is supported by the Scientific and Technological Research Council of Türkiye (TÜBİTAK) under project EEEAG-119E035. Computing resources used in this work were provided by the National Center for High Performance Computing of Türkiye (UHem) under grant number 4009972021.

## REFERENCES

- [1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [2] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.
- [3] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 69–77.
- [4] C. Teflioudi, F. Makari, and R. Gemulla, "Distributed matrix completion," in *2012 IEEE 12th International Conference on Data Mining*, 2012, pp. 655–664.
- [5] H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon, "NOMAD: Non-locking, stochastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion," *Proceedings of the VLDB Endowment*, vol. 7, no. 11, 2014.
- [6] R. Guo, F. Zhang, L. Wang, W. Zhang, X. Lei, R. Ranjan, and A. Y. Zomaya, "BaPa: a novel approach of improving load balance in parallel matrix factorization for recommender systems," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 789–802, 2021.
- [7] R. O. Selvitopi, M. M. Ozdal, and C. Aykanat, "A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 632–645, 2015.
- [8] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [9] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, July 1999.
- [10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, March 1999.
- [11] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," *VLSI design*, vol. 11, no. 3, pp. 285–300, 2000.
- [12] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *proceedings of the 25th international conference on world wide web*, 2016, pp. 507–517.
- [13] M. Wan and J. McAuley, "Item recommendation on monotonic behavior chains," in *Proceedings of the 12th ACM conference on recommender systems*, 2018, pp. 86–94.
- [14] R. He, W.-C. Kang, and J. McAuley, "Translation-based recommendation," in *Proceedings of the eleventh ACM conference on recommender systems*, 2017, pp. 161–169.
- [15] R. Pasricha and J. McAuley, "Translation-based factorization machines for sequential recommendation," in *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018, pp. 63–71.
- [16] J. Rappaz, J. McAuley, and K. Aberer, "Recommendation on live-streaming platforms: Dynamic availability and repeat consumption," in *Fifteenth ACM Conference on Recommender Systems*, 2021, pp. 390–399.
- [17] J. Oh, W.-S. Han, H. Yu, and X. Jiang, "Fast and robust parallel sgd matrix factorization," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 865–874.
- [18] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin, "A fast parallel stochastic gradient method for matrix factorization in shared memory systems," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 1, mar 2015.
- [19] W. Tan, L. Cao, and L. Fong, "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*, 2016, pp. 219–230.
- [20] F. Petroni and L. Querzoni, "GASGD: Stochastic gradient descent for distributed asynchronous matrix completion via graph partitioning," in *Proceedings of the 8th ACM Conference on Recommender Systems*, ser. RecSys '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 241–248.
- [21] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. USA: USENIX Association, 2012, p. 17–30.
- [22] X. Luo, H. Liu, G. Gou, Y. Xia, and Q. Zhu, "A parallel matrix factorization based recommender by alternating stochastic gradient descent," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 7, pp. 1403–1412, 2012, advanced issues in Artificial Intelligence and Pattern Recognition for Intelligent Surveillance System in Smart Home Environment.
- [23] X. Shi, Q. He, X. Luo, Y. Bai, and M. Shang, "Large-scale and scalable latent factor analysis via distributed alternative stochastic gradient descent for recommender systems," *IEEE Transactions on Big Data*, pp. 1–1, 2020.



**Nabil Abubaker** received the BS degree in 2014 from An-Najah National University, Palestine, and the MS and PhD degrees from Bilkent University, Turkey in 2016 and 2022, respectively, all in Computer Engineering. He is currently affiliated with Bilkent University as a postdoctoral researcher. His research interests include parallel computing and algorithms with focus on reducing data movement in scientific and machine learning applications running on HPC systems.



**M. Ozan Karsavuran** received the BS, MS, and PhD degrees in 2012, 2014, and 2020, respectively, in computer engineering from Bilkent University, Turkey. He is currently a postdoctoral scholar of the Computing Sciences Area at the Lawrence Berkeley National Laboratory. His research interests include combinatorial scientific computing, graph and hypergraph partitioning for sparse matrix and tensor computations, and parallel computing in distributed and shared memory systems.



**Cevdet Aykanat** received the BS and MS degrees from Middle East Technical University, Turkey, both in electrical engineering, and the PhD degree from Ohio State University, Columbus, in electrical and computer engineering. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Turkey, where he is currently a professor. His research interests mainly include parallel

computing and its combinatorial aspects. He is the recipient of the 1995 Investigator Award of The Scientific and Technological Research Council of Turkey and 2007 Parlar Science Award. He has served as an Associate Editor of IEEE Transactions of Parallel and Distributed Systems between 2009 and 2013.