Random Sampling to Update Partial Singular Value Decomposition on a Hybrid CPU/GPU Cluster

Ichitaro Yamazaki, Jakub Kurzak, Piotr Luszczek, Jack Dongarra
{iyamazak, kurzak, luszczek, dongarra}@eecs.utk.edu,
Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, Tennessee, U.S.A.

***Abstract*—There is a growing demand for a novel algorithm that can efficiently analyze the massive data being generated from many modern applications. To this end, a partial singular value decomposition (SVD) of the sparse matrix representing the data is a powerful tool. However, computing the SVD of the large data can take a significant amount of time even on a modern large-scale computer. To address this challenge, we propose sampling algorithms to update, instead of recompute, the SVD. Our experimental results demonstrate that these sampling algorithms can obtain the desired accuracy of the SVD with a small number of data accesses, and compared to the state-of-the-art updating algorithm, they often require much lower computational and communication costs. Our performance studies on a hybrid CPU/GPU cluster show that these sampling algorithms can obtain significant speedups over the state-of-the-art algorithm.**

I. Introduction

In recent years, the amount of data being generated from the observations, experiments, and simulations in many areas of studies (e.g., science, engineering, medicine, finance, social media, and e-commerce) is growing at an unprecedented pace [5], [7]. These data is commonly referred to as "Big Data," and the algorithmic challenges to analyze the data are exacerbated by its massive volume, wide variety, and high veracity and velocity [19]. To this end, a partial singular value decomposition (SVD) [10] of the sparse matrix representing the data is a powerful tool. The ability of the SVD to filter out noise and extract the underlying features of the data has been demonstrated in many data analysis tools, including Latent Semantic Indexing (LSI) [6], [1], recommendation systems [6], [26], population clustering [22], and subspace tracking [15]. In many studies, the leverage scores, statistical measurements to sample the data, are also computed based on the SVD [12]. Hence, the SVD has the potential to address the variety and veracity of the modern data sets.

Unfortunately, the traditional approaches to computing the partial SVD access the data several times (e.g., block Lanczos [9]). This is a significant drawback on a modern computer, where the data access has become significantly more expensive compared to arithmetic operations, both in terms of time and energy consumption, and this gap between the communication and computation costs is expected to grow on future computers [8], [11]. To address this hardware trend, a random sampling algorithm [12] has been gaining attention since, compared to the traditional approaches, it often computes the SVD with fewer data accesses. To test its potential, we have developed a sparse solver based on the random sampling [32]. Though such a solver has the potential to efficiently compute the SVD on a modern computer, there are still several obstacles that need to be overcome. For example, due to the sparsity structure of the matrix (i.e., its irregular sparsity pattern and the power-law distribution of its nonzeros), though the sampling algorithm may require only a small number of data accesses, each data access can be expensive, requiring a significant amount of communication. Several techniques to avoid such communication have been proposed [14]. However, these techniques may not be effective for computing the SVD of the modern data because the particular sparsity structure of the matrix often leads to a significant overhead associated with the communication-avoiding techniques [32].

To address the challenges of computing the SVD of the modern data, in this paper, we propose sampling algorithms to update, instead of recompute, the partial SVD as the changes are made to the data set. This is an attractive approach since, compared to recomputing it from scratch, the SVD can be updated more efficiently. Moreover, in some applications, recomputing the SVD may not be possible because the original data is no longer available. At the same time, in modern applications, the existing data are constantly updated and new data is being added. Hence, the size of the update is significant even though it is much smaller than the massive data that has been compressed. Such applications with the rapidly changing data include the communication and electric grids, transportation and financial systems, personalized services on the internet, particle physics, astrophysics, and genome sequencing [5]. Therefore, an efficient updating algorithm is needed to address the large volume and high velocity of the changing data.

There are four main contributions of the paper. First, we introduce three different sampling algorithms to update the partial SVD, and analyze their communication and computation costs. Second, to study the effectiveness of the sampling algorithms, we present case-studies with two particular applications (i.e., LSI and population clustering). Our case-studies using the data sets from real applications demonstrate that the sampling algorithms converge quickly to obtain the desired accuracy of SVD (e.g., three passes over the data sets). Third, we compare the performance of the sampling algorithms with that of the state-of-the-art updating algorithm on a hybrid CPU/GPU cluster. Our performance results demonstrate that the sampling algorithms can obtain significant speedups (e.g., the speedups of up to $14.1\times$ in our experiments). Finally, we present our effort to improve the performance of the

SC15, November 15-20, 2015, Austin, Texas, USA
978-1-4799-5500-8/14/$31.00

sampling algorithms based on the communication-avoiding technique proposed in [16]. Our performance results on a hybrid CPU/GPU cluster show the potential of the implementation. In the final section, we list our current efforts to improve the robustness of this communication-avoiding implementation.

The rest of the paper is organized as follows. After listing related work in Section II, we describe the state-of-the-art updating algorithm and propose the sampling algorithms for updating the partial SVD in Section III. Then, in Section IV, we present our case-studies of using the sampling algorithms for LSI and population clustering. Next, in Section V, we describe our implementation of the sampling algorithms on a hybrid CPU/GPU cluster, and in Section VI, we analyze their computational and communication complexities. Finally, in Section VII, we present our performance studies on a hybrid cluster. The communication-avoiding variant of the sampling algorithm and the final remarks are listed in Sections VIII and IX, respectively.

All of our experiments were conducted on the Tsubame Computer at the Tokyo Institute of Technology[1]. Each of its compute nodes consists of two six-core Intel Xeon CPUs and three NVIDIA Tesla K20Xm GPUs. We compiled our code using the GNU `gcc` version 4.3.4 compiler and the CUDA `nvcc` version 6.0 compiler with the optimization flag `-O3`, and linked it with Intel's Math Kernel Library (MKL) version xe2013.1.046.

II. Related Work

Several algorithms have been proposed to update the partial SVD for the latent semantic indexing. For example, the "fold-in" algorithm [2] is an efficient algorithm, but the precision of a query result could deteriorate as more updates are incorporated into the SVD [29], [30]. The current state-of-the-art algorithm is the "updating" algorithm proposed by Zha and Simon [36], and it has been shown to obtain the same precision as that of the recomputed partial SVD. To reduce the computational cost of the updating algorithm, the "fold-up" algorithm [29] is based on the fold-in algorithm, but periodically discards the updates and recomputes them using the updating algorithm. When the SVD is continually updated with small changes, the fold-up is an efficient approach and obtains the same accuracy as the updating algorithm. We compare the performance of our sampling algorithms with the updating algorithm, and demonstrate that the sampling algorithms can potentially replace the updating algorithm in the fold-up algorithm.

Recently, a column-wise Lanczos was used to update the SVD [30]. Their experimental results in MATLAB demonstrated the efficiency of the algorithm compared to the updating algorithm. However, compared to the sampling algorithms proposed in this paper, Lanczos would require more data accesses to generate the projection subspace of the same dimension. Hence, on a large scale computer, the Lanczos would likely suffer from the greater communication latency. To compute the SVD on a single compute node with multiple GPUs, we have previously compared the performance of the sampling algorithm with the block Lanczos, which often requires fewer data accesses than the column-wise Lanczos, and with a communication-avoiding variant of the block Lanczos [32]. In that study, when the sampling algorithm requires a small number of iterations, even on a single compute node with a marginal inter-GPU communication cost, the sampling algorithm was often more efficient. In addition, compared to the updating algorithm, the projection scheme proposed in [30] reduces the serial bottleneck of solving the projected problem, but it could still be significant (in terms of both time and memory). Our performance results on a hybrid CPU/GPU cluster demonstrate that our projection scheme further reduces the bottleneck, obtaining a significant speedup, while maintaining a similar accuracy.

In recent years, several researchers have demonstrated the ability of the sampling algorithms to analyze the emerging large data [12], [22]. This paper is an extension of these studies to adapt the sampling algorithm for updating SVD and to study its performance on a hybrid CPU/GPU cluster.

A few software packages exist for computing partial SVD. These packages are often developed by researchers from a single discipline and optimized for their specific needs (e.g., SLEPc from linear algebra and GraphLab from graph analysis[2]). In addition, they do not provide a functionality to update SVD. Our ultimate goal is to develop a flexible interface to efficient and robust SVD solvers for a wide range of applications by combining techniques from linear algebra, randomized algorithms, and high performance computing.

III. Algorithms

We assume that a rank-k approximation A_k of the m-by-n matrix A has been computed as $A_k = U_k \Sigma_k V_k^T$, where Σ_k is a k-by-k diagonal matrix whose diagonal entries approximate the k largest singular values of A, and the columns of U_k and V_k approximate the corresponding left and right singular vectors, respectively. Then, we consider the updating problem of computing a k-rank approximation of an m-by-$\widehat{n}$ matrix $\widehat{A}$,

$$\widehat{A} \approx \widehat{U}_k \widehat{\Sigma}_k \widehat{V}_k, \tag{1}$$

where $\widehat{A} = [A, D]$ and an m-by-d matrix D represents the new sets of the sparse columns being added to A.[3] In many cases, D is tall-skinny, having more rows than the columns (i.e., $m \gg d$).

All the algorithms studied in this paper belong to a class of subspace projection methods which are based on the following three steps:

[1] http://tsubame.gsic.titech.ac.jp

[2] http://slepc.upv.es and http://graphlab.org

[3] This was referred to as the *updating-document* problem [2]. Two other problems, *updating-terms* and *term-weight-correction* problems, were considered, which adds new rows to the matrix A and updates the nonzero values of A, respectively. Though we only focus on the updating-document problem, much of our discussion can be extended to the other two problems.

Generate projection subspaces P and Q
for $j = 1, 2, \ldots, s$ do
1. Orthogonalize $\widehat{Q}$
   $QR := \widehat{Q}$
2. Sample range of A
   $\widehat{P} := AQ$
3. Orthogonalize $\widehat{P}$
   $PB := \widehat{P}$
4. Prepare to iterate (sample range of A^T)
   if $j < s$ then
     $\widehat{Q} := A^T P$
   end if
end for

Fig. 1. Sampling algorithm based on power iterations.

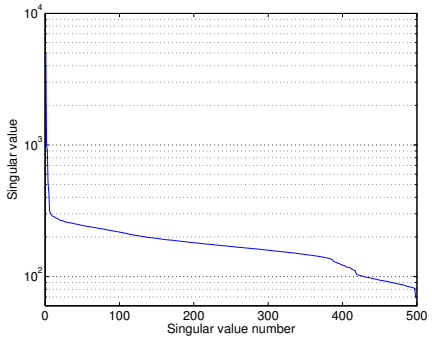


Fig. 2. Singular values of the 116565-by-499 SNP matrix from Section IV-B.

1) Generate a pair of $k + \ell$ orthonormal basis vectors $\widehat{P}$ and $\widehat{Q}$ that approximately span the ranges of the matrices $\widehat{A}$ and $\widehat{A}^T$, respectively,
$$\widehat{A} \approx \widehat{P}\widehat{Q}^T,$$
where ℓ is referred to as an oversampling parameter [12] and selected to improve the performance or robustness of the algorithm.
2) Use a standard deterministic algorithm to compute SVD of the projected matrix B,
$$B = X\widehat{\Sigma}Y^T, \tag{2}$$
where $B = \widehat{P}^T\widehat{A}\widehat{Q}$.
3) Compute the approximation to the partial SVD of $\widehat{A}$,
$$\widehat{A}_k \approx \widehat{U}_k\widehat{\Sigma}_k\widehat{V}_k^T, \tag{3}$$
where $\widehat{U}_k = \widehat{P}X_k$ and $\widehat{V}_k = \widehat{Q}Y_k$.

In this section, we describe algorithms that generate the basis vectors $\widehat{P}$ and $\widehat{Q}$.

A. Sampling Algorithm

To compute the partial SVD of a general sparse matrix A, our implementation of the sampling algorithm is based on the normalized power iterations [20]. Figure 1 shows the pseudocode of the algorithm. The input matrix $\widehat{Q}$ represents the random sampling and projection applied to the matrix A. Though there are several sampling schemes, for all the experiments in this paper, we used the Gaussian random vectors for $\widehat{Q}$, for which extensive theoretical work has been established [12]. When the singular values of A decay slowly (which we typically see for matrices from data-rich fields), we perform the power iterations to reduce the noise and improve the quality of the approximation. The basis vectors are orthonormalized in order to maintain the numerical stability during the iteration, and the projected matrix B is computed as a by-product of the orthogonalization process.

B. Updating Algorithm

To compute the projection basis for updating SVD, the updating algorithm [36] first orthogonalizes the new set of the columns D against the current left singular vectors U_k,
$$\widehat{D} := D - U_k(U_k^T D).$$
Then, the resulting vectors $\widehat{D}$ are orthonormalized based on their QR factorization,
$$QR := \widehat{D},$$
where Q is the m-by-d orthonormal matrix (i.e., $Q^TQ = I$), and R is a d-by-d upper-triangular matrix.

The basis vectors are then given by
$$\widehat{Q} = \begin{bmatrix} U_k, & Q \end{bmatrix} \quad \text{and} \quad \widehat{P} = \begin{bmatrix} V_k & 0 \\ 0 & I_d \end{bmatrix}, \tag{4}$$
where I_d is a d-by-d identity matrix, and the $(k+d)$-by-$(k+d)$ projected matrix B is given by
$$\begin{aligned} B &\equiv \widehat{Q}^T[A_k, D]\widehat{P} \\ &= \begin{bmatrix} \Sigma_k & U^TD \\ & R \end{bmatrix}. \end{aligned} \tag{5}$$

In practice, to reduce the unfeasibly large amount of memory required to store the m-by-d dense vectors Q, the SVD is incrementally updated by adding a subset of the new columns D at a time. Though the memory requirement and the computational cost of the orthogonalization are reduced, all the columns of D are still orthogonalized against U_k. In addition, for each block of D, the corresponding SVD of the projected matrix must be computed, and the approximation U_k and V_k must be updated. It has also been reported that the incremental update can increase the runtime of the algorithm because the required computational kernels often obtain lower performance as they operate on smaller matrices [36], [30]. Finally, the quality of the approximation may degrade when the SVD is incrementally updated [29], [30].

C. Sampling Algorithms to Update SVD

For updating SVD, we propose the following three sampling schemes to generate the projection basis vectors $\widehat{P}$ and $\widehat{Q}$:

a) Sample-1: perform the power iteration on the m-by-$(n + d)$ matrix,
$$\widehat{A}_k = [A_k, D],$$
where $A_k = U_k\Sigma_kV_k$. This generates the pair of r basis vectors, $\widehat{P}$ and $\widehat{Q}$, that approximate the dominant singular vectors of the matrix $\widehat{A}_k$. It has been shown [36] that many

matrices for LSI have so-called approximate low-rank-plus-shift structures, and for this type of matrix, the partial SVD of $\widehat{A}_k$ accurately approximates the partial SVD of $\widehat{A}$. Figure 2 shows the singular values of the SNP matrix used for population clustering studies in Section IV-B. This matrix also has this approximate low-rank-plus-shift structure, where the singular values decay quickly to a constant value. We refer to this sampling scheme as "Sample-1."

When the $k+d$ vectors $\widehat{P}$ in (4) are used as the starting vectors, after one iteration, Sample-1 generates the same projection basis vectors, $\widehat{P}$ and $\widehat{Q}$, as the updating algorithm. Hence, Sample-1 provides a general sampling framework, and we used it to study the effectiveness of the random sampling to compress the desired information into a small projection subspace (e.g., in our experiments, $r = k + \ell$ and $\ell \ll d$).

b) Sample-2: apply the power iterations to the deflated matrix,

$$(I - U_k U_k^T)D,$$

and generate the basis vectors Q. Then, we use the projection basis vectors $\widehat{P}$ and $\widehat{Q}$ defined by Eq. (4). The same projection subspace was used in [30], whereby instead of sampling, the column-wise Lanczos is used to generate Q. It has been shown that $\widehat{P}$ is the optimal right subspace for Q_d, and U_k is used to approximate U_d. We refer to this as "Sample-2."

c) Sample-3: apply the power iterations to the same deflated matrix as Sample-2, but let the right basis vectors $\widehat{Q}$ to be the $\widehat{n}$-by-$(k+\ell)$ matrix,

$$\widehat{Q} = \begin{bmatrix} V_k & 0 \\ 0 & \widehat{V}_\ell \end{bmatrix},$$

while the right projection subspace of Sample-2 is of dimension $k+d$ (i.e., $d \gg \ell$). The cost of computing the SVD of B is significantly reduced using the dominant singular vectors V_ℓ in the right projection subspace. This is called "Sample-3."

IV. Case Studies

In Section VI, we analyze the computational and communication complexities of the proposed sampling algorithms. We then study their performance on a hybrid CPU/GPU cluster in Section VII. Proceeding to these studies, in this section, we demonstrate the potential of the sampling algorithms to compress the desired information into a small projection subspace with a small number of data accesses, focusing on a powerful statistical analysis tool, the principal component analysis (PCA) [3]. In PCA, a multidimensional dataset is projected onto a low-dimensional subspace such that related items are close to each other in the projected subspace. We examine two particular real-world applications of PCA, Latent Semantic Indexing (LSI) and population clustering.

A. Latent Semantic Indexing

For information retrieval by text mining [25], a variant of PCA, Latent Semantic Indexing (LSI) [6], has been shown to effectively address the ambiguity caused by the synonymy or polysemy, which are difficult to address using a traditional lexical-matching [18]. To study the effectiveness of the sampling algorithm, in this section, we apply the updated SVD to LSI. The test matrices are the term-document matrices generated using the Text to Matrix Generator (TMG) and the TREC dataset[4], and are preprocessed using the `lxn.bpx` weighting scheme [17]. These are the standard test matrices and were used in the previous studies [36], [30].

	Total number of documents, $n+d$							
Method	700	800	900	1000	1100	1200	1300	1400
Recomp.	35.6	40.8	40.2	45.3	44.9	44.0	43.6	41.7
Update	35.6	40.8	40.4	45.6	44.9	44.1	43.9	42.2
Update-inc	35.6	40.8	40.4	45.6	44.9	43.9	44.0	42.6
Sample-1	35.6	40.0	40.4	45.6	44.5	44.1	43.6	41.8
Sample-2	35.6	36.7	40.4	45.3	44.4	43.8	43.7	42.0
Sample-3	35.6	36.7	40.4	45.3	44.4	43.6	43.4	41.7

(a) $k = 200$.

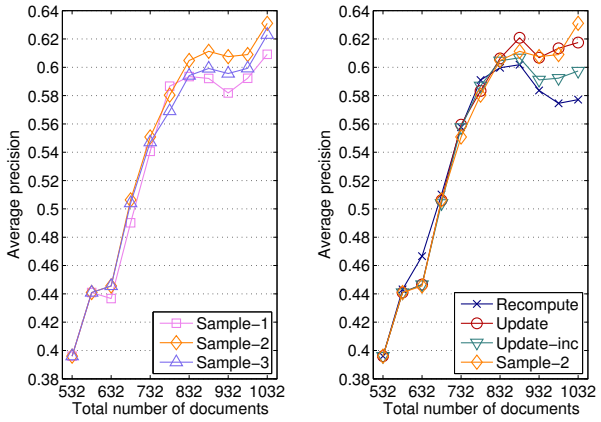
	Total number of documents, $n+d$							
Method	700	800	900	1000	1100	1200	1300	1400
Recomp.	26.7	30.9	32.0	32.5	32.7	31.3	30.8	29.8
Update	26.7	29.8	30.1	30.7	31.5	30.7	30.4	29.7
Update-inc	26.7	29.8	30.1	30.6	30.9	30.1	29.8	29.5
Sample-1	26.7	29.0	29.9	31.9	31.9	30.9	29.5	28.6
Sample-2	26.7	29.6	29.6	30.0	31.0	30.1	30.0	29.7
Sample-3	26.7	29.6	28.2	28.2	27.9	27.4	26.8	25.8

(b) $k = 50$.

Fig. 3. Average 11-point interpolated precision for 6916-by-1400 `CRANFIELD` matrix with 225 queries, $n = 700$.

Figure 3 compares the average 11-point interpolated precisions [17] after adding different numbers of documents from the `CRANFIELD` matrix. Specifically, we first computed the rank-k approximation of the first 700 documents by 20 iterations of sampling. Then, the table shows the average precision after new columns are added (e.g., under the column labeled "1000," 300 documents were added). To recompute the partial SVD of $\widehat{A}$, we performed 20 iterations of sampling, while the sampling algorithms used the oversampling parameter set to be $\ell = k$ (i.e., $r = 2k$), and performed two iterations that access the matrix three times. Since the basis vectors $\widehat{P}$ and $\widehat{Q}$ approximate the ranges of $\widehat{A}$ and $\widehat{A}^T$, respectively, the sampling algorithms access the matrix at least twice. Then, they access the matrix one more time to compute the projected matrix B. We let the incremental update algorithm (Update-inc) add $k+\ell$ columns at a time such that it requires about the same amount of memory as the sampling algorithms. All the results were computed using one GPU (see Sections V for the description of our implementation). We have verified the precisions of recomputing SVD by comparing them against the precisions obtained using the dense SVD of the matrix $\widehat{A}$ in MATLAB. We see that with only three data passes, the sampling algorithms obtained similar precisions as those of the updating algorithm. Figure 4 shows the similar results for the `MEDLINE` matrix. In some cases, the updating and sampling algorithms obtained higher precisions than recomputing the SVD, while the precisions of the incremental update slightly deteriorated at the end. Such phenomena were also reported in the previous studies [29], [36].

[4]http://scgroup20.ceid.upatras.gr:8000/tmg, http://ir.dcs.gla.ac.uk/resources



(a) Different sampling schemes. (b) Sampling/updating algorithms.

Fig. 4. Average 11-point interpolated precision for 5735-by-1033 `MEDLINE matrix` with 30 queries ($k = 50$).

	Update-inc	Sample-1
1	The World Is Not Enough	Mission to Mars
2	Mrs. Doubtfire	The World Is Not Enough
3	Mission: Impossible	Armageddon
4	Die Another Day	Crimson Tide
5	The 6th Day	Mission: Impossible
6	Mission to Mars	Die Another Day
7	The Mummy	Entrapment
8	Die Hard 2: Die Harder	Patriot Games
9	Charlie's Angels	Die Hard 2: Die Harder
10	The Santa Clause	Men of Honor

Fig. 5. Query results for "Tomorrow Never Dies" on 3 GPUs.

We have also conducted the same experiments using the `Netflix` matrix[5], which stores the user scores of the movies, where the matrix columns and rows represent movies and users, respectively (i.e., 2,649,429 users and about 5,654 rankings per movie). We computed the rank-30 approximation of 5,000 movies by 20 iterations of random sampling, then ran two iterations of Sample-1 to update SVD with 5,000 new movies. For comparison, we incrementally updated the SVD by adding 60 movies at a time. Figure 5 shows the results for the query of "Tomorrow Never Dies." We see that with only two iterations, Sample-1 returned reasonable recommendations for the query. Though there are a couple of movies that seem to be irrelevant using the updating algorithm, these are popular movies, and it is reasonable to imagine that the users who liked "Tomorrow Never Dies" also liked these two movies.

B. Population Clustering

PCA has been successfully used to extract the underlying genetic structure of human populations [21], [23], [24]. To study the potential of the sampling algorithms, we used our algorithms to update the SVD, when a new population is added to the population dataset from the HapMap project[6]. Figure 6

[5]http://netflixprize.com

[6]http://hapmap.ncbi.nlm.nih.gov

	JPT+MEX	+ ASW	+ GIH	+ CEU
Recompute	1.00	1.00	1.00	0.97
No update	1.00	0.81	0.84	0.67
Update-inc	1.00	1.00	0.89	0.70
Sample-1	1.00	0.95	0.92	0.86

Fig. 6. Average correlation coefficients of population clustering, where 83 African ancestry in south west USA (ASW), 88 Gujarati Indian in Houston (GIH), and 165 European ancestry in Utah (CEU) are incrementally added to the 116, 565 SNP matrix of 86 Japanese in Tokyo and 77 Mexican ancestry in Los Angeles, USA (JPT and MEX).

shows the correlation coefficient of the resulting population cluster, which is computed using the k-mean algorithm of MATLAB in the low-dimensional subspace given by the dominant left singular vectors. We randomly filled in the missing data with either -1, 0, or 1 with the probabilities based on the available information for the SNP. We let Sample-1 iterate twice, and with only the three data passes, Sample-1 improved the clustering results, potentially reducing the number of times the SVD must be recomputed.

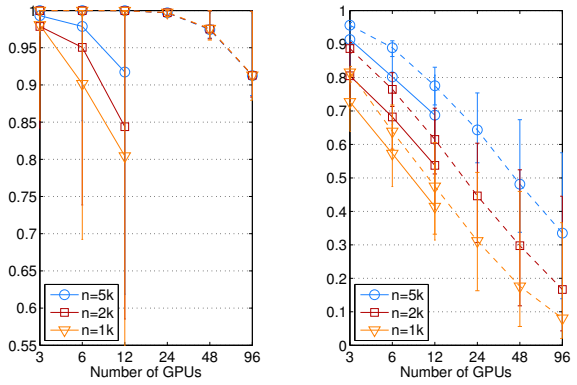
V. Implementation on a Hybrid CPU/GPU Cluster

Since the computational costs of the sampling algorithms are dominated by the cost of generating the projection basis vectors, $\widehat{P}$ and $\widehat{Q}$, we accelerate this step using GPUs, while the SVD of the projected matrix B is redundantly computed by each MPI process on CPU. On a hybrid CPU/GPU cluster with multiple GPUs on each node, each MPI process could manage multiple GPUs on the node to combine or avoid the MPI communication to the GPUs on the same node. However in our previous studies [33], the overhead for each MPI to manage multiple GPUs (e.g., sequentially launching GPU kernels on multiple GPUs) outweighed the benefit of avoiding the intra-node communication, for which many MPI implementations are optimized. Hence, for this paper, we use one MPI process to manage a single GPU.

The two main computational kernels of our SVD solver are the sparse-matrix matrix multiply (SpMM) and the orthogonalization. In the following subsections, we describe our implementations of these two kernels.

A. Sparse Matrix Matrix Multiply

To update SVD by adding new columns D on a hybrid cluster, our first implementation distributes both D and D^T among the GPUs in 1D block row formats. The basis vectors $\widehat{P}$ and $\widehat{Q}$ are then distributed in the same formats. To perform SpMM, each GPU first exchanges the required non-local vector elements with its neighboring GPUs. This is done by first copying the required local elements from the GPU to the CPU, then performing the point-to-point communication among the neighbors using the non-blocking MPI (i.e., `MPI_Isend` and `MPI_Irecv`), and finally copying the non-local vector elements back to the GPU. Then, each GPU computes the local part of the next basis vectors using the CuSPARSE SpMM in the compressed sparse row (CSR) format. This was an efficient communication scheme in our previous studies to develop a linear solver [33], where the coefficient matrix



(a) SpMM with D. (b) SpMM with D^T.

Fig. 7. Number of vector elements needed for SpMM by a GPU, relative to the total number of elements in the global vector. The solid and dashed lines show the results with and without hypergraph partitioning, respectively. PaToH ran out of CPU memory partitioning among more than 12 GPUs.

arising from scientific or engineering simulation is often sparse and structured (e.g., with three-dimensional embedding). Unfortunately, sparse matrices originating from the modern data sets such as social networks and/or commercial applications have irregular sparsity structures, and have wide ranges of numbers of nonzeros per row. In fact, they often exhibit power-law distributions of nonzeros as they result from scale-free graphs. As a result, this point-to-point communication with all the neighbors at once could lead to a prohibitively large communication buffer (see Figure 7).

To reduce the size of the communication buffer required for SpMM with D, our second implementation is based on the point-to-point blocking MPI (i.e., `MPI_Send` and `MPI_Recv`), hence reusing the communication buffer. Unfortunately, this was not scalable. To alleviate the problem, we have tested the following collective communication schemes:

- `MPI_Bcast`: Each MPI process broadcasts its local elements. The buffer is reused to receive each message, which is then unpacked into the internal storage used by the solver. Even though this may perform unnecessary data exchanges, the communication time can be greatly reduced when the matrix has the power-law distribution.
- `MPI_Allgatherv`: With both broadcast and all-gatherv, each process sends its local vector elements, which are needed by at least one of its neighbors, to all the processes. Though this all-to-all approach requires the buffer to store the receiving messages from all the processes at once, it could obtain a significant speedup over the broadcast.
- `MPI_Alltoallv`: All MPI processes exchange the messages of different sizes with their neighbors at once. Hence, this approach may be more effective than `MPI_Allgatherv`, when there are significant differences in the vector elements needed by the MPI processes. This is not the case with the power-law distribution (see Figure 7(a)). In addition, this requires storing the messages sent to a different neighbor separately.

	Send/Recv	Bcast	Allgatherv
3 GPUs, 2 neighbors/GPU			
Time (ms)	7.5 (7.1)	6.6	2.5
Buffer (MB)	0.8	0.8	2.4
12 GPUs, 11 neighbors/GPU			
Time (ms)	229.5 (257.5)	84.1	36.0
Buffer (MB)	0.2	0.2	2.4

(a) $(n, d) = (5000, 5000)$ with $k = 30$.

	Send/Recv	Bcast	Allgatherv
3 GPUs, 2 neighbors/GPU			
Time (ms)	9.9 (10.0)	9.1	2.8
Buffer (MB)	1.6	1.6	4.8
12 GPUs, 11 neighbors/GPU			
Time (ms)	488.1 (552.9)	190.3	85.3
Buffer (MB)	0.4	0.4	4.8

(b) $(n, d) = (10000, 3000)$ with $k = 60$.

Fig. 8. Communication statistics of SpMM with D for `Netflix` matrix. For Send/Recv, the times in parentheses are without hypergraph partitioning.

	Send/Recv	Bcast	Allgatherv	Allreduce
3 GPUs, 2 neighbors/GPU				
Time (ms)	312.9 (271.8)	239.7	145.9	2.9
Buffer (MB)	72.8	74.7	223.7	2.4
12 GPUs, 11 neighbors/GPU				
Time (ms)	1012.0 (977.1)	355.9	155.3	5.8
Buffer (MB)	15.2	18.9	226.3	2.4

(a) $(n, d) = (5000, 5000)$ with $k = 30$.

	Send/Recv	Bcast	Allgatherv	Allreduce
3 GPUs, 2 neighbors/GPU				
Time (ms)	572.3 (533.0)	543.6	260.7	3.8
Buffer (MB)	75.3	76.2	228.6	0.6
12 GPUs, 11 neighbors/GPU				
Time (ms)	1737.0 (1740.0)	688.9	264.4	6.7
Buffer (MB)	17.4	19.1	229.4	0.6

(b) $(n, d) = (10000, 3000)$ with $k = 60$.

Fig. 9. Communication statistics of SpMM with D^T for `Netflix` matrix.

Figure 8 shows the performance of the above communication schemes for updating the `Netflix` matrix (see Sections IV and VII for our experimental setups). To update the SVD by adding the new columns D to A, we distribute the rows of D in the same format as A, while the rows of D^T are evenly distributed among the GPUs. For the point-to-point communication, we tested two partitioning schemes. In the first scheme, we used PaToH[7] to distribute the rows of A and A^T based on a hypergraph algorithm [13]. In the second scheme, we simply partitioned A and A^T such that each GPU has similar numbers of rows. For the collective communication, we used the second scheme which obtains the load balance. Due to the power-law distribution, using the hypergraph algorithm seems to have negligible effects on the communication cost, especially since we do not require a large number of GPUs due to the their high computing power. We clearly see that `MPI_Allgatherv` requires much less time.

In this paper, we focus on the tall-skinny matrix D, which is the case in the applications we considered. To perform

[7]http://bmi.osu.edu/umit/software.html#patoh. Our implementation using METIS* was often slower. *http://glaros.dtc.umn.edu/gkhome/metis/

SpMM with D^T, our next implementation keeps the input and output vectors, $\widehat{P}$ and $\widehat{Q}$, in the 1D block row distribution, but uses D^T in the 1D block column. Since the columns of D^T are the same as the rows of D on each GPU, we avoid the need to separately store D^T and D.[8] In this implementation, each GPU first computes SpMM with its local parts of D^T and $\widehat{P}$, and then copies the partial result to the CPU. Then, the MPI process computes the final result $\widehat{Q}$ by a global reduce, and copies its local part back to the GPU. Hence, this requires each MPI process to store the global vectors $\widehat{Q}$. However, when D^T has the power-law distribution, performing SpMM with D^T in the 1D block row requires each GPU to store the much longer global vectors $\widehat{P}$ (see Figure 7(b), where the hypergraph partitioning reduced the median and minimum numbers of required vector elements, but not the maximum which is on the critical path). Figure 8 clearly demonstrates the advantage of this all-reduce communication. Furthermore, partitioning D^T in the 1D block column often led to a higher performance of SpMM on each GPU as the local submatrix becomes more square than tall-skinny.

Though other techniques may improve the performance of SpMM (e.g., sub-communicator with 2D distribution [4]), in this paper, we use `MPI_Allgather` and `MPI_Allreduce` for SpMM with D and D^T, respectively.[9] In other words, we perform the sparse operations on each GPU, but rely on the dense communication among them (which is effective for the matrix with the power-law distribution as shown above). Though our MPI implementation may not be optimal, in this paper, we focus on developing an efficient solver for updating the SVD and studying the performance of this specific implementation, and not on developing an optimal communication scheme for a specific matrix structure on a specific hardware. In addition, we partition the matrix based on the natural matrix ordering (i.e., the second partitioning scheme above, which is effective for the collective communication). In a real application, repartitioning/redistributing the data may not be feasible. Our solver only requires the result of SpMM with the local data. Then, the final result can be computed through communication, which tends to be the bottleneck. Our objective of the paper is to develop an efficient solver with a small communication cost.

B. Orthogonalization Kernels

In this paper, we use classical Gram-Schmidt (CGS) [10] to orthogonalize a set of vectors against another set of vectors (block orthogonalization, or BOrth in short) and Cholesky QR (CholQR) [28] to orthogonalize the set of vectors against each other. In our previous study, these algorithms obtained great performance on multiple GPUs on a single compute node [31] or on a hybrid CPU/GPU cluster [33]. To orthonormalize $\widehat{Q}$ by CholQR, each GPU first computes the block dot products of its local vectors (i.e., $G^{(g)} := \widehat{Q}^{(g)T}\widehat{Q}^{(g)}$, where $\widehat{Q}^{(g)}$ is the local matrix of $\widehat{Q}$ distributed to the g-th GPU, while n_g is the number of available GPUs). Then, the resulting matrix $G^{(d)}$ is copied to the CPU, and the MPI process computes the Gram matrix via a global reduce (i.e., $G = \sum_{g=1}^{n_g} G^{(g)}$). Next, each MPI process redundantly computes the Cholesky factorization of the Gram matrix on the CPUs (i.e., $RR^T := G$), and copies the Cholesky factor R to its local GPU. Finally, each GPU orthogonalizes the local part of the basis vectors through a triangular solve (i.e., $Q^{(g)} := \widehat{Q}^{(g)}R^{-1}$). CGS is implemented similarly, and is based on the matrix-matrix multiplies on each GPU. All of our orthogonalization kernels use the optimized dense GPU kernels developed earlier [31], [34].

When the matrix $\widehat{Q}$ is ill-conditioned, CholQR can suffer from numerical instability [28]. In our experiments with the sampling algorithms, CholQR was stable with full reorthogonalization. The reliability of CholQR may be explained by the observation that the matrices used in our experiments had the approximate low-rank-plus-shift structure, but they were not ill-conditioned (see Section III-C). On the other hand, the updating algorithm needs to orthogonalize the columns of D, some of which could be empty. To overcome the numerical difficulty, the updating algorithm uses the Singular Value QR (SVQR), which computes the upper-triangular matrix R by first computing the SVD of the Gram matrix, $U\Sigma U^T := G$, followed by the QR factorization of $\Sigma^{\frac{1}{2}}U^T$. When the matrix contains an empty columns, the Cholesky factorization of the Gram matrix will fail, while SVQR overcomes this numerical challenge. Compared to the Cholesky factorization, computing the SVD and QR factorization of the Gram matrix is computationally more expensive. However, the dimension of the Gram matrix is much smaller than that of the input matrix $\widehat{P}$ or $\widehat{Q}$ (i.e., $d \ll m$). Hence, SVQR performs about the same number of flops as CholQR, using the same BLAS-3 kernels, and requires only one global reduce, obtaining similar performance as CholQR.

VI. Computational and Communication Costs

To analyze the computational costs of the proposed sampling algorithms, we separately consider the following two stages of the algorithms: 1) generating the projection basis vectors $\widehat{P}$ and $\widehat{Q}$, and 2) computing SVD of the projected matrix B and generating the singular vectors $\widehat{U}_k$ and $\widehat{V}_k$.

Figure 10 lists the numbers of floating point operations (flops) required by the first stage, where dense matrix-matrix multiply (GEMM) is used for BOrth. When Sample-1 generates ck basis vectors and iterates h times, it requires about $c(2h-1)\times$ more flops for SpMM than Update-inc. On the other hand, Update-inc performs about $\frac{2d}{hck(2+c)}\times$ more flops for orthogonalizing these basis vectors. In our experiments, we let Sample-1 iterate twice with the oversampling parameter set to be equal to the rank of the approximation (i.e., $h = 2$ and $c = 2$). Hence, Sample-1 performs about $6\times$ more flops for SpMM, while Update-inc requires $\frac{d}{8k}\times$ more flops for the orthogonalization. Clearly, when a large number of columns must be added, the orthogonalization

[8] CuSPARSE SpMM in CSC uses atomic operations. To guarantee the reproducibility of the results, we explicitly stored D^T in the CSR format.

[9] Our implementation uses either the point-to-point or the collective communication depending on the average number of neighbors. For all the experiments in this paper, the collective communication was selected.

Method	SpMM	GEMM	CholQR
Update	$2k \cdot nnz(D)$	$2mkd$	$2md^2$
Update-inc	$2k \cdot nnz(D)$	$2mkd$	$2mkd$
Sample-1	$4r \cdot nnz(D)$	$4(m+n)kr$	$2(m+n)r^2$
Sample-2,3	$4r \cdot nnz(D)$	$4mkr$	$2(m+n)r^2$

Fig. 10. Flop counts for first stage (generation of basis vectors). For sampling algorithms, we show the flop counts for one iteration, where r is the number of basis vectors generated (e.g., in our experiments, $r = 2k$). For reorthogonalization, the matrix D becomes fully dense (i.e., $nnz(D) = md$).

Method	Compute SVD of B	Generate U_k and V_k
Update	$O((k+d)^3)$	$O((m+n)k(k+d))$
Update-inc	$O(\frac{d}{\widehat{d}}(k+\widehat{d})^3)$	$O((m+n)kd)$
Sample-1	$O(r^3)$	$O((m+n)kr)$
Sample-2	$O((k+d)(k+r)^2)$	$O((m+n)k(k+r))$
Sample-3	$O((k+r)^3)$	$O((m+n)k(k+r))$

Fig. 11. Flop counts for second stage (updating partial SVD). Update-inc updates the partial SVD by incrementally adding $\widehat{d}$ columns at a time.

becomes the bottleneck in Update-inc (e.g., $\frac{d}{k} = O(10^2)$ in our performance studies). On the other hand, Sample-1 can be computationally more efficient than Update-inc, but even on a small number of GPUs, its execution time is often dominated by SpMM. In the end, with $c = 2$ and $h = 2$, Sample-1 performs more flops than Update-inc when each column of D has more than $\frac{m}{3.5}(1 - 8\frac{k}{d})$ nonzeros in average. For instance, when $k = 100$, and $d = 1000$ or 5000, Sample-1 performs more flops than Update-inc when D is more than 6% or 24% dense, respectively (e.g., `Netflix` matrix is about 1.1% dense).

Compared to the matrix operation $U_k V_k^T$ performed by Sample-1, there are about twice as many flops in the operation $U_k U_k^T$ of Sample-2 or Sample-3 (i.e, $n \ll m$). However, Table 10 shows that all the sampling algorithms perform about the same number of flops in GEMM. This is because when Sample-2 or Sample-3 applies the matrix operation $D^T(I - U_k U_k^T)$ to the vectors $\widehat{P}$, these vectors are already orthogonal to U_k. Hence, the deflation operation $I - U_k U_k^T$ needs to be applied only after SpMM with D. Nevertheless, to maintain the orthogonality of the basis vectors, in our experiments, we performed the full reorthogonalization. Therefore, compared to Sample-1, both Sample-2 and Sample-3 performed about twice as many flops for GEMM.

Table 11 shows the required flop counts for the second stage. The flop count of computing SVD of B with Sample-2 depends linearly on the number of columns in D, while they depend only on k and r with both Sample-1 and Sample-3. Since our implementation of the sampling algorithms let each MPI process redundantly compute the SVD of the projected matrix B, this serial bottleneck can become significant on a much smaller number of GPUs with Sample-2 (i.e., $k, r \ll d$).

Table 12 shows the required communication costs of the updating and sampling algorithms. On a small number of GPUs, the communication volume of the updating algorithm is typically dominated by that of the orthogonalization. But, as the number of GPUs increases, the cost of SpMM becomes

Method	#words	#messages
Incremental update	$O(d(\widehat{d}+k) + k\sum_j^{d/\widehat{d}} g(D_j))$	$O(\frac{d}{\widehat{d}})$
Sampling-1,2,3	$O((r(d+r) + rg(D))h)$	$O(h)$

Fig. 12. Communication costs of updating and sampling algorithms, where $g(D_j)$ is the total number of the nonlocal vector elements needed for SpMM with the j-th block D_j.

Method	n_g	SpMM(A^T/A)	GEMM	CholQR	SVD
Sample-1	3	153 / 99	8	47	1.5+2.0
Sample-2	3	382 / 99	19	47	6935+3.9
Sample-3	3	153 / 99	22	46	3.6+3.0
Sample-1	12	27 / 66	4.4	17	1.5+0.9
Sample-2	12	65 / 65	7.9	16	7014+2.5
Sample-3	12	27 / 65	8.9	16	3.8+1.3
Sample-1	48	18 / 108	4.0	10	1.6+0.6
Sample-2	48	35 / 107	5.4	11	7035+2.1
Sample-3	48	19 / 108	6.6	10	3.9+0.6

(a) $(n, d) = (5000, 5000), k = 30$.

Method	n_g	SpMM(A^T/A)	GEMM	CholQR	SVD
Sample-1	3	186 / 127	21	141	6.3+5.6
Sample-2	3	466 / 127	57	140	746+9.4
Sample-3	3	187 / 128	66	139	19+9.0
Sample-1	12	36 / 130	8.2	42	6.5+1.9
Sample-2	12	89 / 123	18	41	742+3.7
Sample-3	12	37 / 125	21	41	19+2.8
Sample-1	48	20 / 194	5.6	18	6.5+1.0
Sample-2	48	41 / 194	8.3	18	755+2.2
Sample-3	48	21 / 196	9.6	18	26+1.3

(b) $(n, d) = (10000, 3000), k = 60$.

Fig. 13. Performance of sampling algorithms in milliseconds for `Netflix` matrix. Under "SVD," we show, separately, the time spent for SVD of B and generating singular vectors.

more significant. On the other hand, the communication costs of the sampling algorithms can be dominated by that of SpMM even on a small number of GPUs since the orthogonalization cost is less significant compared to the updating algorithm.

VII. Performance Results

We now compare the performance of the proposed sampling algorithms with that of the updating algorithm on a hybrid CPU/GPU cluster. For our studies, we let the updating algorithm add $k + \ell$ columns at a time, while all the sampling schemes generate $k + \ell$ basis vectors, where $\ell = k$. Hence, all the algorithms require about the same amount of memory. Also, we orthogonalized both $\widehat{P}$ and $\widehat{Q}$ with full reorthgonalization. In some cases [27], the sampling algorithms may only need to orthogonalize $\widehat{Q}$, reducing its orthogonalization cost significantly (i.e., $d \ll m$).

Figure 13 compares the performance of the sampling algorithms. Compared to Sample-1, Sample-3 spends about twice as much time in GEMM due to the full re-orthogonalization used to maintain the orthogonality of the basis vectors $\widehat{P}$ against U_k. Then, compared to Sample-3, Sample-2 spends more time in GEMM because it requires additional SpMM and GEMM to generate its projected matrix B (i.e., $U_k^T D$ and $U_k^T \widehat{P}$). However, the main difference is in the time spent computing the SVD of the projected matrix B and generating the singular vectors. This is because the projection subspace of Sample-2 is greater than that of Sample-3, which is larger

Method	n_g	SpMM(A^T/A)	GEMM	TSQR	SVD	Total
Update-inc	3	95	491	2990	274/282	4212
Sample-1	3	153 / 99	8	47	1.5+2.0	314
Update-inc	12	44	177	1010	275+99	1664
Sample-1	12	27 / 66	4.4	17	1.5+0.9	118
Update-inc	48	60	118	563	276+64	1121
Sample-1	48	18 / 108	4.0	10	1.6+0.6	145

(a) $(n, d) = (5000, 5000), k = 30$.

Method	n_g	SpMM(A^T/A)	GEMM	TSQR	SVD	Total
Update-inc	3	107	446	2513	455+148	3720
Sample-1	3	186 / 127	21	141	6.3+5.6	492
Update-inc	12	28	137	921	452+51	1630
Sample-1	12	36 / 138	8.2	42	6.5+1.9	234
Update-inc	48	29	7.5	546	454+26	1167
Sample-1	48	20 / 194	5.6	18	6.5+1.0	243

(b) $(n, d) = (10000, 3000), k = 60$.

Fig. 14. Performance of updating and sampling algorithms in milliseconds for `Netflix` matrix. Update-inc and Sample-1 use SVQR and CholQR for tall-skinny QR (TSQR), respectively.

Method	n_g	SpMM(A^T/A)	GEMM	TSQR	SVD	Total
Update-inc	12	228.3	4,357	20,570	2,146+1,352	29,080
Sample-1	12	284.8 / 192.6	24.7	172.8	4.6+6.4	691.5
Update-inc	48	177.5	1,266	5,454	2,161+438.4	9,856
Sample-1	48	86.1 / 99.9	8.8	49.6	4.6+2.0	253.2
Update-inc	96	166.1	836.1	3,032	2,147+305.4	6,877
Sample-1	96	70.7 / 94.6	6.7	30.2	6.5+1.2	214.7

(a) $k = 50$.

Method	n_g	SpMM(A^T/A)	GEMM	TSQR	SVD	Total
Update-inc	12	326.0	7,208	25,310	6,081+2,022	41,320
Sample-1	12	574.8 / 377.3	76.6	427.3	20.5+22.3	1,500
Update-inc	48	108.7	1,946	6,445	6,059+588.1	15,420
Sample-1	48	162.2 / 201.3	22.9	112.5	26.4+6.1	535.3
Update-inc	96	77.8	1,110	3,494	6,047+351.3	11,350
Sample-1	96	119.3 / 183.0	14.0	62.5	23.8+0.3	411.4

(b) $k = 100$.

Fig. 15. Strong parallel scaling of updating and sampling algorithms in milliseconds for `Inktomi` matrix with $(n, d) = (30000, 20000)$.

Method	n_g	SpMM(A^T/A)	GEMM	TSQR	SVD	Total
Update-inc	12	50.9	1,092	5,438	557+331.3	7,565
Sample-1	12	73.7 / 59.3	24.1	172.6	5.0+6.4	345.5
Update-inc	24	67.1	1,137	5,262	1,097+359.8	8,071
Sample-1	24	79.4 / 75.7	13.6	90.0	5.6+6.4	271.8
Update-inc	48	177.5	1,266	5,454	2,161+438.4	9,856
Sample-1	48	86.1 / 99.9	8.8	49.6	4.6+2.0	253.2
Update-inc	96	224.2	1,632	6,068	4,257+620.7	13,500
Sample-1	96	116.6 / 191.9	7.2	30.6	9.1+1.3	359.0

Fig. 16. Weak parallel scaling of updating and sampling algorithms in milliseconds for `Inktomi` matrix with the fixed parameters $(m, k) = (2559430, 50)$ and varied parameters $(n, d) = (n_g/12) \times (7500, 5000)$.

than that of Sample-1. Since our implementation lets each MPI process redundantly compute the SVD, compared to Sample-1 or Sample-3, this serial bottleneck became significant in Sample-2 on a smaller number of GPUs.

Figure 14 compares the performance of the sampling and updating algorithms. Clearly, the updating algorithm can spend significantly longer time in the orthogonalization, leading to a great speedup obtained by the sampling algorithms (i.e., the speedups of up to 14.1). At the same time, the speedup decreased on a larger number of GPUs. This is because the sampling time is dominated by SpMM, whose strong parallel scaling suffered from the increasing inter-GPU communication cost for this relatively small-scale matrix. On the other hand, the updating algorithm was still spending most of its time on the orthogonalization which was still compute intensive and scaled over the small number of the GPUs. However, on a larger number of GPUs, compared to the sampling algorithm, the updating algorithm is expected to suffer from the greater communication latency. Figures 15 and 16 show the similar results for the document-document matrix used in a previous LSI study [35]. The matrix row contains 2,559,430 documents, and each column contains about $4,176$ nonzero entries. The weak parallel scaling results, in particular, show the advantages of the sampling algorithm due to its ability to compress the desired information into a small projection subspace using a small number of data passes. For the updating algorithm, the accumulated cost of the SVDs of the $\frac{d}{\widehat{d}}$ projected matrices also became significant.

VIII. Communication-Avoiding Implementation

Our performance results on a hybrid CPU/GPU cluster demonstrated the potential of the sampling algorithms to obtain significant speedups over the updating algorithms. However, the execution time of the sampling algorithms is often dominated by SpMM, and due to the particular sparsity structure of the matrix, its parallel scalability can be quickly limited by the inter-GPU communication. In this section, we describe our effort to address this limitation based on the algorithm [16] that applies the matrix powers to dense vectors in a communication-avoiding fashion, where the matrix is represented as a sparse matrix plus a low-rank matrix. While the algorithm can generate different types of basis vectors, we focus on generating the monomial basis vectors. In addition, we use Sample-3 as our example, but the other sampling algorithms can be extended in the same fashion. In fact, Sample-2 and Sample-3 generate the same basis vectors.

Sample-3 is equivalent to the normalized power iteration applied to the following matrix $\widetilde{A}_k$ with the starting vectors $\widetilde{Q}$:

$$\widetilde{A}_k = \begin{pmatrix} 0 & (I - U_k U_k^T)D \\ D^T & 0 \end{pmatrix} \quad \text{and} \quad \widetilde{Q} = \begin{pmatrix} 0 \\ Q \end{pmatrix}.$$

This coefficient matrix $\widetilde{A}_k$ can be split into the following sparse and low-rank matrices,

$$\begin{aligned} \widetilde{A}_k &= \begin{pmatrix} 0 & D \\ D^T & 0 \end{pmatrix} + \begin{pmatrix} 0 & U\widehat{U}^T \\ 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & D \\ D^T & 0 \end{pmatrix} + \begin{pmatrix} U \\ 0 \end{pmatrix} \begin{pmatrix} 0 & \widehat{U}^T \end{pmatrix} \qquad (6) \\ &= \begin{pmatrix} 0 & D \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} U & 0 \\ 0 & I_d \end{pmatrix} \begin{pmatrix} 0 & \widehat{U}^T \\ D^T & 0 \end{pmatrix}, \qquad (7) \end{aligned}$$

where $\widehat{U} = -D^T U$, and the low-rank matrices are of ranks k and $k + d$ in (6) and (7), respectively. The matrix split (7) is motivated by our implementation which computes SpMM with D^T by the local SpMM on each GPU, followed by the global reduce.

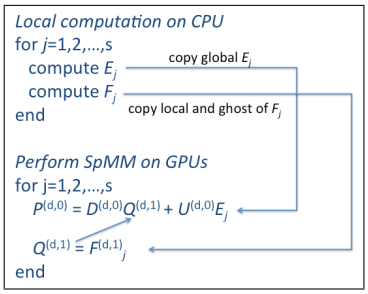
▷ *Perform SpMM on GPUs*
$\widehat{Q} = DQ$
$F_1 = D^T \widehat{Q}$

Local computation
$E_1 = -\widehat{U}^T Q$
$F_1 = F_1 + \widehat{U} E_1$
`for` $j = 2, 3, \ldots, s$ `do`
  $E_j = -\widehat{U}^T F_{j-1}$
  $F_j = \widehat{U} E_j + (D^T D) F_{j-1}$
`end for`

▷ *Generate basis vectors on GPUs*
`for` $j = 1, 2, 3, \ldots, s$ `do`
  `if` $j == 1$ `then`
    $P = \widehat{Q}$
  `else`
    $P = DQ$
  `end if`
  $P = P + U E_j$

  $Q = F_j$
`end for`

(a) Pseudocode.



(b) Communication and data dependency, where the vector $P^{(d,k)}$ consists of the local elements of P, which are distributed to the d-th GPU, and the k-level ghosts that are the non-local elements at most k edges away from the local elements in the adjacency graph of D.

Fig. 17. Matrix powers kernel with the matrix split (7) with initial vectors Q and step size s. Local computation is redundantly performed by each MPI process on the CPU, except the multiply with $D^T D$ which is on the GPU.

n_g	SpMM(D^T/D)	GEMM	MPI(D^T/D)	Total
3	15.1/5.0	0.9	0.29 / 0.15	21.0
12	2.5/1.7	0.3	0.46 / 0.20	4.49

(a) SpMM.

n_g	s	SpMM(D^T/D)	Local	Generate P	MPI(D^T/D)	Total
3	1	15.24 / 4.93	0.2	0.2	0.29 / 0.13	20.5
3	2	7.65 / 2.46	0.3	2.5	0.16 / 0.06	12.9
3	4	3.82 / 1.23	0.4	3.7	0.08 / 0.03	9.1
12	1	2.42 / 1.66	0.2	0.7	0.46 / 0.20	4.54
12	2	1.25 / 0.88	0.3	0.7	0.24 / 0.13	3.14
12	4	0.66 / 0.45	0.4	1.0	0.15 / 0.08	2.47

(b) MPK.

Fig. 18. 100 power iterations in seconds, for `Netflix` matrix with $(n, d) = (5000, 5000)$.

Figure 17 specializes the matrix powers kernel (MPK) [16] for applying s matrix powers based on the matrix split (7). In this implementation, each MPI process redundantly generates the right singular vectors F_j by applying the power iteration on the explicitly-formed Gram matrix. After the right singular vectors are generated, the local parts of the corresponding left singular vectors P are generated by the power iteration without further communication. Since the matrix D is tall-skinny (i.e., $d \ll m$), the generation of the right singular vectors requires much less computation than that needed to generate the left singular vectors. With this implementation, regardless of the sparsity structure of D, an arbitrary number of power iterations can be applied after the initial communication to generate the vectors E_1 and F_1 (i.e., the same communication cost as the single iteration of Sample-3). However, to avoid the communication, each GPU not only requires the Gram matrix, the computation of which requires a global reduce, but it also requires both the local and ghost elements of Q. Figure 18 shows the performance of this implementation

Method	SpMM	GEMM
SpMM	$4s(nnz(D^{(g)}))r$	$2s(k(m+n)/n_g)r$
MPK	$2s(nnz(D^{(g)}) + nnz(D^T D))r$	$2s(k(m/n_g + n)r$

(a) Flop count for g-th GPU.

Method	#messages	#words
SpMM	$4s$	$s(g(D) + g(D^T) + (m+n)r)$
MPK	4	$g(D) + g(D^T) + (m+n)r$

(b) Total communication costs.

Fig. 19. Computational and communication costs to apply s matrix powers.

for updating the `Netflix` matrix. The execution time is reduced not only because the MPI time is reduced but also because SpMM with D^T is replaced with GEMM with $D^T D$, whose dimension is much smaller than that of D^T. Figure 19 summarizes the computational and communication costs of the implementations.

IX. Conclusion

We proposed sampling algorithms to update the partial singular value decomposition (SVD). Our case-studies showed that these algorithms have the potential to obtain the desired accuracy of the updated SVD with a small number of data passes, reducing both the computational and communication costs from those of the state-of-the-art updating algorithm. Our performance results on a hybrid CPU/GPU cluster demonstrated that a significant speedup may be obtained using the sampling algorithms. We also presented our effort to reduce the communication costs of the sampling algorithms. To maintain the numerical stability of the communication-avoiding implementation, we are currently studying different orthogonalization schemes such as the one-sided orthogonalization to orthogonalize only the right-singular vectors during the power iteration [27]. We are also working to improve the performance of the solver (e.g., a parallel SVD of the projected matrix, and an extension of the batched GEMM used in CholQR to a batched sparse GEMM for forming the Gram matrix to setup MPK, or to a batched SpMM with a transpose of a tall-skinny matrix). We focused on the performance of the algorithms on a hybrid CPU/GPU cluster, but we plan to investigate the performance on other architectures. For instance, compared to the hybrid cluster, the costs of SpMM relative to the orthogonalization may be smaller on a CPU cluster, making the sampling algorithms more attractive. We are also conducting the numerical analysis of the proposed sampling algorithms.

Acknowledgments

This research was supported in part by the U.S. Department of Energy Office of Science under Award Number DE-FG02-13ER26137/DE-SC0010042, and the U.S. National Science Foundation under Award Number 1339822. We thank Eugen Vecharynski, Nicholas Knight, and Erin Carson and the members of the DOE EASIR project for helpful discussions, Theo Mary and Osni Marques for providing some of our datasets, and Endo Toshio and Satoshi Matsuoka for providing the access to the Tsubame Computer.

REFERENCES

[1] M. Berry. Large scale singular value computations. *International Journal of Supercomputer Applications*, 6:13–49, 1992.

[2] M. Berry, S. Dumais, and G. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37:573–595, 1995.

[3] C. Bishop. *Pattern recognition and machine learning*. Springer, New York, 2006.

[4] E. Boman, K. Devine, and S. Rajamanickam. Scalable matrix computations on large scale-free graphs using 2D graph partitioning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 50:1–50:12, 2013.

[5] Committee on the Analysis of Massive Data, Committee on Applied and Theoretical Statistics, Board on Mathematical Sciences and Their Applications, Division on Engineering and Physical Sciences, and National Research Council. *Frontiers in Massive Data Analysis*. The National Academies Press, 2013.

[6] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J. Amer. Soc. Info. Sci.*, 41:391–407, 1990.

[7] DOE Office of Science. Synergistic challenges in data-intensive science and exascale computing, 2013. DOE Advanced Scientific Computing Advisory Committee (ASCAC) Data Subcommittee Report.

[8] S. Fuller and L. Millett. Future of computing performance: Game over or next level? The National Academies Press, 2011.

[9] G. Golub, F. Luk, and M. Overton. A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Trans. Math. Softw.*, 7:149–169, 1981.

[10] G. Golub and C. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, 4rd edition, 2012.

[11] S. Graham, M. Snir, and C. Patterson. Getting up to speed: The future of supercomputing. The National Academies Press, 2004.

[12] N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53:217–288, 2011.

[13] B. Hendrickson and T. Kolda. Partitioning rectangular and structurally unsymmetric sparse matrix for parallel processing. *SIAM J. Sci. Comput.*, 21:2048–2072, 2006.

[14] M. Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, University of California, Berkeley, 2010.

[15] I. Karasalo. Estimating the covariance matrix by signal subspace averaging. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-34:8–12, Feb. 1986.

[16] N. Knight, E. Carson, and J. Demmel. Exploiting data sparsity in parallel matrix powers computations. In *Parallel Processing and Applied Mathematics*, volume 8384 of *Lecture Notes in Computer Science*, pages 15–25. Springer Berlin Heidelberg, 2014.

[17] T. Kolda and D. O'Leary. A semidiscrete matrix decomposition for latent semantic indexing information retrieval. *ACM Trans. Inf. Syst.*, 16:322–346, 1998.

[18] R. Krovetz and W. B. Croft. Lexical ambiguity and information retrieval. *ACM Trans. Inf. Syst.*, 10:115–141, 1992.

[19] D. Laney. 3D data management: controlling data volume, velocity, and variety. Application Delivery Strategies by META Group Inc, 2001.

[20] P. Martinsson, A. Szlam, and M. Tygert. Normalized power iterations for the computation of SVD. In *Proceedings of the Neural and Information Processing Systems (NIPS) Workshop on Low-Rank Methods for Large-Scale Machine Learning*, 2010.

[21] P. Menozzi, A. Piazza, and L. C.-Sforza. Synthetic maps of human gene frequencies in Europeans. *Science*, 201:786–792, 1978.

[22] P. Paschou, E. Ziv, E. Burchard, S. Choudhry, W. R.-Cintron, M. Mahoney, and P. Drineas. PCA-correlated SNPs for structure identification in worldwide human populations. *PLoS Genetics*, 3:1672–1686, 2007.

[23] N. Patterson, A. Price, and D. Reich. Population structure and eigenanalysis. *PLoS Genet.*, e190, 2006.

[24] A. Price, N. Patterson, R. Plenge, M. Weinblatt, N. Shadick, and D. Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nat. Genet.*, 38:904–909, 2006.

[25] G. Salton and M. McGill. *Introduction to modern information retrieval*. McGraw-Hill, New York, 1983.

[26] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, pages 158–167, 2000.

[27] H. Simon and H. Zha. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. *SIAM J. Sci. Comput.*, 21:2257–2274, 2000.

[28] A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23:2165–2182, 2002.

[29] J. Tougas and R. Spiteri. Updating the partial singular value decomposition in latent semantic indexing. *Comput. Statist. Data Anal.*, 52:174–183, 2007.

[30] E. Vecharynski and Y. Saad. Fast updating algorithms for latent semantic indexing. *SIAM J. Matrix Anal. Appl.*, 35:1105–1131, 2014.

[31] I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, and J. Dongarra. Improving the performance of CA-GMRES on multicores with multiple GPUs. In *Proceedings of the IEEE International Parallel and Distributed Symposium (IPDPS)*, pages 382–391, 2014.

[32] I. Yamazaki, T. Mary, J. Kurzak, and S. Tomov. Access-averse framework for computing low-rank matrix approximations. In *Proceedings of the international workshop on high performance big graph data management, analysis, and minig*, pages 70–77, 2014.

[33] I. Yamazaki, S. Rajamanickam, E. Boman, M. Hoemmen, M. Heroux, and S. Tomov. Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster. In *the proceedings of the international conference for high performance computing, networking, storage and analysis (SC)*, pages 933–944, 2014.

[34] I. Yamazaki, S. Tomov, T. Dong, and J. Dongarra. Mixed-precision orthogonalization scheme and adaptive step size for CA-GMRES on GPUs. Technical Report UT-EECS-14-730, University of Tennessee, Knoxville. To appear in the 11th international meeting on high-performance computing for computational science (VECPAR), 2014.

[35] H. Zha, O. Marques, and H. Simon. Large-scale SVD and subspace-based methods for information retrieval. *Lecture Notes in Computer Science*, 1457:29–42, 1998.

[36] H. Zha and H. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput.*, 21:782–791, 2006.