Access-averse Framework for Computing Low-rank Matrix Approximations

Ichitaro Yamazaki[†,*], Theo Mary[‡], Jakub Kurzak[†], Stanimire Tomov[†], and Jack Dongarra[†]
[†]Department of Computer Science, University of Tennessee, Knoxville, Tennessee, U.S.A.
[‡]Université de Toulouse, INPT(ENSEEIHT)-IRIT, France

***Abstract*—Low-rank matrix approximations play important roles in many statistical, scientific, and engineering applications. To compute such approximations, different algorithms have been developed by researchers from a wide range of areas including theoretical computer science, numerical linear algebra, statistics, applied mathematics, data analysis, machine learning, and physical and biological sciences. In this paper, to combine these efforts, we present an "access-averse" framework which encapsulates some of the existing algorithms for computing a truncated singular value decomposition (SVD). This framework not only allows us to develop software whose performance can be tuned based on domain specific knowledge, but it also allows a user from one discipline to test an algorithm from another, or to combine the techniques from different algorithms. To demonstrate this potential, we implement the framework on multicore CPUs with multiple GPUs and compare the performance of two representative algorithms, blocked variants of matrix power and Lanczos methods. Our performance studies with large-scale graphs from real applications demonstrate that, when combined with communication-avoiding and thick-restarting techniques, the Lanczos method can be competitive with the power method, which is one of the most popular methods currently used for these applications. In addition, though we only focus on the truncated SVDs, the two computational kernels used in our studies, the sparse-matrix dense-matrix multiply and tall-skinny QR factorization, are fundamental building blocks for computing low-rank approximations with other objectives. Hence, our studies may have a greater impact beyond the truncated SVDs.**

I. Introduction

Low-rank matrix approximations play important roles in many statistical, scientific, and engineering applications. Given that the modern applications that generate "big data" with a massive volume, variety, velocity, and veracity, we face an increasing demand for a software package that can efficiently and robustly compute such approximations for a wide range of applications on modern computers. Though many algorithms have been developed for computing low-rank approximations, they are often developed by the researchers of a given discipline and optimized for their individual needs. As our first attempt to combine these efforts, in this paper, we present the existing algorithms to compute low-rank approximations in a single framework. This framework allows a user to test an algorithm developed by different disciplines, combine techniques from different algorithms, or tune its performance based on their domain specific knowledge (e.g., required solution accuracies or singular value distributions). In addition, the framework may provide a foundation for developing a robust and efficient software package.

While there exist low-rank approximations with different objectives (e.g., matrix completion and matrix sampling), in this paper, we focus on truncated singular value decomposition (SVD) [5]. This is primarily because SVD computes the approximation with the minimum spectral or Frobenius norm, and it has been used in many applications, including principal component analysis, community detection, clustering, node ranking, and collaborative filtering in large graphs. Though there are many algorithms to compute a truncated SVD, we focus on random projection methods [6], [9] because in comparison to the classical algorithms like a block Lanczos [4], these methods are claimed to be not only robust with noisy or incomplete data, but also efficient on large data because it requires fewer matrix accesses and can exploit higher data locality and parallelism. On modern computers, data access and communication are becoming increasingly expensive (in terms of both time and energy consumption), and such "access-averse" properties of the random projection methods to extract as much useful information as possible from each data access are becoming attractive.

To compare the performance of different algorithms, we implemented the framework, which encapsulates all the algorithms described in this paper, including the access-averse algorithms, on multicore CPUs with multiple GPUs. Our experimental results with large-scale graphs from real applications demonstrate that a block Lanczos can be efficient when combined with communication-avoiding [8], [22] and thick-restarting [2], [18] techniques; two techniques developed by two different disciplines – computer science and numerical linear algebra. These two techniques allow us to build the projection subspace with the minimum data access and accelerate the solution convergence by retaining the useful information when restarting the iteration, respectively. Hence, compared to random projection, Lanczos could build a projection subspace that is richer in useful information with fewer communication phases, and potentially with about the same amount of data access, especially when the solution convergence requires a few restart cycles and the matrix can be partitioned well.

The rest of the paper is organized as follows: first, in Sections II and III, we review the random projection and block Lanczos methods, respectively, for computing truncated SVDs. Second, in IV and V, we present thick-restarting and communication-avoiding variants of block Lanczos, respec-

*iyamazak@eecs.utk.edu

978-1-4799-5666-1/14/$31.00 ©2014 IEEE

Notation	Description
m-by-n	dimension of sparse coefficient matrix A
b	block size for block Lanczos
t	targeted number of block singular vectors to be computed
s	step size for matrix powers kernel,
c	restart cycle for block Lanczos
k	number of kept Ritz block vectors for thick-restarted block Lanczos
ℓ	oversampling for random projection
$\mathbf{a}_j$ or $A_{j_1:j_2}$	j-th, or j_1-th to j_2-th columns of A
$A_{i_1,i_2,j_1:j_2}$	i_1-th to i_2-th rows of $A_{j_1:j_2}$
$\mathbf{a}^{(j)}$ or $\mathbf{A}^{(j_1:j_2)}$	j-th, or j_1-th to j_2-th block columns of A
$\mathbf{a}^{(i,j)}$	(i,j)-th block of A

Fig. 1. Notations used in this paper.

tively. Then, in Section VI, we introduce our access-averse framework that encapsulates these methods and describe its implementation on a hybrid CPU/GPU architecture. Next, in Section VII, we discuss our experimental setups to compare the performance of different algorithms in a reasonably fair condition. Finally, in Section VIII, we present our experimental results, using large-scale graphs from real applications. We list our final remarks in Section IX.

Throughout this paper, we aim to compute a tb-rank approximation of an m-by-n sparse matrix A, where t is our target block-wise rank, and b is the block size used in block Lanczos. We denote the j-th column of a matrix A by $\mathbf{a}_j$, while $A_{j:k}$ is the submatrix consisting of the j-th through the k-th columns of A, inclusive. In addition, we use $\mathbf{a}^{(i,j)}$ and $\mathbf{a}^{(j)}$ to represent the (i,j)-th block and the j-th block column of A, respectively, while $\mathbf{A}^{(j:k)}$ is the submatrix consisting of the j-th through the k-th block columns of A. Figure 1 lists the notations that will be used in this paper.

II. Random Projection and Power Methods

A random projection method computes a truncated SVD of an m-by-n input matrix A, based on the following three steps:

1) Generate a pair of $(t+\ell)b$ orthonormal basis vectors P and Q that approximately span the ranges of the matrices A and A^T, respectively,
$$A \approx PQ^T,$$
where b is the block size, t is the number of block singular vectors to be computed, and ℓ is a parameter referred to as oversampling [6].
2) Use a standard deterministic algorithm to compute SVD of the projected matrix B,
$$B = X\Sigma Y^T, \tag{1}$$
where $B = P^TAQ$, X and Y are the left and right singular vectors of B, respectively, and $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{(t+\ell)b})$ with the singular values in the descending order (i.e., $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{(t+\ell)b}$).
3) Compute an approximation to the truncated SVD of A,
$$A \approx U_{1:tb}\Sigma_{1:tb,1:tb}V_{1:tb}^T, \tag{2}$$
where $U = PX$ and $V = QY$.

To generate the orthonormal basis vectors P and Q, one popular approach is to perform power iterations [10]. Namely, starting with an initial n-by-$(t+\ell)b$ Gaussian random vectors $\widehat{Q}^{(1)}$, we perform the following procedure for $j = 1, 2, \ldots,$

1) Compute the orthonormal basis vectors $Q^{(j)}$ by a tall-skinny QR (*TSQR*) factorization of $\widehat{Q}^{(j)}$, i.e., $Q^{(j)}R^{(j-1,j)} := \widehat{Q}^{(j)}$ with an upper-triangular $R^{(j-1,j)}$.
2) Compute $\widehat{P}^{(j)}$ by the sparse-matrix dense-matrix multiply (*SpMM*), $\widehat{P}^{(j)} = AQ^{(j)}$.
3) Compute the orthonormal vectors $P^{(j)}$ by *TSQR* of $\widehat{P}^{(j)}$, i.e., $P^{(j)}R^{(j,j)} := \widehat{P}^{(j)}$ and an upper-triangular $R^{(j,j)}$.
4) Compute $\widehat{Q}^{(j+1)}$ by *SpMM*, $\widehat{Q}^{(j+1)} = A^TP^{(j)}$.

Hence, after the j cycles of the power method, we have the orthonormal projection subspaces that satisfy
$$\begin{cases} \mathrm{span}(Q) = \mathrm{span}((A^TA)^{j-1}\widehat{Q}^{(1)}), \text{ and} \\ \mathrm{span}(P) = \mathrm{span}(A(A^TA)^{j-1}\widehat{Q}^{(1)}), \end{cases}$$
where $Q = Q^{(j)}$ and $P = P^{(j)}$. Since $P^{(j)}R^{(j,j)} = AQ^{(j)}$, the projected matrix B is given by the upper-triangular matrix $R^{(j,j)}$ (i.e., $R^{(j,j)} = P^{(j)T}AQ^{(j)}$). In Section VII, we discuss our implementation of *TSQR*.

III. Block Lanczos Method

In contrast to the subspaces (3) generated by the power method, the c steps of a block Lanczos method [4] generate the Krylov projection subspaces,
$$\begin{cases} \mathrm{span}(\mathbf{q}^{(1)}, \mathbf{q}^{(2)}, \ldots, \mathbf{q}^{(c)}) \\ \quad = \mathrm{span}(\widehat{\mathbf{q}}^{(1)}, (A^TA)\widehat{\mathbf{q}}^{(1)}, \ldots, (A^TA)^{c-1}\widehat{\mathbf{q}}^{(1)}), \text{ and} \\ \mathrm{span}(\mathbf{p}^{(1)}, \mathbf{p}^{(2)}, \ldots, \mathbf{p}^{(c)}) \\ \quad = \mathrm{span}(A\widehat{\mathbf{q}}^{(1)}, A(A^TA)\widehat{\mathbf{q}}^{(1)}, \ldots, A(A^TA)^{c-1}\widehat{\mathbf{q}}^{(1)}). \end{cases}$$
Here, $\widehat{\mathbf{q}}^{(1)}$ is an initial n-by-b random block vector, and the columns of the projection subspaces, $\mathbf{Q}^{(1:c)}$ and $\mathbf{P}^{(1:c)}$, are orthonormal. These basis vectors satisfy the relation,
$$\begin{cases} A^T\mathbf{P}^{(1:c)} = \mathbf{Q}^{(1:c)}B^T + \mathbf{q}^{(c+1)}\mathbf{r}^{(c,c+1)}\mathbf{e}^{(c)T}, \text{and} \\ A\mathbf{Q}^{(1:c)} = \mathbf{P}^{(1:c)}B, \end{cases} \tag{3}$$
where B is a cb-by-cb upper-triangular band matrix,
$$B = \begin{pmatrix} \mathbf{r}^{(1,1)} & \mathbf{r}^{(1,2)T} & & & & \\ & \mathbf{r}^{(2,2)} & \mathbf{r}^{(2,3)T} & & & \\ & & \ddots & \ddots & & \\ & & & \mathbf{r}^{(c-1,c-1)} & \mathbf{r}^{(c-1,c)T} & \\ & & & & \mathbf{r}^{(c,c)} \end{pmatrix},$$
with b-by-b upper-triangular blocks $\mathbf{r}^{(i,j)}$, and $\mathbf{e}^{(c)}$ is the last b columns of the cb-by-cb identity matrix. Hence, given the orthonormal block vectors $\mathbf{q}^{(1)}$, the first step of Lanczos generates $\mathbf{p}^{(1)}$ by *SpMM* to multiply $\mathbf{q}^{(1)}$ with A, followed by *TSQR* to orthonormalize the resulting vectors, i.e., $\mathbf{p}^{(1)}\mathbf{r}^{(1,1)} = A\mathbf{q}^{(1)}$. Then, for the remaining j-th step (i.e., $j = 2, 3, \ldots, c$), Lanczos computes the pair of the basis vectors $\mathbf{q}^{(j)}$ and $\mathbf{p}^{(j)}$ based on the following two-term recurrences,
$$\begin{cases} \mathbf{q}^{(j)}\mathbf{r}^{(j-1,j)} &= A^T\mathbf{p}^{(j-1)} - \mathbf{q}^{(j-1)}\mathbf{r}^{(j-1,j-1)T} \\ \mathbf{p}^{(j)}\mathbf{r}^{(j,j)} &= A\mathbf{q}^{(j)} - \mathbf{p}^{(j-1)}\mathbf{r}^{(j-1,j)T}. \end{cases} \tag{4}$$

To ensure the orthogonality of $\mathbf{q}^{(j)}$, we may perform full reorthogonalization. This is done by first block-orthogonalizing $\mathbf{q}^{(j)}$ against all the previously-orthonormalized basis vectors $\mathbf{Q}^{(1:j-1)}$ (*BOrth*), followed by *TSQR* of $\mathbf{q}^{(j)}$. The basis vectors $\mathbf{p}^{(j)}$ may be reorthogonalized in the same way.

Like the random projection method, an approximation to the truncated SVD is computed based on (1) and (2). To measure the approximation error, we use the i-th residual norm,

$$\|\mathbf{r}_i\|_2 = (\|A\widehat{\mathbf{v}}_i - \widehat{\sigma}_i\widehat{\mathbf{u}}_i\|_2^2 + \|A^T\widehat{\mathbf{u}}_i - \widehat{\sigma}_i\widehat{\mathbf{v}}_i\|_2^2)^{1/2}. \quad (5)$$

Because of the equations (2) and (3), we have

$$\begin{aligned}\|A\widehat{\mathbf{v}}_i - \widehat{\sigma}_i\widehat{\mathbf{u}}_i\|_2 &= \|A\mathbf{Q}^{(1:c)}\mathbf{y}_i - \widehat{\sigma}_i\mathbf{P}^{(1:c)}\mathbf{x}_i\|_2 \\ &= \|B\mathbf{y}_i - \widehat{\sigma}_i\mathbf{x}_i\|_2 = 0,\end{aligned}$$

and

$$\begin{aligned}\|A^T\widehat{\mathbf{u}}_i - \widehat{\sigma}_i\widehat{\mathbf{v}}_i\|_2 &= \|A^T\mathbf{P}^{(1:c)}\mathbf{x}_i - \widehat{\sigma}_i\mathbf{Q}^{(1:c)}\mathbf{y}_i\|_2 \\ &= \|\mathbf{Q}^{(1:c)}(B^T\mathbf{x}_i - \widehat{\sigma}_i\mathbf{y}_i) + \mathbf{q}^{(c+1)}\mathbf{r}^{(c,c+1)}\mathbf{e}^{(c)T}\mathbf{x}_i\|_2 \\ &= \|\mathbf{r}^{(c,c+1)}\mathbf{e}^{(c)T}\mathbf{x}_i\|_2. \quad (6)\end{aligned}$$

Hence, the residual norm (5) can be efficiently computed.

The combined cost of *SpMM*, *BOrth*, and *TSQR* typically dominates the total costs, and hence the total solution time, of Lanczos. To accelerate the solution process, our hybrid CPU/GPU implementation generates the basis vectors on the GPUs, while the SVD of the projected matrix B is computed on the CPU. We distribute both the sparse matrices, A and A^T, and the basis vectors, $\mathbf{Q}^{(1:c+1)}$ and $\mathbf{P}^{(1:c)}$, among the GPUs in a 1D block row format (e.g., using a graph or hypergraph partitioning algorithm, see Section VIII). A more detailed description of our implementation can be found in our previous paper [19].

IV. Thick-restarted Block Lanczos

As the Lanczos iteration proceeds, the explicit orthogonalization of the basis vectors becomes expensive in terms of computation and storage. To avoid the high costs of computing a large projection subspace, we restart the iteration after computing a fixed number c of block basis vectors. There are several strategies to restart the Lanczos iteration for computing eigenvalues [14], [18], which can be adapted to the singular value computation. In this paper, we focus on the *thick restarted* Lanczos [2], which keeps multiple approximate singular vectors, referred to as Ritz vectors, at restart, i.e., $\bar{\mathbf{q}}^{(j)} = \mathbf{v}^{(j)}$ and $\bar{\mathbf{p}}^{(j)} = \mathbf{u}^{(j)}$ for $i = 1, 2, \ldots, k$, where we put a bar on top of the next basis vectors $\bar{\mathbf{q}}^{(j)}$ and $\bar{\mathbf{p}}^{(j)}$ to distinguish them from those of the previous restart-cycle. There are several heuristics to decide which singular vectors to keep at restart [18], [20], but in this paper, we simply keep the fixed number k of block Ritz vectors associated with the largest Ritz values. In addition, the last basis vector from the previous restart-cycle is kept ($\bar{\mathbf{q}}^{(k+1)} = \mathbf{q}^{(c+1)}$). From (3), it can be shown that the residual vectors belong to the space spanned by $\mathbf{q}^{(c+1)}$ (i.e., $\mathbf{r}_i \in \text{span}(\mathbf{q}^{(c+1)})$ for $i = 1, 2 \ldots, k$), and the kept vectors span a Krylov subspace. In addition, for the next restart-cycle, the two-term recurrences (4) are recovered after the $(k+1)$-th block vector $\bar{\mathbf{p}}^{(k+1)}$ is explicitly orthogonalized against all the kept vectors,

$$\bar{\mathbf{p}}^{(k+1)}\bar{\mathbf{r}}^{(k+1,k+1)} = A\bar{\mathbf{q}}^{(k)} - \sum_{i=1}^{k}\bar{\mathbf{p}}^{(i)}\bar{\mathbf{r}}^{(i,k+1)}, \quad (7)$$

where $\bar{\mathbf{r}}^{(i,k+1)} = \mathbf{p}^{(i)T}A\mathbf{q}^{(k)}$ for $i = 1, 2, \ldots, k$. Since $\bar{\mathbf{r}}^{(i,k+1)}$ is equal to $(\mathbf{r}^{(c,c+1)}\mathbf{e}^{(c)T}\mathbf{x}_i)^T$ of (6), it is available as a by-product of computing the residual norm at the previous restart. In the end, after thick-restart, the projected matrix B has the block structure,

$$B = \begin{pmatrix} \sigma^{(1)} & & & \bar{\mathbf{r}}^{(1,k+1)} & & & \\ & \ddots & & \vdots & & & \\ & & \sigma^{(k)} & \bar{\mathbf{r}}^{(k,k+1)} & & & \\ & & & \bar{\mathbf{r}}^{(k+1,k+1)} & \bar{\mathbf{r}}^{(k+1,k+2)T} & & \\ & & & & \ddots & \ddots & \\ & & & & & \bar{\mathbf{r}}^{(c-1,c-1)} & \bar{\mathbf{r}}^{(c-1,c)T} \\ & & & & & & \bar{\mathbf{r}}^{(c,c)} \end{pmatrix},$$

where $\sigma^{(i)} = \text{diag}(\sigma_{(i-1)b+1}, \sigma_{(i-1)b+2}, \ldots, \sigma_{ib})$.

V. Communication-Avoiding Block Lanczos

SpMM, *BOrth*, and *TSQR* all require communication. This includes point-to-point neighborhood communication for *SpMM*, and global reductions in *BOrth* and *TSQR*, as well as data movement through the local memory hierarchy (for reading the sparse matrix and for reading and writing vectors, when they do not fit in cache). On modern computers, such communication is becoming expensive compared to arithmetic operations, in terms of both cycle time and energy.

To improve the performance of block Lanczos for computing singular values, we adapt a communication-avoiding variant of Lanczos for eigenvalue computation [8], [22]. This is done by replacing *SpMM* with a new computational kernel, called matrix powers kernel (*MPK*), that generates a set of s block basis vectors at once. *MPK* applies the matrix power s times to a starting block vector $\mathbf{p}^{(i)}$, and computes $A^T\mathbf{p}^{(i)}$, $(AA^T)\mathbf{p}^{(i)}$, $A^T(AA^T)\mathbf{p}^{(i)}, \ldots,$ $(AA^T)^{s-1}\mathbf{p}^{(i)}$, $A^T(AA^T)^{s-1}\mathbf{p}^{(i)}$. This generates two sets of block vectors:

$$\begin{aligned}\widehat{\mathbf{Q}}^{(i+1:i+s)} &= [A^T\mathbf{p}^{(i)}, A^T(AA^T)\mathbf{p}^{(i)}, \ldots, A^T(AA^T)^{s-1}\mathbf{p}^{(i)}], \\ \widehat{\mathbf{P}}^{(i+1:i+s)} &= [(AA^T)\mathbf{p}^{(i)}, (AA^T)^2\mathbf{p}^{(i)}, \ldots, (AA^T)^{s-1}\mathbf{p}^{(i)}].\end{aligned}$$

To compute such matrix powers on multiple GPUs, each GPU first exchanges with its neighboring GPUs all the required vector elements of $\mathbf{p}^{(i)}$ for computing the local parts of $\widehat{\mathbf{Q}}^{(i+1:i+s)}$ and $\widehat{\mathbf{P}}^{(i+1:i+s)}$. Then, each GPU independently computes the matrix powers by invoking *SpMM* with its local part of the matrix A without further communication [12].

Due to the two-term recurrences (4), the new sets of vectors, $\widehat{\mathbf{Q}}^{(i+1:i+s)}$ and $\widehat{\mathbf{P}}^{(i+1:i+s)}$, can be orthogonalized against the previous vectors by only orthogonalizing $\widehat{\mathbf{q}}^{(i+j)}$ against the j previous block vectors, $\mathbf{Q}^{(i-j:i)}$, and $\widehat{\mathbf{p}}^{(i+j)}$ against the $(j+1)$ previous block vectors $\mathbf{P}^{(i-j-1:i)}$, for $j = 1, 2, \ldots, s$ [22].

Then, these resulting vectors are orthonormalized against each other using *TSQR*,

$$\begin{aligned}\mathbf{Q}^{(i:i+s)}\mathbf{R}_{\mathbf{q}}^{(i:i+s,i:i+s)} &:= \widehat{\mathbf{Q}}^{(i:i+s)},\\ \mathbf{P}^{(i:i+s)}\mathbf{R}_{\mathbf{p}}^{(i:i+s,i:i+s)} &:= \widehat{\mathbf{P}}^{(i:i+s)}.\end{aligned}$$

To maintain the orthogonality, we may perform the full reorthogonalization just as in the block Lanczos (see Section III).

In the end, this communication-avoiding Lanczos, referred to as CA-Lanczos, generates the same basis vectors $\mathbf{Q}^{(1:c)}$ and $\mathbf{P}^{(1:c)}$ as Lanczos, mathematically, but reduces the communication latency by a factor of s (see our implementation in Section VI). In addition, when the matrices A and A^T can be partitioned well, CA-Lanczos communicates about the same amount of data as Lanczos [12].

Once the basis vectors are generated, the band matrix B can be cheaply computed, i.e., for $j = i, i+1, \ldots, i+s$,

$$\begin{aligned}\mathbf{b}^{(j,j+1)} &= (\mathbf{r}_{\mathbf{p}}^{(j,j)})^{-T}\mathbf{r}_{\mathbf{q}}^{(j+1,j+1)T},\\ \mathbf{b}^{(j+1,j+1)} &= \mathbf{r}_{\mathbf{p}}^{(j+1,j+1)}(\mathbf{r}_{\mathbf{q}}^{(j+1,j+1)})^{-1},\end{aligned} \quad (8)$$

where $\mathbf{r}_{\mathbf{q}}^{(j,j)}$ and $\mathbf{r}_{\mathbf{p}}^{(j,j)}$ are the $(j-i+1)$-th diagonal blocks of the upper-triangular matrices, $\mathbf{R}_{\mathbf{q}}^{(1:s+1,1:s+1)}$ and $\mathbf{R}_{\mathbf{p}}^{(1:s+1,1:s+1)}$, respectively.

VI. Framework and Implementation

Figure 2 shows the pseudocode of the framework that aims to encapsulate all the algorithms described in this paper. This framework can compute the low-rank approximations based on a random projection or power method (i.e., $c = 0$), or CA-Lanczos (i.e., $c > 0$). It can also run the standard block Lanczos by orthogonalizing the basis vectors after each *SpMM* during *MPK*, and disabling the orthogonalization after *MPK*. We also implemented an option to run the standard Lanczos iterations for the first few restart cycles before switching to CA-Lanczos. This option not only allows us to build the initial projection subspaces in a stable manner, but it also allows us to gather valuable performance and numerical statistics, which may be used to tune the input parameters of CA-Lanczos for the remaining cycles (e.g., the *MPK* step size s) [21]. Our implementation also includes a few restarting schemes like explicit-restart [4] and thick-restart [18].

For *BOrth* and *TSQR* in our experiments, we used the classical Gram Schmidt (CGS) process [1] and the Cholesky QR (CholQR) factorization [16], respectively, both of which were implemented using the optimized dense GPU kernels that we previously developed in [19]. Though the Householder QR factorization [5] provides an unconditionally stable orthogonalization scheme, CholQR and CGS can exploit the data locality better and obtain higher performance. In addition, for our experiments in this paper, we assumed that the $((t+\ell)b+1)$-st singular value of A is greater than the machine epsilon (i.e., $\sigma_{tb+1} \gg \epsilon$) such that CholQR is numerically stable orthogonalizing the basis vectors. Detailed numerical and performance studies of different orthogonalization schemes for a CA Krylov method can be found in [19]. In most cases in that study, an efficient and stable performance of CA-Krylov was obtained using CholQR and CGS with reorthogonalization. Our implementation also includes several reorthogonalization options (e.g., full reorthogonalization [3] and one-sided reorthogonalization [13]). For *SpMM*, we used CuSPARSE in the compressed sparse row format, while a

Initialization
  Initialize $\widehat{\mathbf{q}}^{(1)}$ as random vectors
  $\widehat{\mathbf{q}}^{(1)} = \mathit{TSQR}(\widehat{\mathbf{q}}^{(1)})$, and $k = 0$
`do` **(Main Loop)**
  1. **Loop-Initialization**
    1.1. **Sparse-Matrix Vector Multiply**
      $\widehat{\mathbf{p}}^{(k+1)} = A\widehat{\mathbf{q}}^{(k+1)}$
    1.2. **Orthogonalization**
      Full Orthogonalization (if needed)
        $\mathbf{p}^{(k+1)} = \mathit{TSQR}(\widehat{\mathbf{p}}^{(k+1)} - \mathbf{P}^{(1:k)}\mathbf{b}^{(1:k,k+1)})$
      Full Reorthogonalization (if needed)
        $\mathbf{p}^{(k+1)} = \mathit{OrthB}(\mathbf{p}^{(k+1)}, \mathbf{p}^{(1:k)})$, update $\mathbf{b}^{(k+1,k+1)}$
  2. **Krlov Iteration**
    `for` $i = k+1, k+1+s, k+1+2s, \ldots, c$ `do`
      2.1. **Matrix Powers Kernel**
        `for` $j = i, i+1, \ldots, i+s$
          $\widehat{\mathbf{q}}^{(j+1)} = A^T\widehat{\mathbf{p}}^{(j)}$ and $\widehat{\mathbf{p}}^{(j+1)} = A\widehat{\mathbf{q}}^{(j+1)}$
      2.2. **Orthogonalization**
        Two-term Orthogonalization (if needed)
          $\mathbf{Q}^{(i,i+s)}\mathbf{R}_{\mathbf{q}} = \mathit{OrthB}(\widehat{\mathbf{Q}}^{(i,i+s)}, \mathbf{Q}^{(min(1,i-s):i)})$
          $\mathbf{P}^{(i,i+s)}\mathbf{R}_{\mathbf{p}} = \mathit{OrthB}(\widehat{\mathbf{P}}^{(i,i+s)}, \mathbf{P}^{(min(1,i-2s):i)})$
        Full Reorthogonalization (if needed)
          $\mathbf{Q}^{(i:i+s)} = \mathit{OrthB}(\mathbf{Q}^{(i:i+s)}, \mathbf{Q}^{(1:i-1)})$, update $\mathbf{R}_{\mathbf{q}}$
          $\mathbf{P}^{(i:i+s)} = \mathit{OrthB}(\mathbf{P}^{(i:i+s)}, \mathbf{P}^{(1:i-1)})$, update $\mathbf{R}_{\mathbf{p}}$
      2.3. **Projected Matrix Computation**
        `for` $j = 1, 2, \ldots, s-1$
          compute $\mathbf{b}^{(i+j-1,i+j)}$ and $\mathbf{b}^{(i+j,i+j)}$ by (8)
    `end for`
  3. **Convergence Check**
    3.1. **Approximation to Truncated SVD**
      $X\widehat{\Sigma}Y^T = \mathit{SVD}(B)$
      $\mathbf{U}^{(1:t)} = \mathbf{P}^{(1:c)}\mathbf{X}^{(1:t)}$ and $\mathbf{V}^{(1:t)} = \mathbf{Q}^{(1:c)}\mathbf{Y}^{(1:t)}$
    3.2. **Sparse-Matrix Vector Multiply**
      $\widehat{\mathbf{q}}^{(c+1)} = A^T\mathbf{p}^{(c)}$
    3.3. **Two-term Orthogonalization** (if needed)
      $\mathbf{q}^{(c+1)} = \mathit{TSQR}(\widehat{\mathbf{q}}^{(c+1)} - \mathbf{q}^{(c)}\mathbf{b}^{(c,c)T})$
    3.4. **Full Reorthogonalization** (if needed)
      $\mathbf{q}^{(c+1)} = \mathit{OrthB}(\mathbf{q}^{(c+1)}, \mathbf{Q}^{(1:c)})$, update $\mathbf{b}^{(c,c+1)}$
    3.5. **Residual Norm Computation**
      `for` $i = 1, 2, \ldots, t$ `do`
        $\mathbf{b}^{(i,i)} = \widehat{\sigma}^{(i)}$ and $\mathbf{b}^{(i,t+1)} = \mathbf{x}^{(c,i)T}\mathbf{b}^{(c,c+1)}$
        `for` $j = 1, 2, \ldots, b$ (and $\hat{j} = (i-1)b + j$)
          $\|A^T\mathbf{u}_{\hat{j}} - \mathbf{v}_{\hat{j}}\sigma_{\hat{j}}\|_2 = \|(\mathbf{b}^{(i,t+1)T})_j\|_2$
      `end for`
      `if` all converged `then`
        return $\mathbf{U}^{(1:t)}, \widehat{\mathbf{\Sigma}}^{(1:t)}, \mathbf{V}^{(1:t)}$
      `else if` explicit restart `then`
        $\mathbf{q}^{(1)} = \mathbf{u}^{(1)}$, and $k = 0$
      `else if` thick restart `then`
        $\mathbf{Q}^{(1:(t+1))} = [\mathbf{U}^{(1:t)}\mathbf{q}^{(c+1)}]$, $\mathbf{P}^{(1:t)} = [\mathbf{V}^{(1:t)}]$, and $k = t$
      `else if` power restart `then`
        $\mathbf{q}^{(1)} = \mathit{TSQR}(\widehat{\mathbf{q}}^{(c+1)})$, and $k = 0$
      `end if`
`while`

Fig. 2. Access Averse Framework for Computing Low-Rank Approximation. In this figure, $\mathit{OrthB}(\mathbf{p}^{(k+1)}, \mathbf{p}^{(1:k)})$ performs *BOrth* to orthogonalize $\mathbf{p}^{(k+1)}$ against $\mathbf{p}^{(1:k)}$ followed by *TSQR* to orthonormalize $\mathbf{p}^{(1:k)}$.

Name	Source	m	n	$\frac{nnz}{m}$	$\widehat{\sigma_1}$
BerkStan	snap.stanford.edu	685,230	685,230	11.1	6.7×10^2
Netflix	netflixprize.com	2,649,429	17,770	37.9	1.9×10^4

Fig. 3. Test matrices.

threaded version of MKL is used to compute the SVD of B.

For the two-term orthogonalization of CA-Lanczos, the block vectors $\widehat{\mathbf{q}}^{(i+j)}$ and $\widehat{\mathbf{p}}^{(i+j)}$ need to be only orthogonalized against the j block vectors $\mathbf{Q}^{(i-j:i-1)}$ and the $(j+1)$ block vectors $\mathbf{P}^{(i-j-1:i-1)}$, respectively, for $j = 0, 1, \ldots, s$ (see Section V).[1] In contrast, our implementation uses *BOrth* to orthogonalize the new set of vectors, $\widehat{\mathbf{Q}}^{(i:i+s)}$ and $\widehat{\mathbf{P}}^{(i:i+s)}$, against the s and $(s+1)$ block vectors, $\mathbf{Q}^{(i-s:i-1)}$ and $\mathbf{P}^{(i-s-1:i-1)}$, respectively. Though this increases the computational cost, we can exploit data locality by orthgonalizing $\widehat{\mathbf{Q}}^{(i:i+s+1)}$ or $\widehat{\mathbf{P}}^{(i:i+s+1)}$ all at once, instead of orthogonalizing each block vector $\widehat{\mathbf{q}}^{(i+j)}$ or $\widehat{\mathbf{p}}^{(i+j)}$ at a time. Since on a modern computer, the data access is becoming increasingly expensive compared to the computation, orthogonalization time can be dramatically reduced by exploiting the data locality.

VII. Experimental Setups

To check for the convergence, we use the equality (6) to compute the residual norms at every restart. This introduces extra computation for the power method, but computing other error measurements (e.g., approximation error $\|A - \widehat{\mathbf{U}}^{(1:t)}\widehat{\mathbf{\Sigma}}^{(1:t,1:t)}\widehat{\mathbf{V}}^{(1:t)}\|$) may require about the same or higher computational cost. In particular, to use (6), we followed the algorithm in [10] and used the orthonormal starting vectors $\mathbf{q}^{(1)}$. Though the random projection could start with non-orthogonal vectors, the initial orthogonalization cost may not be significant, or may be needed to maintain the numerical stability in practice. We study the overhead and accuracy of computing the residual norms in Section VIII.

In our experiments, we fixed the oversampling to be equal to the number of target block singular vectors (i.e., $\ell = t$). Then, to compare the performance of different algorithms, we also fixed the projection subspace dimension (i.e., $t + \ell = c$). With this setup, the power method has about the same storage and computational costs as Lanczos or CA-Lanczos for the first restart cycle (i.e., they generate the same number of orthonormal basis vectors). Then, to thick-restart the Lanczos iteration, we kept two additional block Ritz vectors in addition to the current approximate singular vectors (i.e., $k = t + 2$). Hence, after the first restart-cycle, each Lanczos' cycle only generates the remaining $t - 2$ block basis vectors, $\mathbf{P}^{(t+3:c)}$ and $\mathbf{Q}^{(t+4:c+1)}$, while the power method generates the $2t$ block vectors, $\mathbf{P}^{(1:c)}$ and $\mathbf{Q}^{(c+1:2c)}$ (i.e., $c = 2t$). As a result, compared to Lanczos, the power method performs about $2.0\times$ and $1.25\times$ more floating point operations (flops) for *SpMM* and *Orth*, respectively.

To generate the projection space, the power method has four communication phases (i.e., two *SpMM*s and two *TSQR*). To generate the projection space of the same dimension, Lanczos requires about $4(c-k)$ communication phases. With $c = 2t$ and $k = t + 2$ used in our experiments, this is about t times more communication phases compared to that of the power method. In contrast, CA-Lanczos computes the same basis vector $\mathbf{P}^{(k+1:c)}$ and $\mathbf{Q}^{(k+2:c+1)}$ as Lanczos, but requires only $3(c-k)/s$ communication phases (e.g., two *SpMM*s are replaced by a single *MPK*). In addition, in our experiments, we set the *MPK* step size such that after the first restart cycle, there is only one *MPK* call per restart cycle (i.e., $s = c-k$). Hence, for the second restart loop on, CA-Lanczos has only three communication phases, which is less than the four communication phases required by the power method.

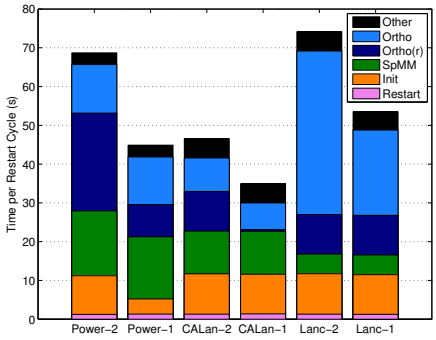


Fig. 4. Time breakdown for BerkStan matrix, with three GPUs.

For the power method and CA-Lanczos, we used a one-sided orthogonalization scheme, where $\mathbf{Q}^{(1:c)}$ is orthogonalized by the two-term orthogonalization, followed by a full reorthogonalization, while $\mathbf{P}^{(1:c)}$ is orthogonalized by a full orthogonalization alone. This orthogonalization scheme was stable with the block size and restart length used for the test matrices in our experiments (i.e., $b = 10$ and $t = 3 \sim 5$). On the other hand, for the one-sided orthogonalization of Lanczos, we used the two-term recurrence, followed by a full-reorthogonalization for $\mathbf{Q}^{(1:c)}$, and just the two-term recurrence for $\mathbf{P}^{(1:c)}$. To maintain the stability of Lanczos, we ran a full reorthogonalization of the last block vector $\mathbf{q}^{(c+1)}$.

VIII. Performance Results

Table 3 shows the two graph matrices from real applications, that we used to study the performance of the power method and Lanczos. For the BerkStan and Netflix matrices, we computed the largest 50 and 30 singular values (in double precision), respectively, using the block size of 10. Unless otherwise specified, both Lanczos and CA-Lanczos are based on thick-restarting. To study the convergence rates, we show the maximum residual norm of the computed singular values at each restart, which was computed using (6). All the experiments were conducted on a single compute node of the Keeneland system at the Georgia Institute of Technology. Each of its compute nodes consists of two six-core Intel Xeon CPUs and three NVIDIA M2090 GPUs, with 24GB of main CPU

[1] Lanczos avoids recomputing the coefficients $\mathbf{r}^{(j-1,j-1)}$ and $\mathbf{r}^{(j-1,j)}$ in (4), which are readily available from the previous iteration due to the symmetry of the coefficient matrix A.

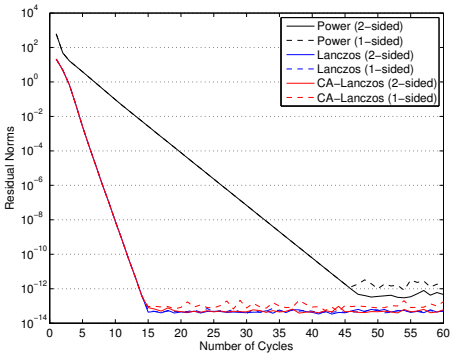
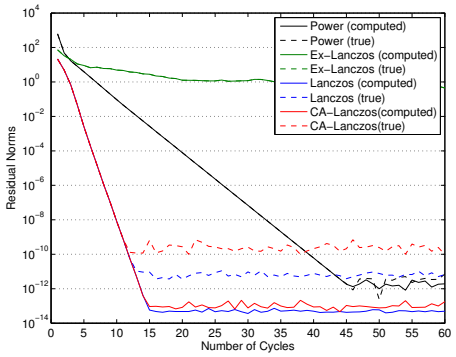


Fig. 5. Convergence history using one-sided or two-sided orthogonalization schemes for `BerkStan` matrix.

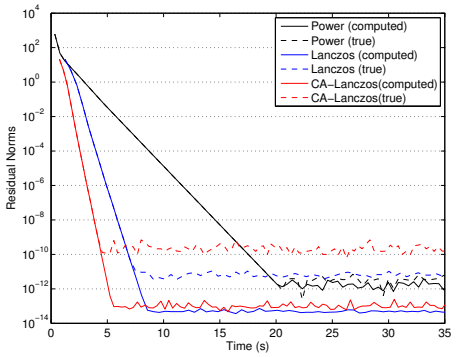
memory per node and 6GB of memory per GPU. We used the GNU `gcc` 4.4.6 compiler and CUDA `nvcc` 5.0 compiler with the optimization flag `-O3`, and linked with Intel's Math Kernel Library (MKL) version 2011_sp1.8.273.

Figure 4 shows the average time spent in the different computational kernels over 100 restart cycles for `BerkStan`, where "Init" is the time required to orthogonalize the first block vector $\mathbf{p}^{(k+1)}$, and "Restart" is the time required to compute the SVD of the projected matrix, the residual norms, and the kept Ritz vectors. The matrix is sparse, and all the methods spent a significant amount of time in orthogonalization. We show the time required to orthogonalize the last basis vectors $q^{(m+1)}$, separately (i.e., "Ortho(r)"). This is a part of the overhead for the power method to compute the residual norm, while Lanczos recycles these basis vectors to restart the iteration. The figure also shows the timing results with both one-sided and two-sided reorthogonalizaiton schemes (e.g., Power-1 and Power-2), where the two-sided scheme performs the two-term orthogonalization followed by the full reorthogonalization. We clearly see that the iteration time was reduced significantly using the one-sided scheme. With the one-sided scheme, the "Init" time of the power method was almost halved because the initial block vector $\mathbf{p}^{(1)}$ is orthogonalized only once. On the other hand, both the one-sided and two-sided schemes of Lanczos orthogonalize the block vector $\mathbf{p}^{(k+1)}$ against the previous vectors $\mathbf{p}^{(1:k)}$ (using the short-cut described with (7) for the first), and spent about the same amount of time in "Init." Figure 5 shows that the one-sided scheme obtained about the same solution convergence as the two-sided scheme.

In Figure 4, we also see that compared to the power method, Lanczos spends less time in *SpMM* per restart cycle. This is because with thick restarting, in total, Lanczos performs *SpMM* with fewer block vectors per restart cycle (i.e., with $t-k$ block vectors). On the other hand, the orthogonalization time of Lanczos was longer because the power method orthogonalizes more block vectors at a time, and the required dense-matrix GPU kernels obtain higher performance, exploiting more data locality and higher parallelism. Finally, CA-Lanczos generates



(a) vs. Restart Cycle.



(b) vs. Time (s).

Fig. 6. Convergence history for `BerkStan` matrix, with three GPUs.

and orthogonalizes the basis vectors all at once, and it obtained the shortest time per restart cycle.

Figure 6 compares the convergence behaviors for `BerkStan`. Figure 6(a) clearly indicates that compared to the power method, Lanczos obtained a faster solution convergence, and CA-Lanczos achieved the same convergence as Lanczos. As a result, though the time per Lanczos iteration was longer than the time per power iteration, Figure 6(b) shows that Lanczos converged faster in terms of wall-clock time. Finally, CA-Lanczos had the shortest time and the fastest convergence per iteration, obtaining the fastest convergence in terms of wall-clock time. The figure also shows that as the iteration proceeds, the computed residual norm could diverge from the true norm. We believe that this gap between the computed and true residual norms may be reduced by integrating more numerical linear algebra techniques (e.g., locking the converged singular values [11], [15]). Finally, in Figure 6(a), we show the convergence behavior of Lanczos with explicit restarting (i.e., Ex-Lanczos with $\widehat{b} = tb$ and $\widehat{c} = 2$, where we put the hat on top of the parameters for explicit-restarting to distinguish them from those used for thick-restarting). In many cases, with explicit restarting, the last few singular values converge slowly, demonstrating the benefits of thick restarting.

Figure 7 shows the breakdown of the average iteration and

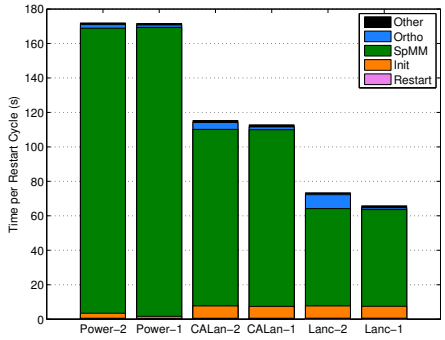
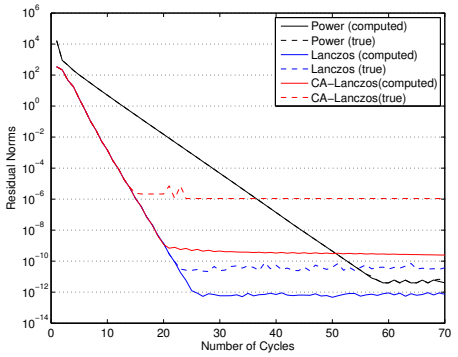
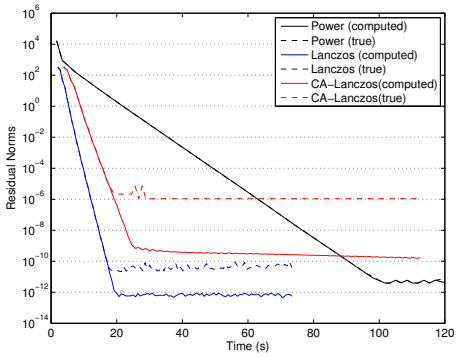


Fig. 7. Time breakdown for `Netflix` matrix, with three GPUs.



(a) vs. Restart Cycle.



(b) vs. Time (s).

Fig. 8. Convergence history for `Netflix` matrix, with three GPUs.

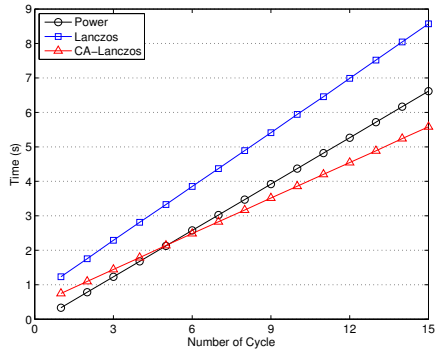
restart time for `Netflix`. Clearly, the iteration time was dominated by *SpMM* for all the methods. This is because compared to `BerkStan`, `Netflix` has more nonzeros, making *SpMM* computationally more expensive. In addition, compared to `BerkStan`, the computational cost of the orthogonalization process, relative to *SpMM*, is lower for `Netflix` because the matrix is rectangular (i.e., $O(mn^2/nnz)$, where $n \ll m$ for `Netfilx`, while $n = m$ for `BerkStan`). Just as with `BerkStan`, due to the thick-restarting, Lanczos spent much less time in *SpMM*, having much shorter iteration time than the power method. Also, for both `BerkStan` and `Netflix`, CA-Lanczos spent more time in *SpMM* than Lanczos. This is because, in order to reduce the communication latency, *MPK* requires $2s$ ghosting of the boundary elements. This could lead to significant computational and storage overheads, and may increase the total communication volume [12], especially for a matrix with an irregular sparsity structure like `Netflix`. Since Lanczos and CA-Lanczos spent significant time in *SpMM* for `Netflix`, the time per CA-Lanzos iteration was longer than the time per Lanczos iteration due to this overhead of *MPK*. Finally, especially for `Netflix`, all the methods spend insignificant time restarting the iteration, demonstrating the small cost of computing the residual norms.

Just like we have observed for `BerkStan`, Figure 8 shows the faster convergence of Lanczos compared to the power method, with respect to both the restart count and wall-clock time. On the other hand, unlike `BerkStan`, for which CA-Lanczos converged faster than Lanczos in terms of wall-clock time, for `Netflix`, CA-Lanczos was slower to converge. This is because CA-Lanczos spent significant time in *SpMM*, and suffered from the overhead of *MPK*.

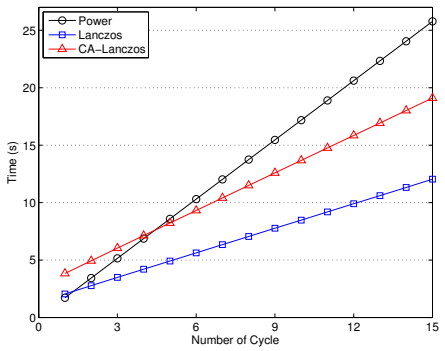
In this paper, we used the residual norms to measure the solution accuracies. In contrast, other applications may prefer other accuracy measures. To study the performance of the power method and Lanczos, independently from the accuracy measures, Figure 9 plots the time required by these methods with respect to the restart cycle. For instance, Figure 9(a) shows that for `BerkStan`, CA-Lanczos may be a better choice if all the methods perform the same number of cycles, and the solution convergence requires more than five iterations (some methods may converge faster than the others, e.g., in our previous experiments, Lanczos converged faster in terms of the residual norms). The first restart cycle of Lanczos was longer than the rest of the cycles because it generates the full c block vectors, while the remaining cycles only generate $c - k$ basis vectors. Figure 9(b) shows the same statistics for `Netflix`. In this case, Lanczos was competitive even for the first restart cycle. This is because the iteration time was dominated by *SpMM*, and the time (and potentially the local communication) required by the power method to perform a single *SpMM* with the 60 vectors was about the same as the time required by Lanczos to perform six *SpMM*s with 10 vectors.

In this paper, we distributed the matrices and the basis vectors among the GPUs such that each GPU has about the same number of rows. We also used METIS and PaToH[2] to partition A and A^T based on graph and hypergraph algorithms [7], [17]. These partitioning algorithms often improve the load balance and reduce the communication among the GPUs for *SpMM*, while the load imbalance may increase during *Ortho*. For `BerkStan`, the performance of the power method was slightly improved using the partitioning algorithm due to the higher performance of *SpMM*, while the Lanczos performance degraded due to the lower *Orth* performance. For `Netflix`,

[2]www.cs.umn.edu/~metis and www.bmi.osu.edu/umit/software.html



(a) `BerkStan` matrix.



(b) `Netflix` matrix.

Fig. 9. Run time vs. number of restart cycles, with three GPUs.

the hypergraph algorithm led to either about the same or lower performance of all the algorithms, while the graph algorithm was slower computing the partition for `BerkStan` and failed for `Netflix`.

IX. Conclusion

In this paper, we compared the performance of a block Lanczos with that of the power method for computing truncated SVDs of graph matrices from real applications. Our performance studies demonstrated the potential of the block Lanczos to obtain high performance, when combined with communication-avoiding and thick-restarting techniques. In particular, Lanczos became competitive to the power method when the power method required a few iterations to converge. We conducted our performance studies on a hybrid CPU/GPU node architecture because we had the optimized computational kernels from the previous studies. We believe the observed performance trends to be true even with other setups. To verify this, we plan to extend our studies for other test matrices, input parameters, objectives, stopping criteria, and hardware architectures. In particular, we plan to extend our implementation to a hybrid CPU/GPU cluster and compute a larger number of singular vectors. In addition, we are investigating other techniques to improve the numerical stability (e.g., orthogonalization scheme, other basis vectors). Finally, to be practical, we would like to develop an interface between our solver and the database where the data is stored.

Acknowledgments

This material is based upon work supported 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.

References

[1] N. Abdelmalek. Round off error analysis for Gram-Schmidt method and solution of linear least squares problems. *BIT Numerical Mathematics*, 11:345–368, 1971.

[2] J. Baglama and L. Reichel. Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM J. Sci. Comput.*, 27:19–42, 2005.

[3] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *J. Soc. Indust. Appl. Math. Ser. B Numer. Anal.*, 2:205–224, 1965.

[4] 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.

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

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

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

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

[9] M. Mahoney. Randomized algorithms for matrices and data. *Found. Trends Mach. Learn.*, 3:123–224, 2011.

[10] P. Martinsson, A. Szlam, and M. Tygert. Normalized power iterations for the computation of svd. Technical report, 2010.

[11] J. McCombs and A. Stathopoulos. Iterative validation of eigensolvers: A scheme for improving the reliability of hermitian eigenvalue solvers. *SIAM J. Sci. Comput.*, 28:23372358, 2006.

[12] M. Mohiyuddin. *Tuning Hardware and Software for Multiprocessors*. PhD thesis, EECS Department, University of California, Berkeley, 2012.

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

[14] D. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Mat. Anal. Appl.*, 13:357–385, 1992.

[15] A. Stathopoulos. Locking issues for finding a large number of eigenvectors of hermitian matrices. Technical Report WM-CS-2005-09, College of William and Mary, 2005.

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

[17] B. Uçar and C. Aykanat. Revisiting hypergraph models for sparse matrix partitioning. *SIAM Rev.*, 49:595–603, 2006.

[18] K. Wu and H. Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM J. Mat. Anal. Appl.*, 22:602–616, 2000.

[19] I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, and J. Dongarra. Improving the performance of CA-GMRES on multicores with multiple GPUs. Technical Report UT-EECS-14-722, University of Tennessee, Knoxville, 2014. To appear in the proceedings of the 2014 IEEE International Parallel and Distributed Symposium (IPDPS).

[20] I. Yamazaki, Z. Bai, H. Simon, L.-W. Wang, and K. Wu. Adaptive projection subspace dimension for the thick-restart Lanczos method. *ACM Trans. Math. Softw.*, 37, 2010.

[21] 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.

[22] I. Yamazaki and K. Wu. A communication-avoiding thick-restart Lanczos method on a distributed-memory system. In *Workshop on Algorithms and Programming Tools for next-generation high-performance scientific and software (HPCC)*, 2011.