Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

STRONG SCALING OF THE SVD ALGORITHM FOR HPC SCIENCE: A PETSC-BASED APPROACH

Paula Ferrero-Roza

Master in Industrial Mathematics
University de La Coruña
Rúa Maestranza 9
15001 La Coruña, SPAIN

José A. Moríñigo

Dept. of Technology
CIEMAT
Avda. Complutense 40
28040 Madrid, SPAIN

Filippo Terragni

Dept. of Mathematics
Universidad Carlos III de Madrid
Avda. de la Universidad 30
28911 Leganés, Madrid, SPAIN

ABSTRACT

The Singular Value Decomposition (SVD) algorithm is ubiquitous in many fields of science and technology. It may be used embedded into other advanced algorithms, solvers or data processing chains. In those scenarios dealing with large data volumes expressed as a huge matrix, there is a need for parallel SVD versions to process it efficiently. We present some ideas and results obtained within the PETSc framework, which enable to design promising HPC scalable solvers. The focused SVD implementations have been taken from the SLEPc library, which is seamless plugged into PETSc to extend its capabilities. Besides its implementation, there is also a randomized-SVD and some wrappers to interface ScaLAPACK and others packages intended to extract singular triplets. This work assesses the strong scaling behaviour attained with these SVD implementations at extracting the leading singular values of a population of both sparse and dense squared matrices. A comparison of performance is provided.

1 INTRODUCTION

The SVD algorithm has a long history with fundamental improvements over the last decades (Dongarra et al. 2018), and has become a keystone in many fields demanding scientific computing: computational chemistry, astronomy, neuroscience, finance, plasma physics or fluid mechanics among others. Typically, preprocessed, recasted data into a matrix entity must be analyzed in terms of their spectral content, which is efficiently done with the SVD. The target matrix may be rather different regarding its properties and computing requirements: sparse or dense; tall-skinny or fat-shaped; and well-conditioned vs. very ill-conditioned. This leads to a vast scenario of cases. Each of the mentioned types is commonly linked to a specific science or technological discipline among those reported above.

One example of demanding computing resources and of interest to our research groups is the embedded SVD solver in the Dynamic Mode Decomposition (DMD) tool (Taira et al. 2020), intended to isolate the persistent coherent structures of complex fluid flows, so making easier the flow understanding. In particular, that SVD version processes dense matrices that comprise ordered spatio-temporal data produced

979-8-3503-6966-3/23/$31.00 ©2023 IEEE

by either time-consuming numerical simulations or by Particle Image Velocimetry (PIV)-based experiments, in which high-rate spatio-frequency sampling is done with optical sensors (Discetti and Coletti 2018). The resulting data-driven entity is a rather large, dense, tall matrix obtained with state-of-the-art tomographic PIV sensors, which may reach several terabytes of storage previous to its SVD processing. Thus, a need for parallel computing is clear because of the intrinsic requirements to deal with such a volume of data. Hence, efficient SVD algorithms should be applied (even because data must fit in the distributed memory of the cluster).

Another promising scenario is to build new preconditioners based on the randomized SVD (Halko et al. 2009; Martinsson 2018) because of its computing economy, then to improve the convergence in iterative solvers applied to large linear systems of equations. Besides accelerating the time-to-solution, other gains may be a better scalability and resilience (Moríñigo et al. 2022). These metrics are key aspects in the design of state-of-the-art solvers at present, as it is the case of communication-avoiding Krylov solvers. Besides, it should be noted that these ideas are much relevant for solvers executed on large computing facilities, more so now that petascale supercomputers are giving way to those of the exascale era.

This investigation explores the performance of the CPU-based SVD implementations available in the Portable, Extensible Toolkit for Scientific Computation (PETSc) framework (Mills et al. 2021) extended with the Scalable Library for Eigenvalue Problem Computations (SLEPc) library (Román et al. 2022), which provides itself the SVD infrastructure. The strong scaling behaviour is analysed with a set of sparse matrices taken from the sparse matrices repository Suite Sparse Matrix Collection (Davis and Hu 2011), and with a set of dense matrices, synthetically generated with specific spectral properties. This population of matrices comprises a wide variety of situations in terms of different mathematical properties.

This article is structured as follows. Next, the computing framework PETSc-SLEPc, which provides the parallel SVD infrastructure, is shortly described. This section includes a schematic introduction to the algorithmic variants of the SVD applied to the matrices. Section 3 presents the population of sparse and dense matrices used in the tests, pointing at some relevant details about their selection and generation (the latter in the case of the dense matrices). Performance metrics for the strong scaling tests, description of the computing cluster at CIEMAT, as well as the main parameters settings are also provided. Section 4 summarizes the results and discussion arising around the set of strong scaling performance plots. Section 5 places this contribution in context within the related work. Finally, some conclusions are given.

2 SVD SOLVERS

2.1 PETSc–SLEPc Framework

PETSc consists of a suite of libraries written in C that are the building blocks for the implementation of large-scale application codes in serial and parallel, which permits easy customization and the extension of both algorithms and implementations. It has parallel linear and nonlinear solvers, and time integrators. As well, it provides the functionality needed within parallel application codes, by means of advanced matrix and vector assembly routines that store, distribute and operate specifically on sparse and dense entities to speedup the computations. This is implemented within its framework in an efficient way by exploiting the C-structures to contain both data and function pointers for operations on the data (similar to C++ classes). PETSc uses the MPI standard for all message-passing communications for distributed computing and recent versions start to include GPU-based functions (several algorithms have been already ported). Also, it is in an underway effort to take advantage of modern architectures (Mills et al. 2021). Since PETSc is designed to support mostly iterative solvers, its native SVD algorithm is constrained to serial (for direct solving of moderate linear systems of equations), with no parallel counterpart included. Optimally, PETSc interfaces a variety of numerical algebra libraries, among them it is SLEPc, which provides several SVD solvers.

SLEPc is a library for the solution of large sparse eigenproblems and other problems such as the computation of partial singular value decomposition of rectangular sparse and dense matrices on parallel computers. It also provides solvers for the computation of the action of a matrix function on a vector by a

Krylov method. The implemented formulations and techniques put the emphasis on problems in which the associated matrices are sparse, as those resulting from the discretization of partial differential equations. But in addition, it provides wrappers and interfaces with other external software packages to carry out parallel SVDs on large dense matrices to extract both singular values and vectors. This is the case of ScaLAPACK (Blackford et al. 1997) and PRIMME (Wu et al. 2017) libraries. SLEPc is built on top of PETSc and works as a seamless extension since it enforces the same programming paradigm following the PETSc's object-oriented style, then supporting the same level of abstraction and an equivalent low-level design of C-structures (such as Mat and Vec for matrix and vector manipulation, respectively).

2.2 SVD Implementations

Next, the set of SVD solvers used in the strong scaling tests are succinctly introduced. Due to extension limits, mathematical details are reduced to a minimum. Exhaustive information can be found in the user's manual of the respective libraries and the formulation is described in many texts, e.g. in (Trefethen and Bau 1997). A separated and somehow more detailed subsection corresponding to the SVD version based on the randomization techniques is given because of its prominent potential in recent applications.

The SVD of an $m \times n$ matrix A corresponds to $A = U\Sigma V^*$, where $U = [u_1, u_2, ...u_m]$ is an $m \times m$ unitary matrix ($U^*U = I$), whose columns are the left singular vectors; $V = [v_1, v_2, ...v_n]$ is an $n \times n$ unitary matrix ($V^*V = I$), whose columns are the right singular vectors; and Σ is an $m \times n$ diagonal matrix with real entries called singular values of A, that is $\Sigma_{ii} = \sigma_i$ for $i = 1, 2, ...min(m, n)$ (illustrated in Figure 1a). If A is real, then U and V result to be real and orthogonal ($U^* = U^T, V^* = V^T$). In the general case of a non-square matrix A with $m \gg n$, the resulting factorization (sometimes called *thin* SVD) exploits the slenderness of the matrix, then U collapses into an $m \times n$ matrix (depicted in Figure 1b), so the factorization reads $A = U_n\Sigma_n V_n^*$. Here, the n singular triplets correspond to (σ_i, u_i, v_i) for $i = 1, 2, ...n$.

2.2.1 Formulations for Sparse Matrices

The computation of the SVD of a matrix can be done with an equivalent eigenvalue problem setup in SLEPc, recasting the matrix A conveniently to extract its singular triplets:

1. The *cross* product matrix method builds the matrix A^*A or AA^*.
2. The *cyclic* matrix method builds the matrix $H(A) = \begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix}$.

The default solver in SLEPc is the one using the cross product matrix (*cross*) as mentioned above. It is noticed that the resulting matrix may be quite different when A is a rectangular matrix $m \times n$ with $m \gg n$. SLEPs selects the minimum size matrix from both options. This has several implications regarding the number of singular values that can be computed because of the size of Σ. Besides, there is a loss of accuracy in computing the smallest singular values by applying the *cross* method. On the other hand, the eigendecomposition of $H(A)$ of the *cyclic* matrix method outputs singular values that are not squared, then the smallest computed values will be more accurate. The expense of this approach is an extra computing cost compared to the *cross* method. Hence, though the *cross* product matrix method tends to be a fastest and most memory-efficient approach for the standard SVD, it is only appropriate when the leading singular values are searched for.

Besides the *cross* and *cyclic* SVD solvers, other specific, robust formulations available in SLEPc are two Lanczos-type solvers: the so-called *Lanczos* and the *thick-restart Lanczos*, both for very large, sparse matrices in parallel computers. Basically, the Lanczos method (*lanczos*) is a two-stage algorithm. Taking the decomposition $A = PBQ^*$ (being P and Q unitary matrices, and B an $m \times n$ upper bidiagonal matrix), the built tridiagonal matrix B^*B is unitarily similar to A^*A (notice that it is unnecessary to build B^*B explicitly to extract the singular values as there are numerical methods to do this directly from B). Hence,

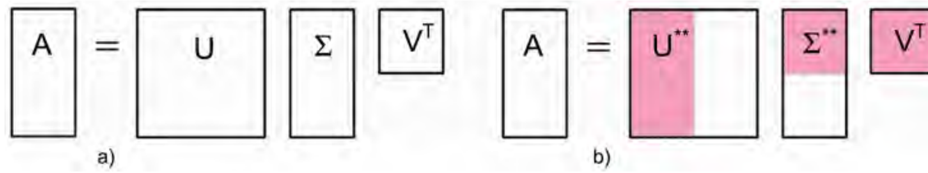

Figure 1: SVD factorization of a rectangular matrix A: standard SVD (left); and *thin* SVD version (right), typically done for $m \gg n$.

expressing the SVD of B as $X\Sigma Y^*$ immediately leads to state the factors $U = PX$ and $V = QY$. The first stage of this algorithm is iterative and corresponds to the bidiagonalization of A that, in general, can be performed using Householder transformations or via Lanczos recurrences (the latter is better intended for sparse matrices). The iteration sequence typically requires a number of Lanczos steps, say k-steps, to build the bidiagonalization. And this implies a computing cost that can be rather high if many singular triplets are requested. In addition, the convergence is highly dependent on the matrix properties, that is on the distribution of singular values and presence of clustered singular values.

The thick-restart Lanczos (*trlanczos*) method is a variant of the above mentioned formulation. It exploits the restarting mechanism, which is a fundamental idea of the projection-based eigensolvers (i.e., Arnoldi or Jacobi-Davidson ones), and conducts one-sided or two-sided reorthogonalization via iterated Classical Gram-Schmidt orthogonalization as part of the process of bidiagonalization, since it demonstrates an improved scalability. In particular, the thick-restart concept is an alternative way and easier to implement than the pure explicit or implicit restarting mechanism used in the Lanczos-based SVD solver. In *trlanczos*, a lower number of k-steps is carried out over the convergence sequence. Under this approach, a set of $l \leq k$ singular triplets can be extracted from the matrix A. The description of the parallel implementation of both Lanczos variants can be found in Hernández et al. (2008) and Alvarruíz et al. (2022).

2.2.2 Formulations for Dense Matrices

Two SVD solvers are considered: function `pdgesvd` of ScaLAPACK; and function `dprimme_svds` of PRIMME, applied to real matrices, both accessed through the SLEPc interface. ScaLAPACK is a mature, widely used numerical algebra library in permanent development by the research community and has become a referential software for computing itself and performance comparison with other algorithms. On the contrary, PRIMME is a more recent library with a smaller community of users. It provides preconditioned solvers for large-scale eigenproblems and SVD factorization of sparse and dense matrices. It is based on Davidson-type methods and permits the computation of a small number of singular triplets (σ_i, u_i, v_i).

2.2.3 Randomized-SVD in SLEPc for Sparse and Dense Matrices

The randomized-SVD (in short, *rsvd*) is a solver that computes the singular triplets in an approximate way by using a low-rank representation of A, built with randomized linear algebra techniques (Halko et al. 2009; Martinsson 2018). The *rsvd* implementation of SLEPc has a low-rank approximation that can be controlled by setting the size of the vector basis *ncv* (the condition $ncv \geq 2 \cdot nsv$ must be fulfilled due to accuracy concerns, being *nsv* the requested number of singular values). The version in SLEPc is iterative and convergence is stated either when the prescribed tolerance condition is fulfilled or the maximum number of iterations is reached. Both parameters should be selected with care to balance the computing cost and attained accuracy of the singular triplets because within the iterative block some time consuming operations (i.e., matrix-matrix and matrix-vector multiplications) are performed.

3 BENCHMARKING SETUP AND METHODOLOGY

The library versions correspond to PETSc v3.18.3 and SLEPc v3.18.1. Other packages installed as external modules of PETSc are MPIch v4.0.2 and ScaLAPACK v2.2.1. For SLEPc, PRIMME 3.2 has been included at the configuration stage. The installation of SLEPc is straightforward once PETSc environment has been set since its `configure` inherits the PETSc setup parameters (only the installation of PRIMME as external module should be added to its default configuration command).

Compilation has been done with the GNU compilers v4.8.5 using the `-O2` flag for optimization. Initial tests with aggressive compilation (flag `-O3`) were conducted for comparison purposes, but finally discarded because the obtained speedup figures were rather similar and this benchmarking is intended to be continued with the development of solvers under `-O2` optimization to avoid possible conflicts. All libraries and modules have been compiled with 32-bit (default) arithmetic (the largest sparse matrices taken from the sparse matrix repository (Davis and Hu 2011) can be adequately managed with it, hence used in the tests). But for dense matrices, the 32-bit arithmetic is more restrictive because the largest set of matrix elements to index implies a maximum square matrix size of about $n \approx 46,000$. This constraint has settled the criterion for dense matrices generation in the present approach. The scaling tests have been executed in cluster ACME located at CIEMAT (Rodríguez-Pascual et al. 2019) and managed under Slurm. An homogeneous partition of 10 nodes, each comprising 2 Intel Gold 6138 processors (20 cores/CPU) @2.0 GHz, with 192 GB RDIMM memory has been used. Dedicated access to the queue is enforced with the `exclusive` command of Slurm, to ensure no interference with other users occur during the benchmarking campaign. Hence, the largest MPI-based parallelization carried out in ACME corresponds to 400 MPI ranks (each one allocated in a dedicated core; multithreading is disabled). Jobs submission in batch mode and postprocessing have been automatized as much as possible.

3.1 Sparse Matrices

In total 17 square, real, sparse matrices have been selected from the Suite Sparse Matrix Collection repository, see Table 1. Mostly, they derive from modelling (PDEs discretizations) real world applications and exhibit a variety of size and sparsity, bringing a comprehensive subset of cases.

3.2 Dense Matrices with Prescribed Properties

In total 16 square, dense matrices of Gaussian numbers have been generated synthetically with prescribed condition number and singular values distribution for each A size: n = 6,400, 12,800, 25,600 and 46,080 (all them multiple of 2^p with $p = 1, 2, ...$). For each size, two condition numbers ($\kappa(A) = 10^2$ and 10^4) and two singular values distributions (arithmetic and geometric decreasing laws) have been prescribed. The combination of n, $\kappa(A)$ and decreasing laws provides the mentioned population of matrices. Matrix acronym follows the rule [Law][Size][$\kappa(A)$ exponent], where [Law] = ari, geo; [Size] = 6,400, 12,800,...; and [exponent] = 2, 4 (e.g., $n = 12,800$, geometric law, and $\kappa(A) = 10^4$ is $geo12800k4$).

3.3 Execution Parameters

Input of Matrix files to PETSc has been accomplished in two steps: a human-readable standard MatrixMarket format file is generated; then, it is converted to PETSc binary format to be read by the SVD-solver. Relative tolerance is set to 10^{-7} and the leading ten singular values are requested (no singular vectors) in the setup. These are stored as part of the code output for performance comparison.

Computing time corresponds, in strict sense, to the SVD solver object execution, thus not considering the time invested in other support data structures and SVD objects setups. Each execution has been repeated three times and then the mean calculated. Distribution of MPI processes over the cluster CPUs has been enforced with the `cyclic` flag of Slurm, set in the batch script. In addition, each SVD solver is controlled via specific command line parameters included in the batch file.

Table 1: Set of sparse matrices used in the strong scaling tests. Number of non-zeroes *nnz* and sparsity percentage are indicated . Matrices are classified according to the number of rows *n* (the largest at the end).

Name	n	nnz	Sparsity(%)	Application kind
ex35	19,716	227,872	0.059	Computational Fluid Dynamics (CFD)
Ill_Stokes	20,896	191,368	0.044	CFD Problem
ABACUS_shell_ud	23,412	218,484	0.040	Model Reduction Problem
af23560	23,560	460,598	0.083	CFD Problem
viscoplastic2	32,769	381,326	0.036	Materials Problem
Onetone1	36,057	335,552	0.026	Frequency Domain Circuit Simulation
Ga10As10H30	113,081	6,115,633	0.048	Quantum Chemistry
PR02R	161,070	8,185,136	0.032	CFD Problem
RM07R	381,689	37,464,962	0.026	CFD Problem
pre2	656,033	5,834,044	0.0013	Frequency Domain Circuit Simulation
boneS10	914,898	40,878,708	0.0049	Model Reduction Problem
HV15R	2,017,169	283,073,458	0.0069	CFD Problem
Bump_2911	2,911,419	127,729,899	0.0015	2D/3D Problem
Queen_4147	4,147,110	316,548,962	0.0018	2D/3D Problem
nlpkkt160	8,345,600	225,422,112	0.00032	Optimization Problem
nlpkkt200	16,240,000	440,225,632	0.00017	Optimization Problem
nlpkkt240	27,993,600	760,648,352	0.000097	Optimization Problem

4 RESULTS AND DISCUSSION

Experiments with sparse matrices reveal that the speedup quickly deteriorates for small to moderate size n. This is explained by the low CPU occupation per core in parallelizations, that is the few degrees-of-freedom assigned per MPI rank ($dofs/rank$) in the solving process. One example of this is shown in Figure 2 for matrix *pre2* with sparsity $= 0.0013\%$. Interestingly, it is observed that with about 10 MPI ranks all the speedup curves experience a visible slope bend, which may be attributed to memory contention issues (it is stressed that benchmarking has been addressed with 10 nodes of the cluster ACME and enforced MPI ranks distribution over the cluster. Hence, when the MPI ranks of the parallelization $>$ number of nodes, various processes are allocated in the same CPU and typically request access to identical RAM addresses, causing contention). In addition to the small speedup observed in small size sparse matrices, this behaviour seems to be common and the location of a curve bend is clearly visible in most cases.

Performance plots of three additional sparse matrices are included in Figure 2 corresponding to much larger ones (notice that matrix *nlpkkt240* is the largest matrix analyzed in the present investigation). It is seen that they improve the speedup notably, in agreement with what is desirable when higher CPU load per core occurs in the computation. Parallelization over 64 MPI ranks (even up to 400 MPI ranks for the matrix *nlpkkt240*) seems to be optimal for these sizes. It should be said that the speedup curve computed with every SVD version approaches the ideal one as *nnz* increases (which means that a larger $dofs/rank$ has been accommodated). This effect is also visible in the respective parallel efficiency plots, which exhibit better efficiency values over a wider range of MPI ranks. Furthermore, the performance variation among the tested SVD solvers is smaller (the speedup curves corresponding to *Bump*_2911 and *nlpkkt240* are quite similar). In the entire set of tested SVDs, the *cyclic* version provides the higher speedup and better efficiency over a wider range of MPI ranks. This is achieved for all matrices, independently of the matrix size. The randomized-SVD (*rsvd*) of SLEPc provides a competitive performance with the largest sparse matrices of the population according to the plots, but in general it underperforms the *cyclic* version. Besides, it is noted that the *rsvd* execution time is worse with small matrices and low parallelizations, albeit it gets closer to the *cyclic* computing cost within the higher range of parallelization. This suggests an advantageous usage

zone in the performance plots related to higher $dofs/rank$, which deserves more attention in a further work dealing with larger matrices. Since it is seen that all the speedup curves tend, somehow, to coalesce with the largest matrices, it seems clear that the criterion of having a smaller execution time is fundamental for taking decisions about the suitability of a specific SVD algorithm.

Figure 3 compares the maximum speedup attained with 17 sparse matrices and two SVD solvers: *cyclic* and *rsvd*, to remark the differences. The trend is similar as the maximum speedup increases with *nnz* for most of the largest matrices. And this behaviour occurs at the highest MPI ranks (256 to 400).

Results for dense matrices differ qualitatively from sparse matrices. First, maximum speedup is now significantly smaller (see Figure 5). Only the SVD implementation of ScaLAPACK (dedicated to dense algebra) provides speedup peaks in the order of the best values attained in the sparse population. And interestingly, it occurs in a non-systematic way: some matrices exhibit this behaviour, but it does not depend on the matrix size n, $\kappa(A)$ or singular values distribution law. Necessarily a better understanding of this pattern would require a much larger population of dense matrices to test.

Figure 5 depicts the maximum speedup attained with four SVD versions: *cyclic*, ScaLAPACK, PRIMME and *rsvd*. It shows that they all underperform, to a great extent, the performance of ScaLAPACK. Interestingly, PRIMME version gives a much lower time-to-solution performance, which besides turns to be quite independent of the matrix properties, but its speedup is worse (see Figure 4). Driven by paper extension constraints, only four representative dense matrices of increasing n (from smallest to largest size) taken from the entire population have been included in Figure 4. The sequence indicates that the speedup experiences a more accentuated drop after the maximum speedup location. But now the smallest matrices exhibit an acceptable strong scaling behaviour. This follows from the dense scenario having much more CPU load in contrast to the sparse scenario for equal size n (and $dofs/rank$). Also an acceptable parallel efficiency is linked to the smallest sizes. It should be said that users and code developers are interested in minimizing time-to-solution, so any parallel efficiency is acceptable meanwhile the number of MPI ranks involved provide a significant increase of the speedup, hence an execution drop. But from a cluster's administration standpoint, a too small efficiency implies an inefficient assignment of the available HPC resources, then an implicit penalty to other queued jobs pending of execution, which could potentially use more effectively the cluster. An accepted criterion of how to set a parallel efficiency threshold depends on the organization, but being over a 50% may be a reasonable objective in parallel executions to avoid to waste too much computing time.

5 RELATED WORK

Strong and weak scaling studies on algorithms and solvers are fundamental to understand their bottlenecks, then to clarify how they may impact on the scalability of those HPC applications that implement them. Bibliography on strong and weak scaling benchmarking on supercomputers is extensive, but the one related to the SVD algorithm is rather scarce and driven by the renewed interest over the last years to achieve better scaling implementations. Indeed, it is an active field of research since until today it has been difficult to overcome the SVD memory contention and communication constraints of its parallel computing versions.

The review paper (Dongarra et al. 2018) provides an introduction to the state-of-the-art SVD implementations for dense matrices computations with CPUs and accelerators such as GPUs, followed by a discussion of the results obtained with various historic and current SVD versions. Systematic tests on a multicore machine (that is, in-node performance) and a distributed computing platform have been carried out. Among the compared numerical algebra libraries, they present results for the SVD of ScaLAPACK (Blackford et al. 1997) in a distributed-memory computer. Their comparison stresses the impact of the successive algorithmic changes of the SVD formulations over the last forty years. Hence, clear improvements arise in terms of time-to-solution and speedup, which has increased by several orders of magnitude. Also they quantify the attained relative speedup with a reference SVD in distributed computing for increasing matrix size. But they do not analyze the absolute speedup provided by these SVD versions, neither they discuss techniques for solving SVD problems with sparse linear systems. Interestingly, besides the benchmarking

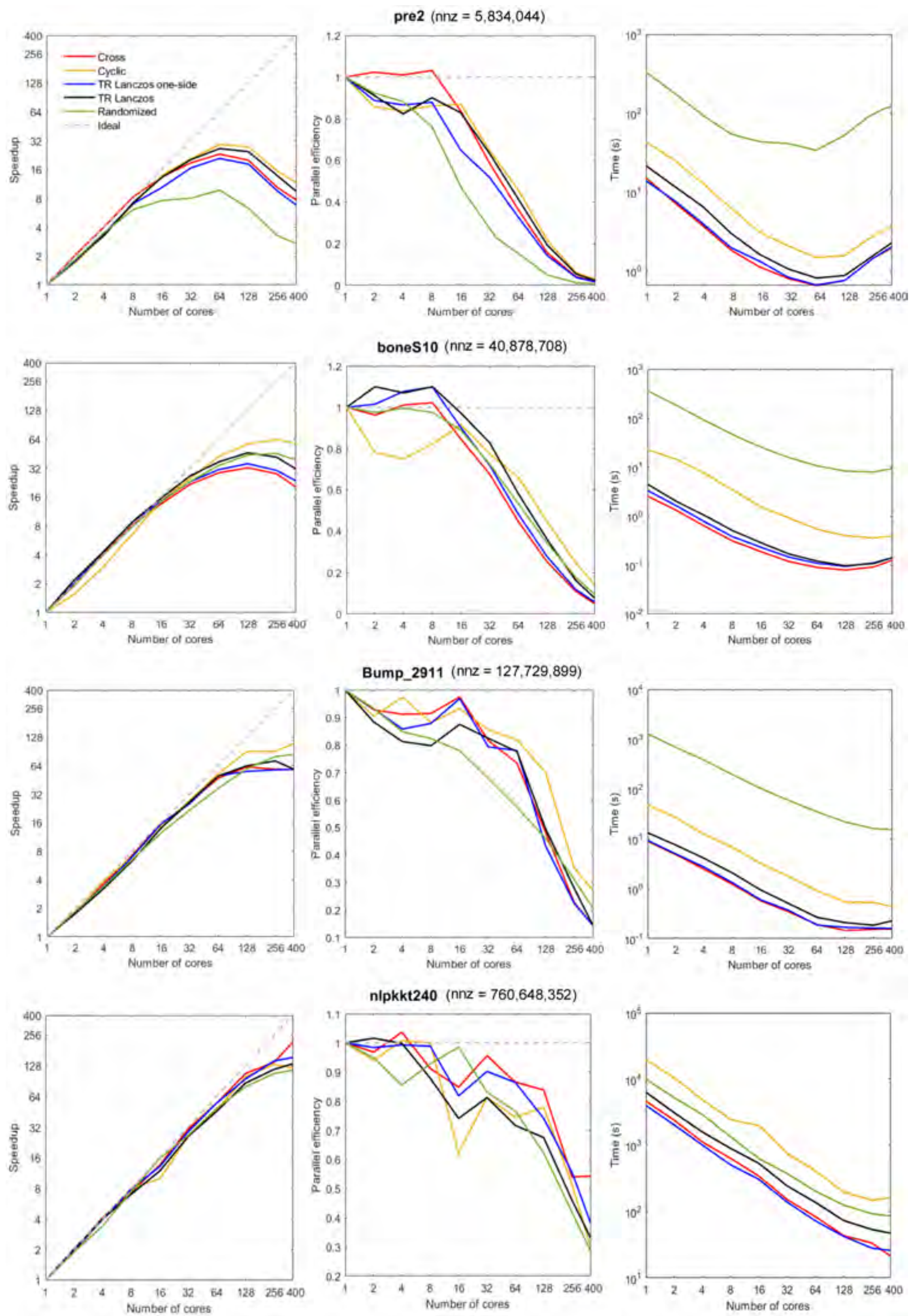

Figure 2: Speedup, parallel efficiency and execution time attained with different SVD algorithms for sparse matrices. Plots correspond to four matrices of increasing *nnz*, indicated by the matrix acronym.

with square matrices of increasing size, they analyze the performance at computing SVDs of tall and very-tall dense matrices, showing that a clear improvement in speedup occurs with the tallest matrices.

The scaling behaviour of the SVD has been analyzed in the Parallel Numerical Linear Algebra for Future Extreme Scale Systems (NLAFET) project (Blanchard et al. 2019), funded by the European Union's

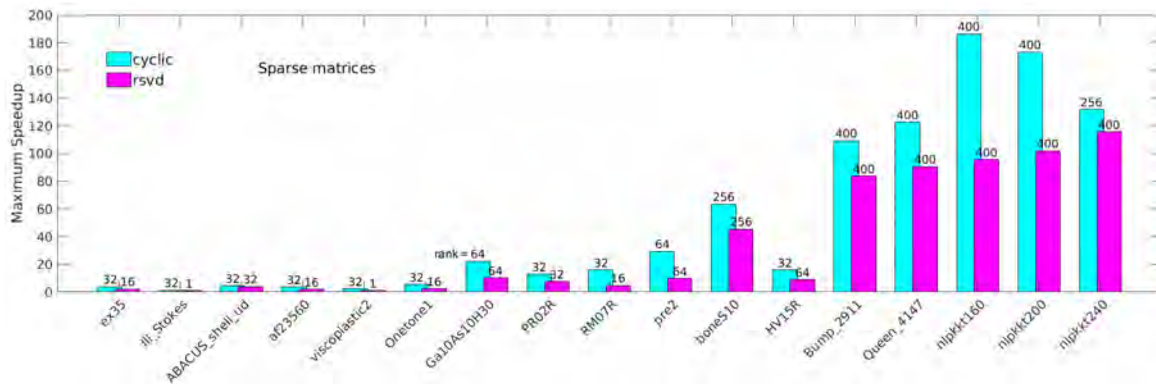

Figure 3: Maximum speedup with sparse matrices for SVD solvers *cyclic* and *rsvd*. Corresponding MPI-rank at max-speedup is on top of the bars (matrix size increases from left to right).

Horizon 2020 research and innovation program, where formulations using the full computing capabilities of novel architectures are investigated. A developed implementation of the polar decomposition is explored to compute the matrix factorization of the SVD. Then, its performance is compared with other well established numerical linear algebra libraries as PLASMA (Dongarra et al. 2019), ScaLAPACK and others on pure CPU distributed-memory computers and hybrid CPU-GPU platforms. The research stresses the scaling gain thanks to the novel one-stage SVD versions compared to the two-stage SVD ones.

In survey (Schmidt 2020) three SVD solvers corresponding to the normal-equations, QR-based and randomized-SVD formulations are analyzed and executed on the Summit supercomputer (USA). These implementations use MPI communications and are tested with a group of tall/skinny matrices. The attained execution speed is compared with its counterpart corresponding to versions coded for GPU, as well as with the standard SVD included in ScaLAPACK. Precisely, it is such mature stage of ScaLAPACK and its SVD implementation the reason of being so widely used across the scaling literature, as referred above.

In Hernández et al. (2008) the authors conduct some strong scaling tests using the PETSc-SLEPc framework on a reduced set of sparse matrices, hence to test the scaling properties of the SVD solver based on the restart Lanczos bidiagonalization. It should be mentioned that the same authors have given various presentations about the strong and weak scaling attained with some SVD versions available in SLEPc, but far from being comprehensive. Another fundamental aspect has to do with the matrix generation and input to the solvers. Benchmarking of SVD libraries requires sparse and dense matrices of increasing size as input to the algorithm. There exist sparse matrices repositories of ample usage, as it is the Suite Sparse Matrix Collection (formerly maintained by the Florida State University) (Davis and Hu 2011). When large dense matrices of prescribed singular values or condition number are needed, an efficient way to generate them is described in Fasi and Higham (2021). Basically, a SVD-based algorithm builds the three factors, then leading to the final matrix of prescribed properties. Their approach is capable to deal with extreme-scale matrices, thus suitable for scaling tests in petascale computers and beyond.

The present investigation provides a systematic benchmarking of the most practical parallel SVD solvers coded in the PETSc-SLEPc framework, which leads to a better knowledge of their suitability standalone or as a building block into a more complex solver that needs a SVD factorization.

6 CONCLUSIONS

The strong scaling behaviour of seven SVD versions has been analyzed with a representative population of sparse and dense matrices (33 matrices in total). Five SVD versions come from the SLEPc own implementations. And two SVDs (of ScaLAPACK and PRIMME) derive from libraries interfaced with PETSc-SLEPc. As expected, strong scaling rapidly deteriorates with the smallest sparse matrices (too

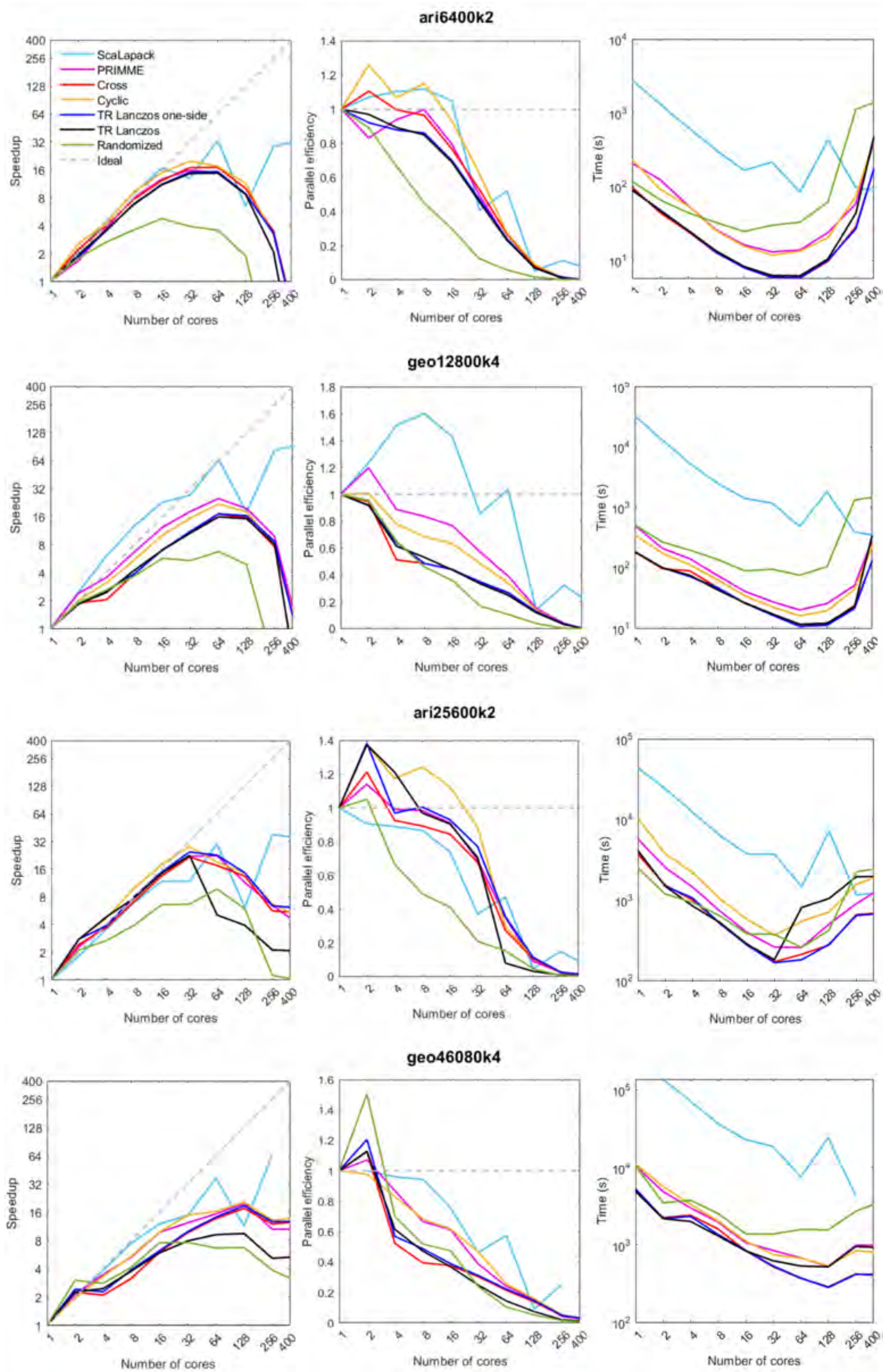

Figure 4: Speedup, parallel efficiency and execution time attained with different SVDs for dense matrices. Rows correspond to increasing *n* (smallest: ari6400k2, upper row; largest: geo46080k4, bottom row).

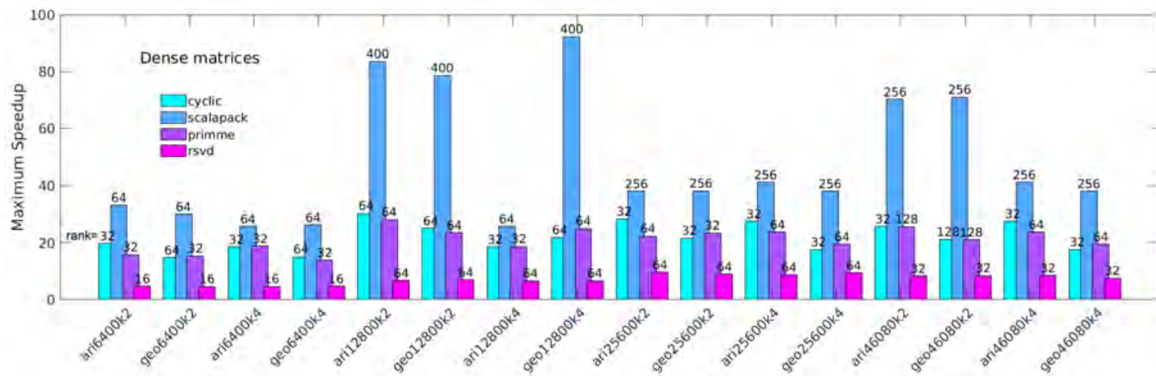

Figure 5: Maximum speedup with dense matrices for SVD solvers *cyclic*, ScaLAPACK, PRIMME, *rsvd*. Corresponding MPI-rank at max-speedup is on top of the bars (matrix size increases from left to right).

low $dofs/rank$), opposite to the largest sparse matrices, which are the most appropriate to carry out large MPI-based executions of SVDs. The *cyclic* version provides the best scaling in the sparse scenario.

On the contrary, the attained SVDs performance with dense matrices results to be more homogeneous across the entire matrix population. The few $dofs/rank$ allocated for all the analyzed dense matrices drive the rather modest attained scalability. Thus, it is concluded that much larger dense matrices must be generated to adequately test the SVD scalability. A major observation in the dense scenario is that ScaLAPACK clearly outperforms the other tested SVDs. This is explained because of its fine optimization for this kind of matrices.

The randomized-SVD of SLEPc has been benchmarked with sparse and dense matrices. At present, no conclusive results can be firmly stated about it, but its performance looks promising at least for extracting the leading singular values of large sparse matrices. However, its application to the dense scenario requires optimization to be competitive with the other approaches. Finally, it is stressed that the largest matrices analyzed in the present investigation are on the edge of what is allowed with the 32-bit arithmetic. Hence, a next study on performance with much larger sparse and dense matrices requires to configure the PETSc-SLEPc framework under 64-bit arithmetic. Several issues remain open: the sensibility of the results to the cluster ACME architecture, thus the interest in repeating the tests with at least another supercomputer; weak scaling results, not included here due to space restrictions; and also how the SVD performance with tall-skinny matrices compares to the square matrices results. These aspects deserve further analysis to go deeper into the SVDs scalability.

ACKNOWLEDGMENTS

CIEMAT contribution was partially funded by the ERDF Spanish Ministry of Science, Innovation, and Universities CODEC-OSE project (RTI2018-096006-B-I00), the co-funded Comunidad de Madrid project CABAHLA-CM (S2018/TCS- 4423) and the EC H20202 project RISC2 (Grant ID: 101016478). F. Terragni (UC3M) was supported by the ERDF Spanish Ministry of Science, Innovation, and Universities under grant PID2020-112796RB-C22. We thank Fernando Varas-Mérida, coordinator of the Master in Industrial Mathematics (M2i) at the Technical University of Madrid (UPM), for enabling the participation of CIEMAT in the M2i. Last, the authors thank the ACME cluster administrator A. J. Rubio-Montero for his support.

REFERENCES

Alvarruíz, F., C. Campos, and J. E. Román. 2022. "Thick-restarted Joint Lanczos Bidiagonalization for the GSVD". *ArXiv*:1–25.

Blackford, L. S., J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley, and J. J. Dongarra. 1997. *ScaLAPACK User's Guide*. USA: Society for Industrial and Applied Mathematics.

Blanchard, P., M. Zounon, J. Dongarra, and N. Higham. 2019. "Novel SVD Algorithm - Deliverable D2.9". Technical Report H2020-FETHPC-2014:GA671633.

Davis, T. A., and Y. Hu. 2011. "The University of Florida Sparse Matrix Collection". *ACM Transactions on Mathematical Software* 38(1):1–25.

Discetti, S., and F. Coletti. 2018, March. "Volumetric Velocimetry for Fluid Flows". *Measurement Science and Technology* 29(4):042001.

Dongarra, J., M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki. 2018. "The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale". *SIAM Review* 60(4):808–865.

Dongarra, J., M. Gates, A. Haidar, J. Kurzak, P. Luszczek, P. Wu, I. Yamazaki, A. Yarkhan, M. Abalenkovs, N. Bagherpour, S. Hammarling, J. Šístek, D. Stevens, M. Zounon, and S. D. Relton. 2019. "PLASMA: Parallel Linear Algebra Software for Multicore Using OpenMP". *ACM Trans. Math. Softw.* 45(2).

Fasi, M., and N. J. Higham. 2021. "Generating Extreme-Scale Matrices With Specified Singular Values or Condition Number". *SIAM Journal on Scientific Computing* 43(1):A663–A684.

Halko, N., P.-G. Martinsson, and J. A. Tropp. 2009. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". *ArXiv*.

Hernández, V., J. E. Román, and A. Tomás. 2008. "A Robust and Efficient Parallel SVD Solver Based on Restarted Lanczos Bidiagonalization". *ETNA. Electronic Transactions on Numerical Analysis [electronic only]* 31:68–85.

Martinsson, P. 2018. *Randomized Methods for Matrix Computations*, 187–230. American Mathematical Society.

Mills, R. T., M. F. Adams, S. Balay, J. Brown, A. Dener, M. Knepley, S. E. Kruger, H. Morgan, T. Munson, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, and J. Zhang. 2021. "Toward Performance-portable PETSc for GPU-based Exascale Systems". *Parallel Computing* 108:102831.

Moríñigo, J. A., A. Bustos, and R. Mayo-García. 2022. "Error Resilience of Three GMRES Implementations under Fault Injection". *J. Supercomputing* 78(5):7158–7185.

Rodríguez-Pascual, M., J. A. Moríñigo, and R. Mayo-García. 2019. "Effect of MPI Tasks Location on Cluster Throughput Using NAS". *Cluster Computing* 22(4):1187–1198.

Román, J. E., C. Campos, L. Dalcin, E. Romero, and A. Tomas. 2022. "SLEPc Users Manual". Technical Report DSIC-II/24/02 - Revision 3.18, D. Sistemes Informàtics i Computació, Universitat Politècnica de València.

Schmidt, D. 2020, nov. "A Survey of Singular Value Decomposition Methods for Distributed Tall/Skinny Data". In *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, 27–34. Los Alamitos, CA, USA: IEEE Computer Society.

Taira, K., M. S. Hemati, S. L. Brunton, Y. Sun, K. Duraisamy, S. Bagheri, S. T. M. Dawson, and C.-A. Yeh. 2020. "Modal Analysis of Fluid Flows: Applications and Outlook". *AIAA Journal* 58(3):998–1022.

Trefethen, L. N., and D. Bau. 1997. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics.

Wu, L., E. Romero, and A. Stathopoulos. 2017. "PRIMME_SVDS: A High-Performance Preconditioned SVD Solver for Accurate Large-Scale Computations". *SIAM Journal on Scientific Computing* 39(5):S248–S271.

AUTHOR BIOGRAPHIES

PAULA FERRERO-ROZA earned her MSc in Industrial Mathematics at the University of La Coruña. Formerly, she graduated in Mathematics at the University of Oviedo (Spain). Her interests are connected to applying Maths to problem solving in industry, and she is mostly focused on the applied branches of Mathematics. Recently she conducted research in high performance computing in the context of her MSc thesis, which was carried out in the Department of Technology of CIEMAT. Her email address is paula.ferrero.roza@udc.es.

JOSÉ A. MORIÑIGO is Senior Staff Researcher at the Department of Technology of the Centre for Energy, Environmental and Technological Research (CIEMAT). His research interests include the development of scalable, resilient solvers of PDEs and numerical algebra for supercomputers. He has a long experience in simulation of turbulent compressible flow, mostly applied to aerospace propulsion. He is lecturer of space propulsion at the School of Industrial Engineering of Bilbao. His e-mail is josea.morinigo@ciemat.es and his research group website is http://rdgroups.ciemat.es/web/sci-track.

FILIPPO TERRAGNI is an Associate Professor in Applied Mathematics at the Mathematics Department of the Universidad Carlos III de Madrid. His research interests include reduced order models based on modal expansions and data-processing for problems involving fluid dynamics, pattern-forming systems, and transport phenomena. On the other hand, he is also working on modeling and numerical simulation of biological processes, like the angiogenesis. He is a lecturer in the Interuniversity Master in Industrial Mathematics (Spain). His email is fterragn@ing.uc3m.es and his research group website is https://scala.uc3m.es/.