High-Performance SVD Partial Spectrum Computation

David Keyes
Extreme Computing Research Center
Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
David.Keyes@kaust.edu.sa

Hatem Ltaief
Extreme Computing Research Center
Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
Hatem.Ltaief@kaust.edu.sa

Yuji Nakatsukasa
Mathematical Institute
University of Oxford
Oxford, United Kingdom
nakatsukasa@maths.ox.ac.uk

Dalal Sukkari
Innovative Computing Laboratory
University of Tennessee
Knoxville, USA
sukkari@icl.utk.edu

ABSTRACT

We introduce a new singular value decomposition (SVD) solver based on the QR-based Dynamically Weighted Halley (QDWH) algorithm for computing the partial spectrum SVD (`QDWHpartial-SVD`) problems. By optimizing the rational function underlying the algorithms in the desired part of the spectrum only, the `QDWHpartial-SVD` algorithm efficiently computes a fraction (say 1-20%) of the leading singular values/vectors. We develop a high-performance implementation of `QDWHpartial-SVD` [1] on distributed-memory manycore systems and demonstrate its numerical robustness. We perform a benchmarking campaign against counterparts from the state-of-the-art numerical libraries across various matrix sizes using up to 36K MPI processes. Experimental results show performance speedups for `QDWHpartial-SVD` up to 6X and 2X against vendor-optimized PDGESVD from ScaLAPACK and KSVD on a Cray XC40 system using 1152 nodes based on two-socket 16-core Intel Haswell CPU, respectively. We also port our `QDWHpartial-SVD` software library to a system composed of 256 nodes with two-socket 64-Core AMD EPYC Milan CPU and achieve performance speedup up to 4X compared to vendor-optimized PDGESVD from ScaLAPACK. We also compare energy consumption for the two algorithms and demonstrate how `QDWHpartial-SVD` can further outperform PDGESVD in that regard by performing fewer memory-bound operations.

CCS CONCEPTS

• **Mathematics of computing** → **Solvers**; **Mathematical software performance**; • **Computing methodologies** → **Distributed algorithms**; **Massively parallel algorithms**.

[1] https://github.com/ecrc/qdwhpartial-svd.

This work is licensed under a Creative Commons Attribution International 4.0 License.

SC '23, November 12–17, 2023, Denver, CO, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0109-2/23/11.
https://doi.org/10.1145/3581784.3607109

KEYWORDS

Singular Value Decomposition, Partial Spectrum, Parallel Numerical Algorithms, Distributed-Memory Systems

ACM Reference Format:
David Keyes, Hatem Ltaief, Yuji Nakatsukasa, and Dalal Sukkari. 2023. High-Performance SVD Partial Spectrum Computation. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3581784.3607109

1 INTRODUCTION

Computing the singular value decomposition (SVD) [13, 14, 41] represents one of the main computational phases for many scientific problems, e.g., signal processing [36], pattern recognition [9], and statistics [34]. There are several numerical algorithms that require only the most significant singular values with their associated singular vectors from the SVD, e.g., determining the pseudo-inverse of a matrix [27] or performing low-rank matrix approximations [2, 3, 16].

The state-of-the-art numerical libraries `LAPACK` [4] and `ScaLAPACK` [7] for shared-memory and distributed-memory systems, respectively, provide SVD implementations. They first reduce the original dense matrix into condensed bidiagonal form, before computing the singular values/vectors. Although this initial reduction phase occupies a small part of the floating-point operations (flops), it may still account for up to half of the overall time taken by the overall SVD solvers. This is due to its memory-bound execution during the expensive panel factorization based on Level-2 BLAS, which requires accessing the entire unreduced trailing submatrix. The memory bandwidth may quickly become saturated and adding more computational resources may actually further slow the execution. Two-stage matrix reductions [6, 22] for SVD have become popular, as they allow to cast some of the Level-2 BLAS operations into compute-bound Level-3 BLAS.

In many applications, one is interested only in a subset of the spectrum; usually the dominant singular triplets. However, the traditional one/two-stage reduction-based approaches still require to transform the whole matrix into bidiagonal form, so the overall runtime is comparable to that of a full decomposition.

In this paper, we design and implement algorithms that remove these aforementioned limitations in order to compute the partial spectrum for the SVD solver. Based on the polar decomposition, this new high performance SVD algorithm relies on the QR-based Dynamically Weighted Halley (QDWH) method to compute a partial spectrum. As introduced in [31], QDWH is an expensive approach requiring a much higher number of flops than standard approaches when employed to compute the full SVD spectrum [33]. However, it is ultimately a competitive approach for the SVD, since these extra flops can be executed with a higher level of concurrency and the QDWH numerical kernels are compute-bound. Here, we leverage the QDWH-based SVD algorithm to compute a partial spectrum for the SVD (QDWHpartial-SVD). Our new QDWHpartial-SVD algorithm permits to focus computational power on the operations necessary for the computation of the spectrum of interest.

We deploy our QDWHpartial-SVD implementation on a large distributed-memory system and ensure the original numerical robustness of QDWH-based full SVD is maintained, for various matrix types. We then assess its performance and compare it against its respective counterparts from the state-of-the-art numerical libraries (i.e., ScaLAPACK [7] and KSVD [37]). Experimental results show performance speedups for QDWHpartial-SVD up to 6X and 2X against PDGESVD from ScaLAPACK and KSVD, respectively.

The remainder of the paper is organized as follows. Section 2 describes related work for state-of-the-art SVD solvers. Section 3 reviews the background of the QDWH approach for the polar decomposition and its application to SVD solvers. Section 4 introduces the new QDWHpartial-SVD algorithm for computing only the partial spectrum. Section 5 describes the implementation details and Section 6 estimates the algorithmic operation counts. Section 7 highlights the numerical robustness of QDWHpartial-SVD algorithm. Section 8 assesses the achieved performance results and we conclude in Section 9.

2 RELATED WORK

For computing the *full* spectrum for SVD solvers, the state-of-the-art approaches can be split into two categories. The one-stage approaches, as implemented in LAPACK [4] and ScaLAPACK [7], reduce the dense matrix into a condensed bidiagonal form using a single phase of orthogonal transformations before extracting the spectrum of interest. To further promote Level-3 BLAS operations during this single stage, two-stage approaches [6, 22] have emerged as an efficient algorithmic alternative in better extracting the hardware performance. Although they come at the price of extra floating-point operations (flops), their high performance implementations have contributed in their wide adaption in the software ecosystem within the PLASMA [17, 18, 23–25, 28] and MAGMA [19] libraries on shared-memory systems (possibly equipped with GPUs) for SVD solvers.

When it comes to calculating the *partial* spectrum for SVD solvers, the one and two-stage approaches are inefficient as described above. A more recent work [29] shows how to partially compute the SVD out of the bidiagonal form using an associated tridiagonal eigenproblem. Yet, the condensed form remains the starting point and is one of the most expensive computational operations.

Completely different classes of algorithms for computing a small part of the spectrum have been developed, most prominently the Lanczos algorithm (more generally Krylov subspace methods) and the more recent randomized algorithms [20]. While these can be very powerful, they come with certain drawbacks in the situation that we consider.

Krylov methods are usually suitable only when a very small fraction of the spectrum (usually $O(1)$ eigenvalues or singular values) is required, and sometimes fail to provide full accuracy. In this work we consider the case where a nonnegligible portion of the spectrum (say $1-20\%$) is desired.

Randomized algorithms can be an extremely effective means of finding an approximate SVD, and are rapidly gaining popularity. However, they usually come with poorer accuracy guarantees, giving outputs that are suboptimal by an $O(1)$ factor; see [20, §10], [30, §3] (these guarantees are still remarkable—especially when the spectrum decays rapidly—and enough in many applications [1, 2, 8]).

In this paper, we propose to revisit and modify the QDWH-SVD algorithm [31, 33] in order to provide support for determining only a partial spectrum for the SVD solvers. We aim to compute the singular values and vectors essentially to full working precision. These algorithms and their high performance implementations [37–40] do not require a reduction to tridiagonal or bidiagonal forms. They iteratively compute the polar decomposition—based on conventional, compute-bound, and highly-parallel dense linear algebra operations—as a preprocessing step toward the eigenvalue and SVD solvers. By altering the core algorithmic feature of QDWH-SVD, the new QDWHpartial-SVD approach transforms directly the original dense matrix to a much smaller one, with a size roughly of the spectrum of interest. Since the transformation occurs at the beginning of the QDWHpartial-SVD procedure, the power of computational resources is tailored solely to operations that are intimately related to the spectrum of interest.

3 QDWH-BASED POLAR DECOMPOSITION AND ITS APPLICATION TO FULL SVD SOLVERS

The Polar Decomposition (PD) $A = U_p H \in \mathbb{C}^{m\times n}$, where $U_p \in \mathbb{C}^{m\times n}$ is the *unitary polar factor* with $U_p^* U_p = I_n$ and H is Hermitian positive semidefinite, exists for any matrix. It is an important matrix decomposition for various applications, including inertial navigation [5], chemistry [12], and computation of block reflectors in numerical linear algebra [35]. It can be used as a first computational phase toward computing the SVD [33] in the context of the QR-based Dynamically Weighted Halley (QDWH) method.

3.1 The QDWH-Based PD Algorithm

The dynamically weighted Halley iteration to find the PD can be summarized as follows:

$$\begin{aligned} X_0 &= A/\alpha, \\ X_{k+1} &= X_k(a_k I + b_k X_k^* X_k)(I + c_k X_k^* X_k)^{-1}. \end{aligned} \tag{1}$$

The scalars (a_k, b_k, c_k) are chosen dynamically to speed up the convergence [31]. More specifically, they are chosen so that the rational function $r_k(x) = x(a_k + b_k x^2)/(1 + c_k x^2)$ is the scaled *Zolotarev* function of type $(3, 2)$, the best rational approximation to the sign

function on $[-1,-\ell_k]\cup[\ell_k,1]$. Here $\ell_0 = 1/\kappa_2(A)$ (or its estimate) and follows the updating formula $\ell_k = r_k(\ell_{k-1})$. The singular values of X_k are $\Sigma_k = r_k(\cdots r_2(r_1(\Sigma)))$, and lie in $[\ell_k, 1]$. Remarkably, the composition of the rational functions $r_k(\cdots r_2(r_1(\Sigma)))$ is again a Zolotarev function, of much higher type $(3^k, 3^k-1)$. Together with the exponential convergence of Zolotarev functions, QDWH converges in at most *six* iterations to obtain $X_k \to U_p$ (and $\ell_k \to 1$) in double precision for matrices with $\kappa_2(A) \le 10^{15}$.

Based on the fact [21, p. 219] that $cX(I + c^2X^*X)^{-1} = Q_1Q_2^*$, where $\begin{bmatrix} cX \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R$ is the QR decomposition, with $X, Q \in \mathbb{R}^{m\times n}$ and $Q_2, R \in \mathbb{R}^{n\times n}$, the Equation (1) can be replaced with the following inverse-free and stable QR-based implementation [33]:

$$\begin{aligned} \begin{bmatrix} \sqrt{c_k}X_k \\ I \end{bmatrix} &= \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \\ X_{k+1} &= \frac{b_k}{c_k}X_k + \frac{1}{\sqrt{c_k}}\left(a_k - \frac{b_k}{c_k}\right) Q_1Q_2^*. \end{aligned} \tag{2}$$

This represents the QR-based Dynamically Weighted Halley (QDWH) algorithm. Further details can be found in [32].

After a few QDWH iterations from Equation (2), the X_k eventually become well-conditioned and $\ell_k = O(1)$. Once this happens, a lower-cost Cholesky-based iteration can be used instead, as follows:

$$\begin{aligned} X_{k+1} &= \frac{b_k}{c_k}X_k + \left(a_k - \frac{b_k}{c_k}\right)(X_kW_k^{-1})W_k^{-*}, \\ W_k &= \mathrm{chol}(Z_k),\ Z_k = I + c_kX_k^*X_k. \end{aligned} \tag{3}$$

A higher-order variant of the QDWH algorithm that employ higher-degree Zolotarev functions has been developed [32], which further increases the degree of parallelism.

3.2 Applying QDWH to Full SVD Solvers

First, we recall the mechanism of `QDWH-EIG` [33], on which the new `QDWHpartial-SVD` algorithm will be based. Let A be an $n\times n$ symmetric matrix and write

$$\begin{aligned} A &= V\,\mathrm{diag}(\Lambda_+, \Lambda_-)V^* \\ &= V\,\mathrm{diag}(I_{n-k}, -I_k)V^* \cdot V\,\mathrm{diag}(\Lambda_+, |\Lambda_-|)V^* \\ &\equiv U_pH, \end{aligned} \tag{4}$$

where U_pH is the polar decomposition [21, Ch. 8]. k is the number of negative eigenvalues, which we do not assume to be known. Suppose that the unitary polar factor U_p is computed. This means we have mapped all the eigenvalues to 1 or -1. We partition $V = [V_+,\ V_-]$ conformably with Λ, and note that

$$\begin{aligned} \frac{1}{2}(I - U_p) &= \frac{1}{2}\left(I - [V_+\ V_-]\begin{bmatrix} I_{n-1} & 0 \\ 0 & -I_k \end{bmatrix}[V_+\ V_-]^*\right) \\ &= [V_+\ V_-]\begin{bmatrix} 0 & 0 \\ 0 & I_k \end{bmatrix}[V_+\ V_-]^* \\ &= V_-V_-^*. \end{aligned} \tag{5}$$

Hence the symmetric matrix $C = \frac{1}{2}(I-U_p) = V_-V_-^*$ is an orthogonal projector onto $\mathrm{Span}(V_-)$, the invariant subspace corresponding to the negative eigenvalues. We can then project the matrix A (Rayleigh-Ritz process) to obtain the eigenvalues and eigenvectors: the eigenvalues of $V_-^*AV_-$ are equal to those of Λ_-, and denoting by $V_-^*AV_- = W\Lambda_-W^*$ the eigenvalue decomposition, we see that V_-W is the matrix of eigenvectors. Analogously, we can obtain V_+ by finding the subspace spanned by $\frac{1}{2}(I+U_p)$.

Second, the polar decomposition can be also used directly toward calculating the SVD, i.e., $A = U_pH = U_p(V\Sigma V^*) = (U_pV)\Sigma V^\top = U\Sigma V^*$ where Σ is the matrix containing all the singular values, and U and V are the orthogonal matrices containing the left and right singular vectors, respectively. The resulting `QDWH-SVD` procedure relies on `QDWH-EIG` (or any other eigensolvers) to compute the intermediate eigendecomposition $H = V\Sigma V^*$ required for the final SVD.

4 LEVERAGING QDWH FOR COMPUTING THE PARTIAL SPECTRUM OF SVD SOLVERS

In this section, we present modified versions of `QDWH-SVD` to extract the leading singular values and their corresponding singular vectors (`QDWHpartial-SVD`). In what follows we treat nonreal matrices $A \in \mathbb{C}^{n\times n}$; when A is real, the superscripts $\cdot^*$ should be replaced by $\cdot^\top$ and everything can be executed using only real arithmetic.

4.1 The Idea at the Core of QDWHpartial-SVD

Given a general matrix $A \in \mathbb{C}^{m\times n}(m \ge n)$, we consider the task of computing its dominant singular triplets, namely computing the singular values and singular vectors corresponding to the singular values above a given user-specified relative threshold $s > 0$ (to simplify the discussion, below we assume we work with $\tilde{A} := A/\alpha$ such that $\|A\|_2 \lesssim 1$; this can be enforced using an inexpensive norm estimator $\alpha \gtrsim \|A\|_2$.

Recall that the mathematics underlying the `QDWH-EIG` and `QDWH-SVD` algorithms is rational approximation: find a rational function r that approximates the sign function, so that r maps all the singular values to 1, while preserving the singular vectors.

Now, suppose that we only need the singular triplet U_1, Σ_1, V_1 corresponding to the singular values larger than s. Can we cut corners? The answer is yes—and this is not just that we can skip computing the trailing singular vectors V_* once the unitary polar factor U_p is computed. We shall avoid computing U_p all together. Essentially, we need to ensure only that the singular values larger than s are mapped to 1; the singular values in $[0, s)$ are irrelevant. Namely, the idea is to find a rational function r of the form $r(x) = xf(x^2)$ such that $|1 - r(x)| \le 10^{-15}$ for $x \in [s, 1]$. Note that when evaluated at a matrix argument we define $r(\tilde{A}) = \tilde{A}f(\tilde{A}^T\tilde{A})$. In other words, the idea is simply to compute $r(A) := Ur(\Sigma)V^*$, where r is a rational function that maps the interval $[s, 1]$ to 1, to machine precision. The upshot is that if $s \gg \sigma_{\min}(A)$, then r is allowed to be of much lower degree than would be needed for computing U_p.

4.2 Impact of the s Interval on Convergence

It is worth noting that the degree of r and the number of QDWH iterations depends on the user-defined value s. For example, as illustrated in Figures 1 and 2, if $s > 0.87$, two QDWH iterations suffice to get $|1 - r(x)| \le 10^{-15}$, while for $s \in [0.1, 0.87]$ we need three iterations (after k QDWH iterations, the rational function r is of type $(3^k, 3^k-1)$), respectively. With $k = 4$ iterations, $s = 10^{-4}$. For the choice of k for smaller s, see Table 1 of [32]. Note that if $s \ll 1$ (say $s < 10^{-3}$), for the first QDWH iteration it is advisable

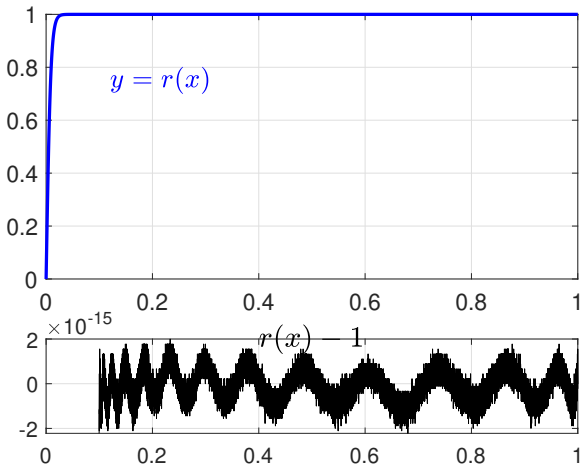


Figure 1: A type $(27, 26)$ Zolotarev function r (top) that maps the interval $[s, 1]$ to 1 for $s = 0.1$, and its the error $r - 1$ (below). Note the error is $O(u)$ on $[s, 1]$; illustrating that three QDWH iterations is enough to find the singular values larger than $s = 0.1$.

to use the QR-based implementation in Eq. (2) rather than Eq. (3) to avoid instabilities.

We also note that we use $|\ell_k - 1| < O(u)$ as the stopping criterion for QDWH. This is because the standard condition, which requires convergence of X_k, is not necessarily satisfied when $[s, 1]$ has been mapped to $1 \pm O(u)$, because the singular values $\sigma_i(\tilde{A}) < s$ have not converged and lie somewhere in $[0, 1]$.

4.3 Computing the Corresponding Singular Subspace

We would like now to compute the corresponding singular subspace. Suppose that we have computed $r(\tilde{A})$. It has w (or more) singular values equal to 1, where w is the number of singular values of $\tilde{A}$ lying in $[s, 1]$. One can find the desired row space V_1 by finding the null space of $I - r(\tilde{A})^* r(\tilde{A})$. Although computing the matrix $r(\tilde{A})^* r(\tilde{A})$ may seem ill-advised, as it squares the condition number, this is not an issue here, as the quantity of interest is the singular subspace corresponding to the largest singular values of $r(\tilde{A})$ (which are approximately 1).

To compute the null space of $I - r(\tilde{A})^* r(\tilde{A})$ (or a larger subspace that contains it), we compute the QR factorization

$$I - r(\tilde{A})^* r(\tilde{A}) = [Q_1, Q_2] \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}, \tag{6}$$

where the size of R_{11} (and hence of $R_{22} \in \mathbb{C}^{\ell \times \ell}$, where $\ell \geq w$) is chosen so that it only has "large" singular values; the idea is that Q_2 then contains the null space of $I - r(\tilde{A})^* r(\tilde{A})$ that we require. To quantify the claim we use matrix perturbation theory.

Theorem 4.1. *Let $B = [Q_1, Q_2] \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \in \mathbb{C}^{p \times n} (p \leq n)$ be a full QR factorization such that $[Q_1, Q_2] \in \mathbb{C}^{p \times p}$ is unitary with $Q_2 \in \mathbb{C}^{p \times \ell}$ and $R_{11} \in \mathbb{C}^{(p-\ell) \times (p-\ell)}$. Let $V_0 \in \mathbb{C}^{p \times w}$ have orthonormal columns $V_0^* V_0 = I_w$, with $w \leq \ell$. Then*

$$\sin \angle(V_0, Q_2) \leq \frac{\|V_0^* B\|_2}{\sigma_{\min}(R_{11})}. \tag{7}$$

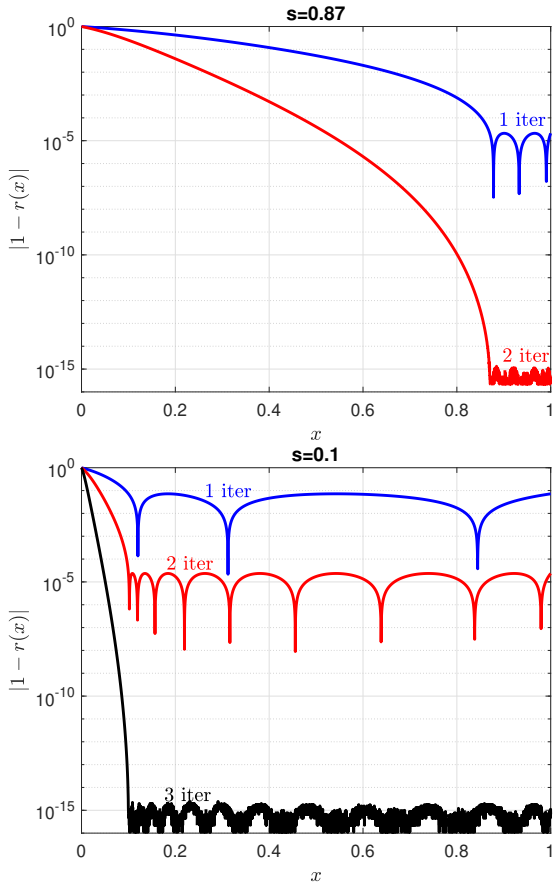


Figure 2: Semilog plot of $|1 - r(x)|$ for varying number of QDWH iterations; The rational function r is of type $(3, 2)$ with one iteration, $(9, 8)$ with two, and $(27, 26)$ with three. Top: $s = 0.87$, bottom: $s = 0.1$.

Remark. The main situation of interest is when V_0 spans an approximate left null space of B such that $\|V_0^* B\|_2 = O(u\|B\|_2)$, where u is unit roundoff; then the theorem shows the subspace V_0 is approximately contained in Q_2, up to $O(u/\sigma_{\min}(R_{11}))$.

Proof. First recall that the canonical angles $\angle_1(V_0, Q_1), \ldots, \angle_w(V_0, Q_1)$ between two subspaces of dimensions $w, \ell (\ell \geq w)$ spanned by the orthonormal matrices $V_0 \in \mathbb{C}^{p \times w}$ and $Q_2 \in \mathbb{C}^{p \times \ell}$ (for which Q_1 is the orthogonal complement $Q_1^* Q_2 = 0$) are defined by $\sin \angle_i(V_0, Q_1) = \sigma_i(V_0^* Q_1)$ [14, § 2] for $i = 1, \ldots, w$. It thus suffices to show that $\|V_0^* Q_1\|_2 \leq \frac{\|V_0^* B\|_2}{\sigma_{\min}(R_{11})}$.

Now we have

$$V_0^* B = [V_0^* Q_1, V_0^* Q_2] \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} = \left[V_0^* Q_1 R_{11}, V_0^* Q_1 R_{12} + V_0^* Q_2 R_{22} \right].$$

Hence we obtain $\|V_0^* B\|_2 \geq \|V_0^* Q_1 R_{11}\|_2 \geq \|V_0^* Q_1\|_2 \sigma_{\min}(R_{11})$. It follows that $\|V_0^* Q_1\|_2 \leq \frac{\|V_0^* B\|_2}{\sigma_{\min}(R_{11})}$, as required. □

Let us make two remarks about the theorem.

- While Theorem 4.1 allows for rectangular B ($p \leq n$), it does not apply directly to the $p > n$ case, as then the left null space V_0 is larger than the rank deficiency of B.
- Finding a numerical null space of a matrix is a classical problem in numerical linear algebra, and a reliable algorithm is usually based on either the SVD or a strong rank-revealing QR factorization [15]. The assumptions in the theorem are much weaker; the reason they suffice is that (7) only states that the null space V_0 is *contained* in (and not equal to) Q_2; in particular, it does not claim $\|BQ_2\|_2$ is small. Once such Q_2 is obtained, our algorithm will extract a null space of BQ_2 by projecting the original matrix A to Q_2, that is, compute AQ_2 and find its SVD, and extract its dominant singular triplets.

When Theorem 4.1 is applied with $B \leftarrow I - r(\tilde{A})^* r(\tilde{A})$ as in (6) (so $p = n$), it gives $\sin \angle(V_0, Q_2) \leq \frac{\|V_0^*(I - r(\tilde{A})^* r(\tilde{A}))\|_2}{\sigma_{\min}(R_{11})}$, where V_0 is the numerical null space of $r(\tilde{A}) + I$, which has dimension $\geq w$ by construction. Therefore $\|V_0^*(r(\tilde{A}) + I)\|_2 = O(u)$, and it follows that we have $\sin \angle(V_0, Q_2) \leq O(u/\sigma_{\min}(R_{11}))$, so Q_2 contains V_0 to working accuracy provided that $\sigma_{\min}(R_{11}) \geq tol$ for some tolerance $tol = \Omega(1)$, for example $tol = 0.01$, which is the choice we make by default. Choosing it too large, $tol \approx 1$, results in $\ell \approx n$ so we get no computational savings, while $tol \ll 1$ causes loss of accuracy.

4.4 Introducing the QDWHpartial-SVD Algorithm

Algorithm 1 presents the the overall algorithm. The outputs of QDWHpartial-SVD are U_1, Σ_1, V_1^* such that $U_1 \Sigma_1 V_1^*$ is the truncated SVD of A, truncated at the first singular value smaller than s.

Algorithm 1 QDWHpartial-SVD. Given $A \in \mathbb{C}^{m \times n}$ ($m \geq n$) and threshold $s > 0$, compute the singular values larger than $s\|A\|_2$ and the corresponding singular vectors.

1: Estimate $\alpha \approx \|A\|_2$ with a norm estimator.
2: Apply QDWH to $\tilde{A} := A/\alpha$ with $\ell_0 = s$ until $|\ell_k - 1| < O(u)$ to obtain $r(\tilde{A})$.
3: Calculate $[Q\ R] = QR(I - r(\tilde{A})^* r(\tilde{A}))$.
4: Find the index $ind = \min(find(abs(diag(R)) < tol = 0.01))$.
5: Extract $Q_2 = Q(:, ind : end)$.
6: Projection: Compute AQ_2 and its SVD $AQ_2 = \tilde{U}\tilde{\Sigma}\tilde{V}^*$.
7: Extract the singular triplets with singular values larger than $s\|A\|_2$, call them $\tilde{U}_1, \tilde{\Sigma}_1, \tilde{V}_1$.
8: Let $U_1 := \tilde{U}_1, \Sigma_1 := \tilde{\Sigma}_1$ and $V_1 := Q\tilde{V}_1$.

The computational savings comes from the fact that AQ_2 is much thinner than A; the number of columns of AQ_2 is slightly more than the number of singular values of A larger than s. Of course, if $m < n$ one can simply apply the algorithm to A^*.

5 IMPLEMENTATION DETAILS

Algorithm 2 describes the pseudo-code of the distributed-memory implementation of QDWHpartial-SVD based on ScaLAPACK [7].

Following the 2D Block-Cyclic Data Distribution (2D-BCDD) used in ScaLAPACK, we define the MPI process grid configuration as $P \times Q$. Each data structure owns a handle or a descriptor that expresses how the data structure is distributed following the 2D-BCCD. ScaLAPACK relies on the Basic Linear Algebra Communication Subprograms (BLACS) library, which is in charge of performing data movements during the matrix computations through the traditional MPI. ScaLAPACK relies on block algorithms, which can be expressed by two successive computational stages: the panel factorization and the update of the trailing submatrix. While the former is memory-bound, and typically sequential and may not benefit from having many processors participating, the latter is rich in compute-bound operations. This is where most of ScaLAPACK dense linear algebra operations extract parallel performance by means of calls to Level-3 BLAS, as implemented in the Parallel BLAS (PBLAS) layer. The blocking size referred to as nb is an internal tuning parameter that trades off the degree of parallelism and the performance of the computational kernels. Moreover, the number of processors should be properly calibrated into a rectangular shape with $Q > 1.5P$ to carry on, in parallel, the update of the trailing submatrix.

As shown in Algorithm 2, the QDWHpartial-SVD code is mostly composed of conventional dense linear algebra matrix kernels rich in compute-intensive Level-3 BLAS operations that are capable of achieving a decent percentage of the system's theoretical peak performance. Since these matrix kernels are widely available in vendor optimized numerical libraries, porting to various hardware architectures should not be cumbersome. The code is written in double precision arithmetics, and can be extended to other precisions for a broader application coverage.

6 OPERATION COUNTS

Table 1 reports the operation counts of various SVD solvers on square matrices of size n: the PDGESVD and QDWH-SVD routines for computing the full spectrum and the QDWHpartial-SVD routines for computing a subset of the spectrum. We refer the reader to [33] for further details on the costs of the standard and QDWH-based SVD solvers.

The operation count of QDWHpartial-SVD depends on the number of Cholesky-based QDWH iterations (typically two or three) and, the QR, GEMM and SYRK to form the reduced problem matrix of size $N_s \geq s$, with s the size of the partial spectrum of interest. The actual full SVD occurs now only on the reduced problem matrix of size N_s. Assuming $N_s \ll N$ and three Cholesky-based iterations to get the polar factor from QDWH, the total number of operations is up to $24N^3$ for QDWHpartial-SVD ($it_{QR} = 1$ and $it_{Chol} = 3$).

SVD variants	Cost
Standard full SVD	$17N^3$
Full QDWH-SVD	$20N^3 \leq \cdots \leq (50 + \frac{1}{3})N^3$
QDWHpartial-SVD	QDWH: (8+2/3)N^3 x #it_{QR} + (4+1/3)N^3 x #it_{Chol} QR + SYRK + 2×GEMM: $4/3N^3 + N^3 + 4NN_s^2$ SVD: $17N_s^3$

Table 1: Operation counts for various SVD algorithms.

Algorithm 2 Pseudo-code of the QDWHpartial-SVD using ScaLAPACK.

/* **Set block size and initiate CBLACS context** */
$Cblacs\_get(0, 0, ictxt)$
$Cblacs\_gridmap(ictxt, imap, P, P, Q)$
$Cblacs\_gridinfo(ictxt, P, Q, myrow, mycol)$
/* **Initialize data structures using the 2D-BCDD descinit()** */
$descinit(nb, nb, A, descA); Fill\_in(A, descA)$
/* **Estimate the two-norm of the matrix** */
$\alpha = pdgenm2(A)$
$pdlascl(\alpha, 1., A)$
/* **Computing the polar factor U_p of the matrix A using QDWH** */
$k = 1,\ Li = threshold(s),\ conv = 100$
while $(|Li - 1| \geq 5eps)$ **do**
  $L2 = Li^2,\ dd = \sqrt[3]{(4(1 - L2)/L2^2)}$
  $sqd = \sqrt{1 + dd}$
  $a1 = sqd + \sqrt{8 - 4 \times dd + 8(2 - L2)/(L2 \times sqd))}/2$
  $a = real(a1);\ b = (a - 1)^2/4;\ c = a + b - 1$
  $Li = Li(a + b \times L2)/(1 + cL2)$
  pdlacpy(U, U1)
  /* **Compute U_k from U_{k-1}** */
  $pdlaset(Z, 0., 1.)$
  $pdgemm(U^\top, U, Z)$
  $pdgeadd(U, B)$
  $pdposv(Z, B)$
  $pdgeadd(B, U)$
  $pdgeadd(U, U1)$
  $pdlange(U1, conv)$
  $k = k + 1$
end while
/* **U_p contains the isolated subspectrum of interests** */
$pdlaset(0.0, 1.0, B)$
$pdgemm(U_p^\top, U_p, B)$
$pdgeqrf(B)$
$ind = min(find(abs(diag(B)) < tol = 0.01))$
$pdorgqr(B, Q)$
/* **size**$(\tilde{A}) = N - ind$ */
$Q_2 = Q(:, ind{:}end)$
$pdgemm(A, Q_2, \tilde{A})$
/* **Calculate the SVD on the reduced problem** */
$pdgesvd(\tilde{A}, \tilde{U}, \tilde{\Sigma}, \tilde{V})$
/* **$\tilde{U_1}, \tilde{\Sigma_1}, \tilde{V_1}$ are the singular triplets with singular values larger than the $s \times \alpha$** */
$U = \tilde{U_1}$
$\Sigma = \tilde{\Sigma_1}$
$pdgemm(\tilde{V_1}, Q_2^T, V)$

7 NUMERICAL ACCURACY

The numerical accuracy of the QDWH-based algorithms to compute the polar decomposition and singular value decomposition (QDWH-SVD) have been verified in [33]. The robustness of their high performance implementations has been studied on shared-memory systems [39] and on distributed-memory systems [37, 40]. In this Section, we present the numerical robustness of the QDWHpartial-SVD implementation on distributed-memory system.

7.1 Environment Settings

We run our experiments on the *Shaheen-2* Cray XC40 system with the Cray Aries network interconnect, which implements a Dragonfly network topology. The system has 6174 compute nodes, each with two-socket 16-core Intel Haswell running at 2.3GHz and 128GB of DDR3 main memory.
We run also on a second system with 256 nodes based on two-socket 64-core AMD EPYC 7763 Milan CPUs, clocked at 2.45 GHz with 512 GB DDR4-3200. This system is provisioned with Slingshot 11 network interconnect and is a proxy of the *LUMI* Supercomputer. The work load managers on both systems is native SLURM. We use the Intel compiler and rely on the vendor-optimized ScaLAPACK implementation from the high performance Cray LibSci numerical library, which depends on the MPI programming model for inter-node communications. We refer to Intel Haswell and AMD Milan for the former and latter systems, respectively.
All runs for a given process configuration have been submitted in the same job submission script to reduce the impact from the system jitter. All the experiments are performed using IEEE double-precision arithmetic.

7.2 Synthetic Matrices

The dense synthetic matrices $A \in \mathbb{R}^{N \times N}$ are generated using the ScaLAPACK routine PDLATMS $A = Q_1 D {Q_2}^\top$ with mode set to 0. For testing the SVD solvers, the distribution of the singular values of the generated matrices follows a geometrical series: $D[i] = (0.5)^{\frac{i}{N} * 100}$. We compute then orthogonal matrices Q_1 and Q_2 generated by calculating the QR factorization of arbitrary matrices to form the SVD, while $Q_1 = Q_2$ for the symmetric EIG solvers. The performance of the matrix generation step may be expensive and can be improved [11] but this is beyond the scope of this paper.

7.3 Norm Definitions

For a given general matrix $A \in \mathbb{R}^{N \times N}$, let $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_k)$ be the k computed singular values, and U and V be the corresponding k computed left and right singular vectors. The norm $\| \ . \ \|_F$ denotes the Frobenius norm. The accuracy assessment of the partial computation of the SVD is based on the following metrics:

$$\frac{\|I - UU^\top\|_F}{n} \text{ and } \frac{\|I - VV^\top\|_F}{n}, \tag{8}$$

for the orthogonality of the left and right k computed singular vectors U and V, respectively, and

$$\frac{\|\Sigma - \Delta\|_F}{\|\Delta\|_F}, \tag{9}$$

for the accuracy of the k computed singular values Σ, where Δ is the k exact singular values (analytically known), and

$$\max_i(\|AU(:, i) - \sigma_i V(:, i)\|_F) \text{ and } \max_i(\|AV(:, i) - \sigma_i U(:, i)\|_F) \tag{10}$$

for the accuracy (residuals) of the singular value triplets. Similarly, for a symmetric matrix $A \in \mathbb{R}^{n \times n}$, the accuracy of the computed k negative eigenvalues, the orthogonality of their corresponding eigenvectors and the overall residual can be accordingly measured using Equations (8), (9) and (10), respectively.

7.4 Accuracy Assessments against Full SVD Solvers

This section highlights the numerical robustness of the QDWHpartial-SVD implementation. Fig. 3 (a, b, c) shows the numerical accuracy of the computed singular values (Equation 9), the orthogonality of their corresponding singular vectors (Equation 8), and the right/left residual of the computed SVD (Equation 10) on a 16×36 grid configuration (similar accuracy results for larger grid sizes 32×72, 64×144 and 128×288) using synthetic ill-conditioned matrices. Herein, we compare the accuracy of three implementations of the SVD solvers: QDWHpartial-SVD (for different threshold settings s) against PDGESVD from ScaLAPACK and from KSVD[2] [37]. The QDWHpartial-SVD is capable of extracting only the singular values/vectors of interest within the user-defined threshold s. This threshold s can be tuned with *a priori* knowledge on the singular value distribution (e.g., globally low-rank structure). This tunable parameter can directly influence the number of the computed singular values/vectors. For instance, in Fig. 3 (a, b, c), we study the accuracy for $s = 0.1, 0.01, 0.001, 0.0001$, which translates into the percentages 3%, 7%, 10%, 13% of the computed singular values/vectors, respectively, and as a result affects the performance of QDWHpartial-SVD. It is noteworthy that the ScaLAPACK *PDGESVD* computes first the whole SVD, then the requested singular values/vectors are filtered out using the threshold parameter s.

These extensive numerical tests in Fig. 3 demonstrate the numerical robustness of QDWHpartial-SVD in providing satisfactory accuracy up to the machine precision for double precision computations across all studied matrix sizes.

7.5 Accuracy Comparison Against Randomized SVD

The randomized SVD popularized by Halko, Martinsson and Tropp [20] is a highly efficient approach for finding a low-rank approximation to a matrix $A \in \mathbb{C}^{m \times n}$. The algorithm is very simple: draw a random matrix $\Omega \in \mathbb{R}^{n \times (\tilde{k})}$ (where $\tilde{k} > k$ for a rank-k approximation), and compute the QR factorization $Q\Omega = QR$. Then QQ^TA is the low-rank approximation. One can find its SVD via the SVD of Q^TA.

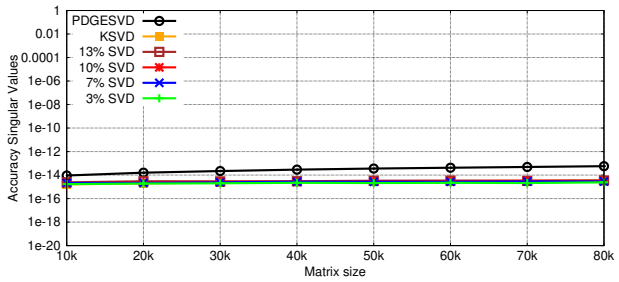
We compare the quality of solutions obtained by QDWHpartial-SVD with that of randomized SVD. The goal is to compute the singular triplets corresponding to the singular values in $[0.1, 1]$ as accurately as possible.

We generate an $n \times n$ matrix whose singular values decay geometrically as $1, \delta, \delta^2, \ldots, \delta^n$ with $\delta = 0.9$, $n = 2000$, with singular vector matrices generated randomly from the Haar distribution. We take $s = 0.1$ and run QDWHpartial-SVD (which finds the correct 0.1-rank, namely 22) and randomized SVD (with oversampling parameter 10), and examine the residuals (10). The results are summarized in the table below.

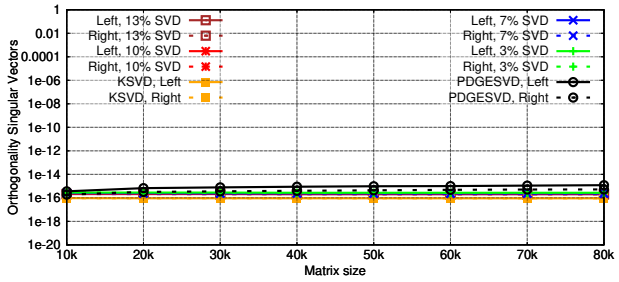
We see that while both algorithms found a good low-rank approximation, QDWHpartial-SVD finds the singular triplets with significantly better accuracy, with much smaller residual. This is perhaps not surprising, given that the purpose of randomized SVD is simply to find a low-rank approximation A_k with roughly $\|A - A_k\| = O(\sigma_k)$, while QDWHpartial-SVD by design attempts to compute the leading singular triplets to almost working precision.

We conclude that if the goal is to find the dominant singular triplets accurately, QDWHpartial-SVD significantly outperforms randomized SVD.

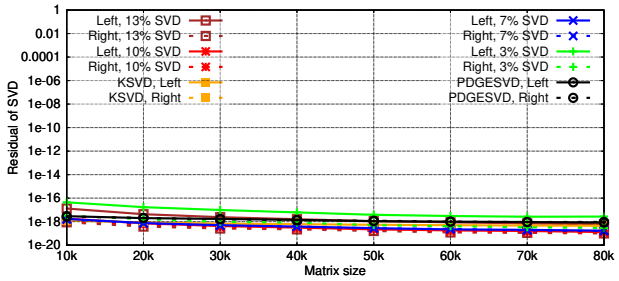
[2] Available at https://github.com/ecrc/ksvd



(a) Accuracy of singular values.



(b) Orthogonality of left/right singular vectors.



(c) Backward stability.

Figure 3: Assessment of numerical accuracy/robustness using 16×36 grid topology for several SVD solver variants.

Table 2: Comparison of QDWHpartial-SVD and randomized SVD. The matrix has $\sigma_{k+1}(A)$ = 9.8477e-02, and with QDWHpartial-SVD the approximation error $\|A - \hat{A}_k\|_2$ is equal to $\sigma_{k+1}(A)$ to working precision $< 10^{-16}$.

	residual (10)	$\|A - \hat{A}_k\|_2$
QDWHpartial-SVD	5.6e-13	9.8477e-02
Randomized SVD	9.2e-2	1.1438e-01

8 PERFORMANCE RESULTS

This section studies the impact of processor mapping on performance, compares performance of QDWHPartial-svd against its counterparts, assesses performance scalability, and reports on energy consumption.

8.1 Performance Impact of Rank Reordering

Rank reordering is an important non-intrusive optimization that permits to properly map processor grid onto the network topology. This optimization is done offline and supported by MPICH library. It identifies how to reorder a given processor grid given the hardware specifications such as the number of nodes, the number of cores, and the underlying network topology. Once the proper ordering is determined, we provide SLURM with the mapping and can execute the code on the specified process grid.

Figure 4 shows the performance impact when enabling (or not) row-wise ordering on the elapsed time of QDWHPartial-svd, when requesting 13% of the spectrum using a 128×512 processor grid on the AMD Milan system. The speedup is significant (up to 3X) and highlights the importance of this optimization when running on large-scale systems [10]. For all subsequent results, we perform this optimization by default.

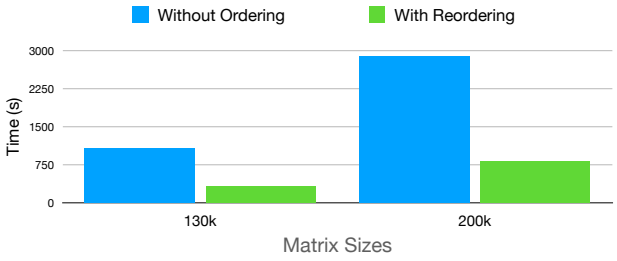


Figure 4: Performance impact of rank reordering on QDWHpartial-SVD with 13% using a 128×512 processor grid on the AMD Milan system.

8.2 Performance Comparisons

Figures 5 highlights the performance comparisons of QDWHpartial-SVD against the two other SVD solvers, ScaLAPACK *PDGESVD* and KSVD [37] across various matrix sizes and process grid configurations on the Intel Haswell system. When only 13% of the spectrum is needed, QDWHpartial-SVD achieves performance superiority as we increase the matrix sizes up to $[6.3X, 2X]$ on 16×36, $[6X, 2X]$ on 32×72, $[3.3X, 2.3X]$ on 64×144 and $[4X, 1.8X]$ on 128×288 grid topologies against [ScaLAPACK *PDGESVD*, KSVD], respectively. Moreover, on 16×36 grid configuration, QDWHpartial-SVD achieves similar performance when only 13%-10%-7% of the spectrum is needed, since the common QDWH-based polar decomposition accounts for the most time consuming computational phase. We observe a slightly faster time to solution though, when only 3% of the spectrum is calculated, since the size of the reduced problem may be relatively smaller than the aforementioned partial spectra.

Figure 6 shows similar performance comparisons of QDWHpartial-SVD against vendor-optimized *PDGESVD* on the more recent AMD Milan hardware system, achieving up to 4X performance speedup on various process grids. Although the performance comparisons of QDWHpartial-SVD across both systems are similar, it is noteworthy to mention that we use 10% less overall MPI processes on the AMD Milan system than the Intel Haswell system. The higher memory bandwidth of Milan compared to Haswell does not play a major role here since QDWHpartial-SVD is mostly compute-intensive.

8.3 Performance Scalability

Figures 7 and 8 show respectable performance scalability for QDWHpartial-SVD as the matrix sizes increases using various grid topologies on the Intel Haswell and AMD Milan systems, respectively. Notice also that the various slopes flatten for both solvers, since the critical computational phase, i.e., the QDWH-based polar decomposition, enters into the compute-bound regime of operations along with a better hardware occupancy. It is worth emphasizing that the size of the reduced problem may sometimes be slightly larger than the number of singular values requested.

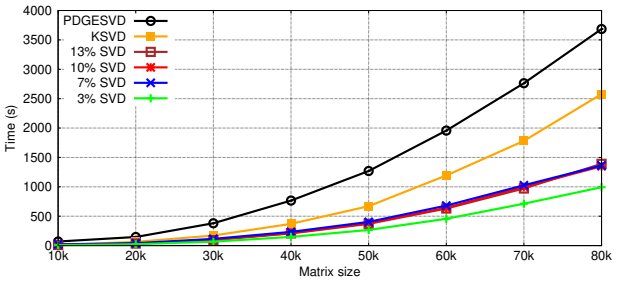
8.4 Energy Consumption

Given that QDWHpartial-SVD is algorithmically different from the state-of-the-art two-sided bidiagonal reduction SVD solver (i.e., PDGESVD), studying energy consumption is paramount in assessing the impact of our contributions, beyond the usual time-to-solution metric.

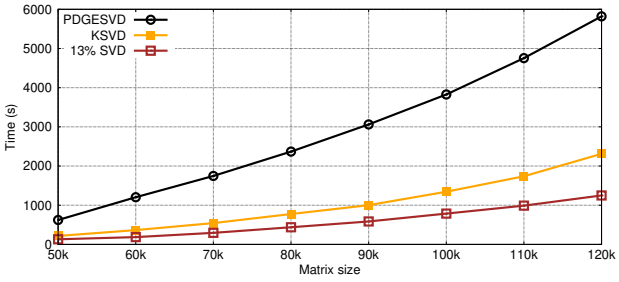
Figure 9 provides such insight by reporting the energy consumption (in MJ) of QDWHpartial-SVD and PDGESVD when only 13% of the spectrum is required using the 128×256 process grid on the AMD Milan system. We look at two matrix sizes, i.e., the smallest and largest ones, for which the corresponding timing results are reported in Figure 6. The 4X energy saving for both matrix sizes exceeds the 3.5X and 3.7X performance speedups for the corresponding matrix sizes. This is because PDGSEVD performs lots of memory-bound operations (i.e., Level-2 BLAS) during the panel factorization to form the condensed bidiagonal form before launching the compute-bound phase of the trailing submatrix updates. In contrast, QDWHpartial-SVD is built upon one-sided factorizations (i.e., Cholesky and QR) that do not require the expensive Level-2 BLAS kernel calls of PDGESVD. This gain in energy saving is higher for the smallest matrix size since the workload per process may not be large enough to hide the expensive data movement required by the Level-2 BLAS kernel calls for PDGESVD.

9 SUMMARY AND FUTURE WORK

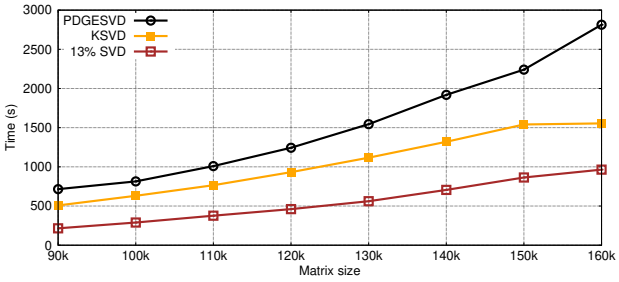
This paper introduces a new algorithm for computing a partial spectrum of the singular value decomposition. By relying on QDWH-based polar decomposition, we demonstrate performance advantages in execution time and energy for QDWHpartial-SVD against counterpart routines from state-of-the-art open-source (i.e., KSVD)



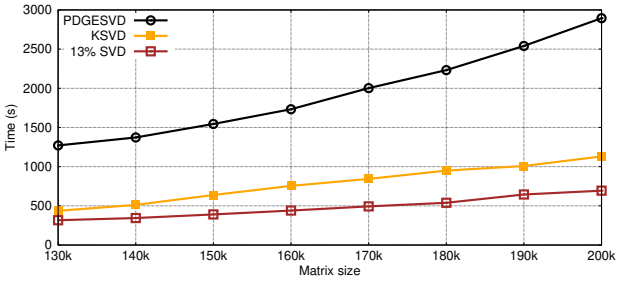
(a) P=16 and Q=36.



(b) P=32 and Q=72.

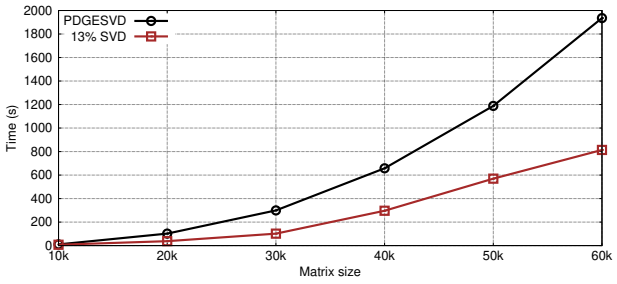


(c) P=64 and Q=144.

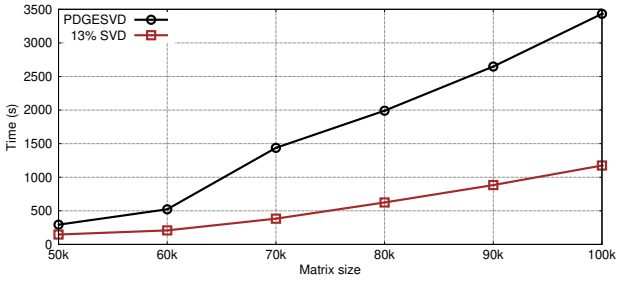


(d) P=128 and Q=288.

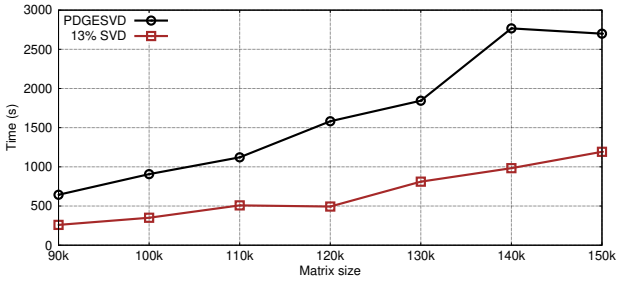
Figure 5: Performance comparisons in seconds of SVD solvers for various grid topologies on the Intel Haswell system.



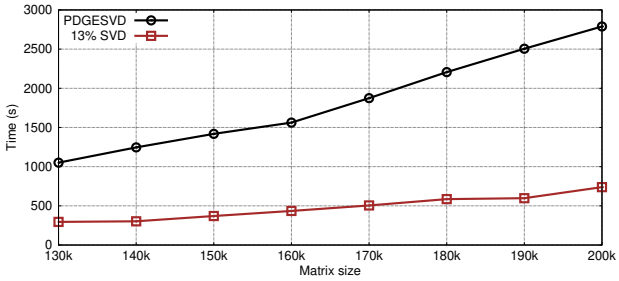
(a) P=16 and Q=32.



(b) P=32 and Q=64.



(c) P=64 and Q=128.



(d) P=128 and Q=256.

Figure 6: Performance comparisons in seconds of SVD solvers for various grid topologies on the AMD Milan system.

and vendor-optimized (i.e., ScaLAPACK from Cray Scientific Library) numerical libraries, without compromise in accuracy or robustness.

For the same cutoff, we also show substantial advantages in accuracy compared to the popular randomized SVD, whose primary application is somewhat different, namely to compute the principal partial spectrum to a given cutoff at low cost, for purposes such as low-rank approximation.

The ratio of savings in time and energy over the other highly accurate algorithms depends upon matrix dimension and on the fraction of the spectrum of SVD triplets desired, with the largest

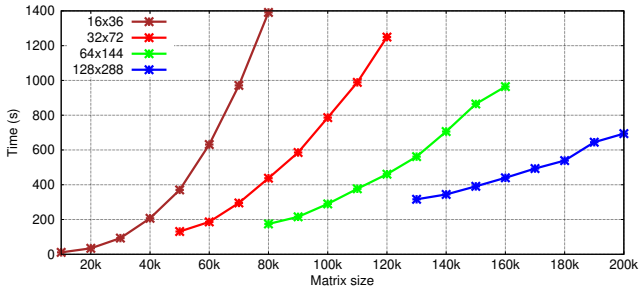


Figure 7: Performance scalability of `QDWHpartial-SVD` with 13% of the spectrum for various grid topologies on the Intel Haswell system.

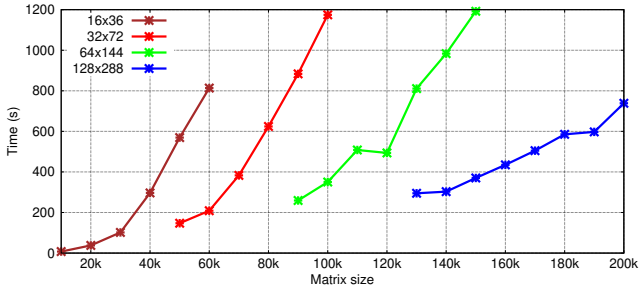


Figure 8: Performance scalability of `QDWHpartial-SVD` with 13% of the spectrum for various grid topologies on the AMD Milan system.

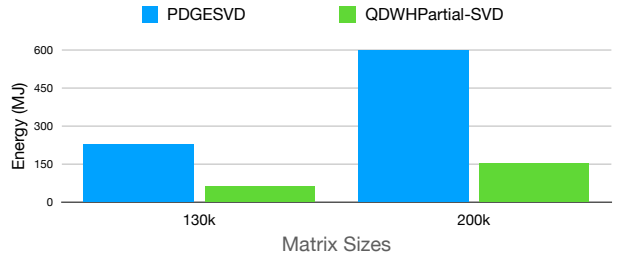


Figure 9: Energy consumption of `QDWHpartial-SVD` with 13% of the spectrum for the 128×256 process grid on the AMD Milan system.

ratios coming at large matrix dimension and small fraction. Even for substantial fractions of the spectrum, the savings can be very significant, such as $4X$ for 13% of the spectrum for an AMD Milan system on a matrix of dimension 200K. Energy ratio improvements are slightly greater than execution time ratio improvements, due to less data motion in distributed memory environments.

We plan to further improve our current implementation by using ZOLO-based polar decomposition [26]. Recent work on leveraging task-based programming model associated with dynamic runtime systems for tackling heterogeneous hardware environment [38] may also be considered to further speed up the current implementation on distributed-memory systems equipped with GPU accelerators.

ACKNOWLEDGMENTS

For computer time, this research used *Shaheen-2* supercomputer hosted at the Supercomputing Laboratory at KAUST. We would like to thank Aniello Esposito from HPE for providing remote access on the AMD Milan system and conducting the corresponding experiments.

REFERENCES

[1] Sameh Abdulah, Hatem Ltaief, Ying Sun, Marc G Genton, and David E Keyes. 2018. ExaGeoStat: A High Performance Unified Software for Geostatistics on Manycore Systems. *IEEE Transactions on Parallel and Distributed Systems* 29, 12 (2018), 2771–2784.

[2] Kadir Akbudak, Hatem Ltaief, Aleksandr Mikhalev, Ali Charara, Aniello Esposito, and David Keyes. 2018. Exploiting Data Sparsity for Large-Scale Matrix Computations. In *Euro-Par 2018: Parallel Processing*, Marco Aldinucci, Luca Padovani, and Massimo Torquati (Eds.), Vol. 11014. Springer International Publishing, Cham, 721–734.

[3] Patrick Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari, Jean-Yves L'Excellent, and Clément Weisbecker. 2015. Improving Multifrontal Methods by Means of Block Low-Rank Representations. *SIAM Journal on Scientific Computing* 37, 3 (2015), A1451–A1474. https://doi.org/10.1137/120903476

[4] Edward Anderson, Zhaojun Bai, Christian Heinrich Bischof, Laura Susan Blackford, James Weldon Demmel, Jack J Dongarra, Jeremy J Du Croz, Anne Greenbaum, Sven Hammarling, A McKenney, and Danny C Sorensen. 1999. *LAPACK User's Guide* (3rd ed.). SIAM, Philadelphia.

[5] I.Y. Bar-Itzhack. 1975. Iterative Optimal Orthogonalization of the Strapdown Matrix. *IEEE Trans. Aerospace Electron. Systems* AES-11, 1 (Jan 1975), 30–37. https://doi.org/10.1109/TAES.1975.308025

[6] Christian H. Bischof, Bruno Lang, and Xiaobai Sun. 2000. Algorithm 807: The SBR Toolbox—Software for Successive Band Reduction. *ACM Trans. Math. Software* 26, 4 (2000), 602–616. https://doi.org/10.1145/365723.365736

[7] L. Suzan Blackford, J. Choi, Andy Cleary, Eduardo F. D'Azevedo, James W. Demmel, Inderjit S. Dhillon, Jack J. Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, Ken Stanley, David W. Walker, and R. Clint Whaley. 1997. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia.

[8] Qinglei Cao, Sameh Abdulah, Rabab Alomairy, Yu Pei, Pratik Nag, George Bosilca, Jack Dongarra, Marc G. Genton, David E. Keyes, Hatem Ltaief, and Ying Sun. 2022. Reshaping Geostatistical Modeling and Prediction for Extreme-Scale Environmental Applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'22)*. IEEE Press, Dallas, Texas, Article 2, 12 pages.

[9] Lars Eldén. 2007. *Matrix Methods in Data Mining and Pattern Recognition*. Society for Industrial and Applied Mathematics. x + 224 pages.

[10] Aniello Esposito, David E. Keyes, Hatem Ltaief, and Dalal Sukkari. 2018. Performance Impact of Rank-Reordering on Advanced Polar Decomposition Algorithms. In *Cray Users' Group Conference*. http://hdl.handle.net/10754/628026

[11] Massimiliano Fasi and Nicholas J. Higham. 2021. Generating Extreme-Scale Matrices With Specified Singular Values or Condition Number. *SIAM Journal on Scientific Computing* 43, 1 (2021), A663–A684. https://doi.org/10.1137/20M1327938 arXiv:https://doi.org/10.1137/20M1327938

[12] Jerome A. Goldstein and Mel Levy. 1991. Linear Algebra and Quantum Chemistry. *Amer. Math. Monthly* 98, 10 (Oct. 1991), 710–718. https://doi.org/10.2307/2324422

[13] Gene H. Golub and C. Reinsch. 1970. Singular Value Decomposition and Least Squares Solutions. *Numerische Mathematik* 14 (1970), 403–420.

[14] Gene H. Golub and Charles F. Van Loan. 2012. *Matrix Computations* (4th ed.). The Johns Hopkins University Press.

[15] Ming Gu and Stanley C. Eisenstat. 1996. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing* 17, 4 (1996), 848–869.

[16] Wolfgang Hackbusch. 2015. *Hierarchical Matrices: Algorithms and Analysis*. Vol. 49. Springer. Springer Series in Computational Mathematics.

[17] Azzam Haidar, Hatem Ltaief, and Jack Dongarra. 2011. Parallel Reduction to Condensed Forms for Symmetric Eigenvalue Problems Using Aggregated Fine-grained And Memory-aware Kernels. In *Proceedings of SC'11 Conference on High Performance Computing Networking, Storage and Analysis*. ACM SIGARCH/IEEE Computer Society, Seattle, WA, USA, 8.

[18] Azzam Haidar, Hatem Ltaief, and Jack Dongarra. 2012. Toward a High Performance Tile Divide and Conquer Algorithm for the Dense Symmetric Eigenvalue

Problem. *SIAM Journal on Scientific Computing* 34, 6 (2012), 249–274.

[19] Azzam Haidar, Stanimire Tomov, Jack Dongarra, Raffaele Solcá, and Thomas Schulthess. 2014. A Novel Hybrid CPU-GPU Generalized Eigensolver for Electronic Structure Calculations Based on Fine-Grained Memory Aware Tasks. *The International Journal of High Performance Computing Applications* 28, 2 (2014), 196–209. https://doi.org/10.1177/1094342013502097

[20] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.

[21] Nicholas J. Higham. 2008. *Functions of Matrices: Theory and Computation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. xx+425 pages.

[22] Bruno Lang. 1999. Efficient Eigenvalue and Singular Value Computations On Shared Memory Machines. *Parallel Comput.* 25, 7 (1999), 845–860.

[23] Hatem Ltaief, Piotr Luszczek, and Jack Dongarra. 2012. Enhancing Parallelism of Tile Bidiagonal Transformation on Multicore Architectures Using Tree Reduction. In *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Waśniewski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 661–670.

[24] Hatem Ltaief, Piotr Luszczek, and Jack Dongarra. 2012. High Performance Bidiagonal Reduction using Tile Algorithms on Homogeneous Multicore Architectures. *ACM Trans. Math. Software* 39, 3 (2012), 1–22.

[25] Hatem Ltaief, Piotr Luszczek, Azzam Haidar, and Jack Dongarra. 2011. Solving the Generalized Symmetric Eigenvalue Problem using Tile Algorithms on Multicore Architectures. In *PARCO (Advances in Parallel Computing, Vol. 22)*, Koen De Bosschere, Erik H. D'Hollander, Gerhard R. Joubert, David A. Padua, Frans J. Peters, and Mark Sawyer (Eds.). IOS Press, 397–404. http://dx.doi.org/10.3233/978-1-61499-041-3-397

[26] Hatem Ltaief, Dalal Sukkari, Aniello Esposito, Yuji Nakatsukasa, and David Keyes. 2019. Massively Parallel Polar Decomposition on Distributed-Memory Systems. *ACM Transactions on Parallel Computing* 6, 1, Article 4 (June 2019), 15 pages. https://doi.org/10.1145/3328723

[27] Hatem Ltaief, Dalal Sukkari, Oliver Guyon, and David Keyes. 2018. Extreme Computing for Extreme Adaptive Optics: The Key to Finding Life Outside Our Solar System. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (Basel, Switzerland) *(PASC'18)*. ACM, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/3218176.3218225

[28] Piotr Luszczek, Hatem Ltaief, and Jack Dongarra. 2011. Two-Stage Tridiagonal Reduction for Dense Symmetric Matrices using Tile Algorithms on Multicore Architectures. In *2011 IEEE International Parallel & Distributed Processing Symposium*. ACM, Anchorage, AK USA, 944–955.

[29] Osni Marques, James Demmel, and Paulo B. Vasconcelos. 2020. Bidiagonal SVD Computation via an Associated Tridiagonal Eigenproblem. *ACM Trans. Math. Software* 46, 2, Article 14 (May 2020), 25 pages. https://doi.org/10.1145/3361746

[30] Yuji Nakatsukasa. 2020. Fast and stable randomized low-rank matrix approximation. *arXiv:2009.11392* (2020).

[31] Yuji Nakatsukasa, Zhaojun Bai, and François Gygi. 2010. Optimizing Halley's Iteration for Computing the Matrix Polar Decomposition. *SIAM J. Matrix Anal. Appl.* 31, 5 (2010), 2700–2720. https://doi.org/10.1137/090774999 arXiv:https://doi.org/10.1137/090774999

[32] Yuji Nakatsukasa and Roland W. Freund. 2016. Computing Fundamental Matrix Decompositions Accurately via the Matrix Sign Function in Two Iterations: The Power of Zolotarev's Functions. *SIAM Rev.* 58, 3 (2016), 461–493. http://dx.doi.org/10.1137/140990334

[33] Yuji Nakatsukasa and Nicholas J. Higham. 2013. Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD. *SIAM Journal on Scientific Computing* 35, 3 (2013), A1325–A1349. https://doi.org/10.1137/120876605

[34] Ivan V. Oseledets and Eugene E. Tyrtyshnikov. 2009. Breaking the Curse of Dimensionality, Or How to Use SVD in Many Dimensions. *SIAM Journal on Scientific Computing* 31, 5 (Oct. 2009), 3744–3759. https://doi.org/10.1137/090748330

[35] Robert Schreiber and Beresford Parlett. 1988. Block Reflectors: Theory and Computation. *SIAM J. Numer. Anal.* 25, 1 (1988), 189–205. https://doi.org/10.1137/0725014

[36] Rémi Soummer, Laurent Pueyo, and James Larkin. 2012. Detection and characterization of exoplanets and disks using projections on Karhunen-Loève eigenimages. *The Astrophysical Journal Letters* 755, 2 (2012), L28.

[37] Dalal Sukkari, Hatem Ltaief, Aniello Esposito, and David Keyes. 2019. A QDWH-Based SVD Software Framework on Distributed-Memory Manycore Systems. *ACM Trans. Math. Software* 45, 2, Article 18 (April 2019), 21 pages. https://doi.org/10.1145/3309548

[38] Dalal Sukkari, Hatem Ltaief, Mathieu Faverge, and David Keyes. 2017. Asynchronous Task-Based Polar Decomposition on Single Node Manycore Architectures. *IEEE Transactions on Parallel and Distributed Systems* PP, 99 (2017), 1–1. https://doi.org/10.1109/TPDS.2017.2755655

[39] Dalal Sukkari, Hatem Ltaief, and David E. Keyes. 2016. A High Performance QDWH-SVD Solver Using Hardware Accelerators. *ACM Trans. Math. Software* 43, 1 (2016), 6:1–6:25. http://doi.acm.org/10.1145/2894747

[40] Dalal Sukkari, Hatem Ltaief, and David E. Keyes. 2016. High Performance Polar Decomposition on Distributed Memory Systems. In *Euro-Par 2016: Parallel Processing - 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9833)*, Pierre-François Dutot and Denis Trystram (Eds.). Springer, 605–616. http://dx.doi.org/10.1007/978-3-319-43659-3

[41] Lloyd N. Trefethen and David Bau. 1997. *Numerical Linear Algebra.* SIAM, Philadelphia, PA. http://www.siam.org/books/OT50/Index.htm

Received 7 April 2023; revised 17 June 2023; accepted 30 June 2023

Appendix: Artifact Description/Artifact Evaluation

ARTIFACT IDENTIFICATION

(1) We introduce a new singular value decomposition (SVD) solver based on the QR-based Dynamically Weighted Halley (QDWH) algo- rithm for computing the partial spectrum SVD (QDWHpartial-SVD) problems. The QDWHpartial-SVD algorithm efficiently computes a fraction (say 1-20singular values/vectors.
(2) We develop a high-performance implemen- tation of QDWHpartial-SVD 1 on distributed-memory manycore systems and demonstrate its numerical robustness.
(3) We perform a benchmarking campaign against counterparts from the state-of-the- art numerical libraries across various matrix sizes using up to 36K MPI processes.
(4) Experimental results show performance speedups for QDWHpartial-SVD up to 6X and 2X against vendor-optimized PDGESVD from ScaLAPACK and KSVD on a Cray XC40 system using 1152 nodes based on two-socket 16-core Intel Haswell CPU, respectively.
(5) We also port our QDWHpartial-SVD software library to a system composed of 256 nodes with two-socket 64-Core AMD EPYC Milan CPU and achieve performance speedup up to 4X compared to vendor-optimized PDGESVD from ScaLAPACK.
(6) We also compare energy consumption for the two algorithms and demonstrate how QDWHpartial-SVD can further outperform PDGESVD in that regard by performing fewer memory-bound operations.

REPRODUCIBILITY OF EXPERIMENTS

1 ARTIFACT DETAILS

(1) We run our experiments on the Shaheen-2 Cray XC40 system with the Cray Aries network interconnect, which implements a Dragon- fly network topology. The system has 6174 compute nodes, each with two-socket 16-core Intel Haswell running at 2.3GHz and 128GB of DDR3 main memory.
(2) We run also on a second system with 256 nodes based on two-socket 64-core AMD EPYC 7763 Milan CPUs, clocked at 2.45 GHz with 512 GB DDR4-3200
(3) We use the Intel compiler and rely on the vendor-optimized ScaLAPACK implementation from the high performance Cray LibSci numerical library, which depends on the MPI programming model for inter- node communications. We refer to Intel Haswell and AMD Milan for the former and latter systems, respectively.

2 ARTIFACT

The artifact can be downloaded at this link:

https://doi.org/10.5281/zenodo.7834248.

The artifact contains folder for QDWHpartial-SVD implementation as follows:

install.sh
run.sh
qdwhpartial-svd/

The installation scripts can be run as follows:

cd qdwhpartial-svd
source ../install.sh

Note that the installation script must be sourced since it load system modules and change environment variables.

The following commands are used to reproduce the QDWHpartial-SVD results on Shaheen:

cd qdwhpartial-svd
bsub ../run.sh