High Performance Batch SVD using GPUs

Ahmad Abdelfattah
ahmad@icl.utk.edu
University of Tennessee
Knoxville, Tennessee, USA

CCS Concepts

• **Mathematics of computing → Mathematical software performance**; **Solvers**.

Keywords

Singular Value Decomposition (SVD), GPU Computing, One-sided Jacobi Algorithm

ACM Reference Format:
Ahmad Abdelfattah. 2025. High Performance Batch SVD using GPUs. In *SC'25: The International Conference for High Performance Computing, Networking, Storage, and Analysis, November 16-21, 2025, St. Louis, MO*. ACM, New York, NY, USA, 1 page. https://doi.org/XXXXXXX.XXXXXXX

1 Poster Summary

We consider the problem of computing the singular value decomposition (SVD) of many relatively-small matrices using GPUs. This is an essential component in various scientific applications, including computational chemistry, low-rank approximations, hierarchical matrices, and others. Standard algorithms in the LAPACK library often rely on reducing the matrix to a bi-diagonal form, followed by an SVD solver, such as the implicit QR algorithm or the Divide-and-Conquer algorithm, which iteratively reduce the bi-diagonal structure to a diagonal form, thus revealing the singular values. However, these algorithms are not trivial to adapt for optimizations on heterogeneous systems with GPUs. Instead, we consider the one-sided Jacobi algorithm, which operates directly on the original matrix. We also focus on the problem of solving several relatively small matrices at the same time concurrently (e.g., a batch solver). For an SVD problem $A_{m\times n} = U\Sigma V^T$, the one-sided Jacobi algorithm implicitly solves the Hermitian eigenvalue problem $G = A^TA = V^TDV$. The V matrix is never computed explicitly, but is rather computed as sequences of *Jacobi rotations*. Post-multiplying A with these rotations reveals the product $\bar{A} = U\Sigma$, for which the singular values Σ can be computed using the column-wise 2-norm of $\bar{A}$.

We use two different designs of the algorithm depending on the matrix size. If the matrix is small enough to fit in the shared memory of the GPU, we use a parallel-scalar version of the algorithm, which runs entirely in the shared memory of the GPU, iteratively annihilating off-diagonal entries of the G matrix using parallel independent Jacobi rotations. The second design uses a parallel-blocked Jacobi algorithm, which tries to orthogonalize independent pairs of block-columns of A. The algorithm can be summarized in three stages. **First**, for every pair (i, j) of block columns, we compute the corresponding Gram matrix $G_{ij} = [A_iA_j]^T \times [A_iA_j]$. **Second**, a Hermitian eigen-solver is used to factorize $G_{ij} = V_{ij}^T D_{ij} V_{ij}$. **Third**, the V_{ij} matrix is used to update the corresponding block-columns of A as well as the matrix of right singular vectors. The first and third stages of the algorithm can directly benefit from the optimized level-3 BLAS routines in the vendor libraries. However, we also show that customized kernels used for these computational stages offer significant performance gains against standard kernels. We also propose that customized kernels accept a "mask" input that is used to cancel a certain operation in the batch if the corresponding SVD problem has already converged.

There are four competitive solvers against which we compare our work.

(1) rocSOLVER: is developed by AMD. Its batch SVD solver is based on a bi-diagonalization stage followed by an iterative SVD solver. It supports all standard precisions (single, double, single-complex, double-complex). Our tests show that they support arbitrary dimensions satisfying $m \geq n$.
(2) cuSOLVER: is developed by NVIDIA. Its batch SVD solver is Jacobi-based, and supports all standard precisions. However, it does not support dimensions greater than 32.
(3) KBLAS: is an open-source library implementing a subset of BLAS and LAPACK algorithms. Its batch SVD solver is Jacobi-based, supports only real precisions, and requires that $m \leq 1024$.
(4) WcycleSVD: is an open source Jacobi-based SVD solver. It is implemented only in double precision and requires $m \leq 1024$. We point out that this solver sometimes fails the orthogonality check for the singular vectors on supported dimensions.

Our proposed batch SVD solver 1) supports all standard precisions (single, double, single-complex, and double-complex), 2) supports arbitrary dimensions (both $m \geq n$ and $m < n$), and 3) outperforms all the above solvers on a wide range of dimensions. On a GH200 GPU, our solver is up to 33.9× faster than cuSOLVER, up to 9× against KBLAS, and up to 8.8× against WcycleSVD. On the MI300A GPU, our solver is up to 514.7× against rocSOLVER. The proposed batch SVD solver is lined up for release in the widely-used MAGMA library (https://icl.utk.edu/magma/).

SC'25, St. Louis, MO

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX