# High Performance Batch SVD using GPUs

**Ahmad Abdelfattah**
UNIVERSITY OF TENNESSEE

## Abstract

We consider the problem of computing the singular value decomposition (SVD) on many relatively-small matrices using GPUs. This is an essential component in various scientific applications, including computational chemistry, low-rank approximations, and others. Our approach is based on the parallel one-sided Jacobi algorithm, which has a large degree of parallelism, and also heavily relies on compute-bound level-3 BLAS operations, such as matrix multiply. Our approach uses two design strategies. The first one targets very small matrices using a single GPU kernel for the entire SVD operation. The second design strategy uses a blocked version of the parallel Jacobi algorithm, which supports matrices of arbitrary dimensions. The proposed solution supports any matrix shape (square, tall-skinny, or short-wide), requires no limitations on the matrix dimensions, and delivers superior performance against state-of-the-art solutions. This work is set to be released in the MAGMA library.

## Algorithmic Background

We consider the one-sided Jacobi algorithm for solving the batch reduced SVD problem.

1) Implicitly solves $G = A^TA = V^TDV$ ($G$ is never fully explicitly computed)
2) Multiply $A \times V = U\Sigma$ ($V$ is obtained through a series of Jacobi rotations)
3) Each rotation annihilates one off-diagonal element in $G$ by solving a 2×2 symmetric eigenvalue problem
4) Several independent off-diagonal elements can be considered in parallel (parallel Jacobi)
5) We use a round-robin parallel ordering to generate independent off-diagonals
6) Block Jacobi: targets an off-diagonal block instead of one element at a time


**One-sided Jacobi Algorithm**

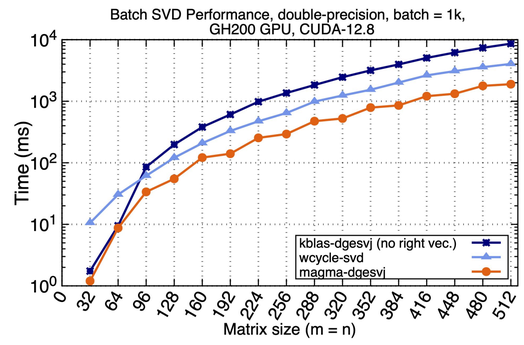
```
Function [U, Σ, V] = GESVJ(A_mxn)
    V = I
    while ( not converge ) {
        // begin Jacobi sweep
        for each pair (i, j), i < j {
            g_ii = A_i^T A_i
            g_jj = A_j^T A_j
            g_ij = A^T_i A_j
            if |g_ij| ≥ k.ϵ.sqrt(g_ii g_jj) {
                G_ij = [g_ii g_ij; g_ij g_jj]
                Solve G_ij = J^T D J
                A = A J
                V = V J
            }
        }
    }
    // Finalize values and vectors
    for i in 1:n {
        σi = ‖ai‖2
        Ui = ai / σi
    }
    // Sort Σ and reorder U and V
```

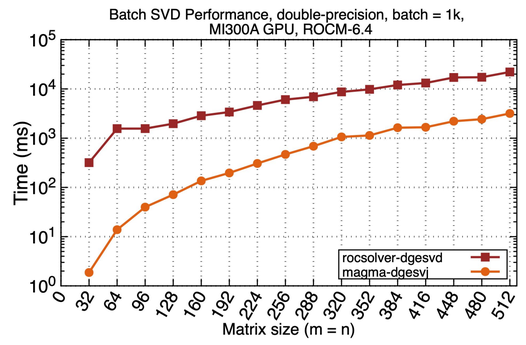
## Design Highlights

- Batch Hermitian eigensolver (HEEVJ):
  - Used to orthogonalize a batch of block-column pairs
  - Implements a two-sided Jacobi algorithm in shared memory
  - The value ***nb*** must be carefully tuned to respect the shared memory capacity
- Gram matrix computation
  - A baseline implementation require a batch Hermitian rank-k update (HERK)
  - Batch HERK is not supported in all backends: use batch GEMM instead
- Left and Right Vector Updates:
  - Baseline implementation uses batch GEMM, which dominates the execution time asymptotically (up to 67% in some cases)
  - A custom vector update kernel is developed, which delivers a **2.1*X*** speedup against the baseline updates (and more than 50% reduction in solution time, asymptotically)
- Masked Batch Operations:
  - A batch SVD solver may converge at different rates for different problems
  - The batch HEEVJ can be partially masked-off for converged problems
  - Similarly, the batch vector updates can be masked-off if the corresponding block-column pair(s) are already mutually orthogonal
  - Saves ~12% of the solution time, asymptotically

## Performance on relatively-large square problems

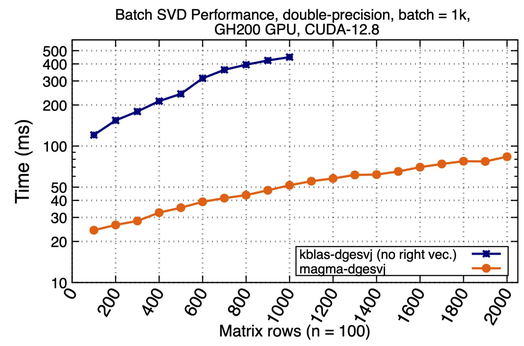
- Speedups:**1.7*X*** - **8.8*X*** against WcycleSVD, **1.1*X*** - **4.7*X*** against KBLAS, **7*X*** - **170.8*X*** against rocSOLVER
- WcycleSVD sometimes fails the orthogonality check for the left vectors

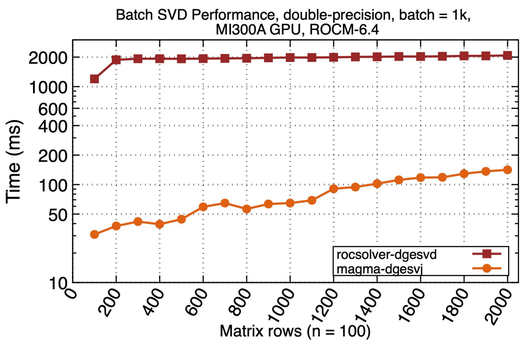



## Performance on Rectangular Shapes

- Speedups: **5*X*** - **9*X*** against KBLAS, **14.7*X*** - **49.6*X*** against rocSOLVER
- WcycleSVD did not produce accurate results in this test
- KBLAS has a limit on the number of rows, also uses a batch QR factorization as a pre-processing step




## Accuracy Requirements & Other GPU Solvers

All the following errors should be at most $\mathcal{O}(10^{-15})$

- $e_1 = \|A - U\Sigma V^T\| / (\|A\| \times max(M,N))$, $e_2 = \|I - U^TU\| / (M)$, $e_3 = \|I - VV^T\| / (N)$

| | cuSOLVER | rocSOLVER | KBLAS[1] | WcycleSVD[2] | MAGMA (this work) |
|---|---|---|---|---|---|
| **Algorithm** | Jacobi-based | Bidiagonalization + QR Algorithm | Jacobi-based | Jacobi-based | Jacobi-based |
| **Precisions** | FP32/FP64, real/complex | FP32/FP64, real/complex | FP32/FP64, real only | FP64, real only | FP32/FP64, real/complex |
| **Limitations on size** | M and N ≤ 32[1] | M ≥ N[1] | M ≥ N, M ≤ 1024[1] | M ≤ 1024[2] | None[1] |

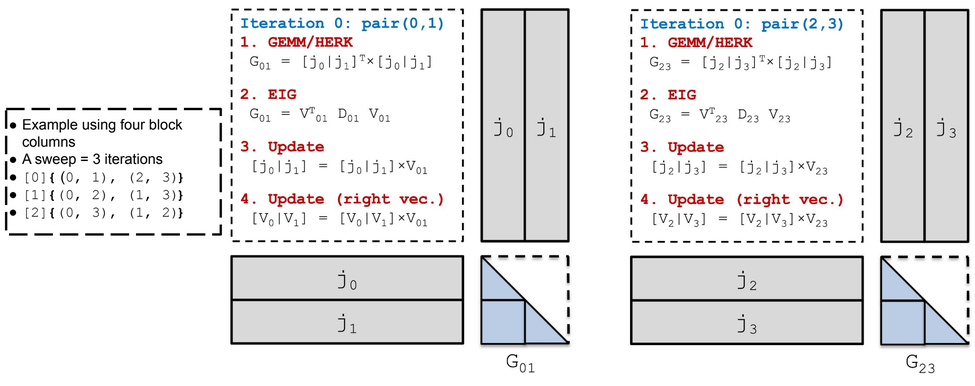
[1] All matrices must fit in GPU memory (along with any required workspace)
[2] Sometimes fails accuracy checks for supported dimensions

## Very Small Problems

- MAGMA uses a parallel scalar Jacobi algorithm that runs entirely in shared memory
- Speedups: **1.4*X*** - **2.6*X*** against KBLAS, **2.2*X*** - **33.9*X*** against cuSOLVER, **77*X*** - **514.7*X*** against rocSOLVER

## Parallel-blocked Jacobi Algorithm

- $A_{M \times N}$ is partitioned into block columns of width ***nb***
- Independent pairs of block-columns are orthogonalized




## Future Directions

- Explore performance optimizations for computing the Gram matrix
- Study the impact of the QR factorization on performance and convergence of the Jacobi method
- Extend the HEEVJ implementation to relax the constraints on the range of values for ***nb***

## ACKNOWLEDGEMENTS

This work was partially supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

This work used resources on the Frank cluster at the Performance Research Laboratory, University of Oregon.

This work used resources from the Experimental Computing Laboratory at Oak Ridge National Laboratory.

## REFERENCES

1. Wajih Halim Boukaram, George Turkiyyah, Hatem Ltaief, David E. Keyes, "Batched QR and SVD algorithms on GPUs with applications in hierarchical matrix compression," Parallel Computing, Volume 74, 2018, Pages 19-33, ISSN 0167-8191, https://doi.org/10.1016/j.parco.2017.09.001
2. Junmin Xiao, Yunfei Pang, Qing Xue, Chaoyang Shui, Ke Meng, Hui Ma, "W-Cycle SVD: A Multilevel Algorithm for Batched SVD on GPUs," SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, 2022, pp. 1-16, https://ieeexplore.ieee.org/document/10046109