POSTER: A W-cycle Algorithm for Efficient Batched SVD on GPUs

Junmin Xiao, Qing Xue, Hui Ma, Xiaoyang Zhang, and Guangming Tan
State Key Laboratory of Computer Architecture, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China
xiaojunmin@ict.ac.cn xueqing18s@ict.ac.cn mahui@ncic.ac.cn zhangxiaoyang@ncic.ac.cn tgm@ict.ac.cn

Abstract

As a fundamental factorization operation, the singular value decomposition (SVD) plays a paramount role in a broad range of domains such as scientific computing and machine learning. Due to its computational bottleneck of factorization for small matrices in real-world applications, many GPU-accelerated batched SVD algorithms have been investigated recently. However, these algorithms failed to achieve a balance between data locality and parallelism because their workflows depend on the size of each matrix. In this work, we propose a matrix-size-independent W-cycle algorithm to accelerate the batched one-side Jacobi SVD on GPUs, which successfully strikes the balance between data locality and parallelism. The experimental evaluation demonstrates that the proposed algorithm achieves 4.5× performance speedup on average over the state-of-the-art cuSOLVER.

CCS Concepts: • **Computing methodologies → Parallel algorithms**; *Shared memory algorithms.*

1 Introduction

The singular value decomposition (SVD) is a basic matrix factorization, which is the generalization of eigenvalue decomposition of a positive semi-definite matrix [2]. It is one of the most widely used high-performance kernels in diverse domains such as scientific computing and machine learning.

The recent decade has witnessed the development of the batched SVD on GPUs [1]. To the best of our knowledge, the GPU-CPU hybrid algorithm for batched SVD was first proposed for the detection of quiet targets in underwater acoustic array signal processing. The algorithm focused on the pair generation for the Jacobi algorithm and handled the bi-diagonalization phase of Gram matrices on CPU. Afterward, the batched SVD on GPU was studied to generate rank 1 matrices for approximating 2D filters in convolutional neural networks, which concentrated on efficiently obtaining the largest singular values and the corresponding singular vectors of many small matrices with sizes of less than 15×15. And then, for the image mosaic assemble application, conventional methods for batched SVD on GPUs were developed, where each thread within a warp was applied to compute the SVD of a single matrix. In recent years, the fine-grained analysis on batched SVD suggested that the matrices of different sizes need different SVD algorithm designs to achieve high performance on GPUs [1], and the related works were primarily matrix-size-dependent, which hardly achieves the tradeoff between data locality and parallelism.

To address the problems mentioned above, we investigate the column block orthogonalization deeply and re-design the workflow. Based on the fine-grain analysis, a W-cycle algorithm is proposed for accelerating the batched one-sided Jacobi SVD on GPUs.

Our contributions can be summarized as follows:

- Propose a W-cycle algorithm, which provides a uniform workflow for a batch of SVDs to strike the balance between data locality and parallelism.
- Evaluate and analyze the performance of the W-cycle algorithm, which proves that the proposed algorithm supports the batched SVD on GPUs well.

2 W-cycle Algorithm

Figure 1 shows a specific example to illustrate the proposed W-cycle algorithm, which supports a uniform workflow for batched SVD with different matrix sizes. Assume that there are 4 matrices A^k (where $k = 1, 2, 3, 4$) with sizes of 32×32, 48×48, 32×128, and 128×128. The W-cycle algorithm constructs a three-level workflow for their SVDs with the column widths $w_1 = 32$ and $w_2 = 16$ for partitioning each matrix into small sub-matrices at Levels 1 and 2 respectively. Assume that the shared memory for each thread block can store $2,304$ double-precision elements ($2,304 = 48 * 48$).

Step 0: The four matrices are placed at Level 0. Since A^1 and A^2 can be entirely stored in the shared memory (SM), a batched kernel would parallelly accomplish the SVDs of A^1 and A^2 in the shared memory. By using $w_1 = 32$, A^3 is divided into multiple sub-matrices along the column direction. Denote the i-th and j-th sub-matrices as $A_i^{(1,3)} \in \mathbb{R}^{32\times32}$ and

PPoPP '22, April 2–6, 2022, Seoul, Republic of Korea
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9204-4/22/04.
https://doi.org/10.1145/3503221.3508443

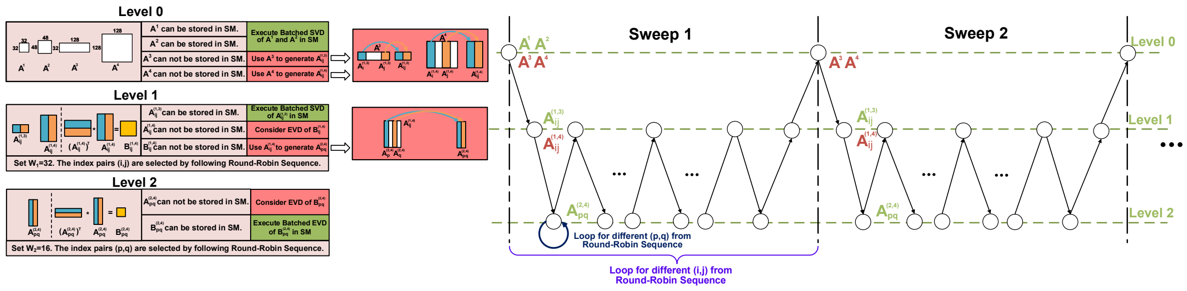
Figure 1. W-cycle Algorithm for Batched SVD.

$A_j^{(1,3)} \in \mathbb{R}^{32\times32}$, which are joined together to form $A_{ij}^{(1,3)} = [A_i^{(1,3)}, A_j^{(1,3)}]$. For A^4, $A_{ij}^{(1,4)}$ is generated the same way. The sizes of $A_{ij}^{(1,3)}$ and $A_{ij}^{(1,4)}$ are 32×64 and 128×64 respectively.

Step 1: The SVDs of $A_{ij}^{(1,3)}$ and $A_{ij}^{(1,4)}$ are considered at Level 1. Since $A_{ij}^{(1,3)}$ is small enough, its SVD can be executed in shared memory. Set $B_{ij}^{(1,4)} = (A_{ij}^{(1,4)})^T A_{ij}^{(1,4)}$ with size of 64×64. Since $A_{ij}^{(1,4)}$ and $B_{ij}^{(1,4)}$ are both larger than the shared memory size, we use $w_2 = 16$ to further partition $A_{ij}^{(1,4)}$ into smaller sub-matrices $A_h^{(2,4)} \in \mathbb{R}^{128\times16}$ along the column direction. Further, $A_{pq}^{(2,4)} = [A_p^{(2,4)}, A_q^{(2,4)}] \in \mathbb{R}^{128\times32}$ is generated, and its SVD is considered at Level 2.

Step 2: At Level 2, $B_{pq}^{(2,4)} = (A_{pq}^{(2,4)})^T A_{pq}^{(2,4)}$ is a 32×32 Gram matrix which can be entirely stored in shared memory. A batched EVD kernel would accomplish the EVDs of matrices $B_{pq}^{(2,4)}$ in shared memory parallelly for different p and q. After the index pairs (p, q) go through all of the available choices, the workflow goes back to Level 1.

Step 3: At Level 1, the index pairs (i, j) are changed. Steps 1 and 2 are repeated until all the available (i, j) are chosen exactly once. After that, the workflow returns to Level 0.

Step 4: Repeat the process above from Step 0 to Step 3 for A^3 and A^4 in a loop. If all the column block sub-matrices of A^k are orthogonal with each other, the SVD of A^k is completed and A^k jumps out the loop ($k = 3$ or 4). When the loop is finished, the W-cycle algorithm ends.

3 Evaluation

Figure 2 shows the performance comparison of the W-cycle algorithm with the batched SVD kernel in cuSOVLER on NVIDIA V100 GPU. The W-cycle algorithm achieves 4.5× performance speedup on average. The observation from the results is as follows.

Firstly, we find that when m and n are fixed, the performance benefit of the W-cycle algorithm increases as the batch size enlarges. Since both rows and columns are not larger than 32, every matrix can be stored entirely in shared memory. In this case, there are 2 levels in the W-cycle algorithm, and each SVD is executed by a batched kernel function, which maximizes the data reuse in shared memory.

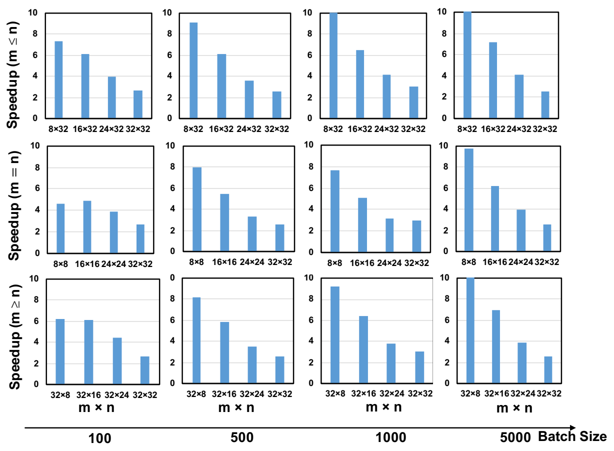

Figure 2. Performance Improvement over cuSOVLER.

Secondly, when the batch size is fixed, the performance benefit of the W-cycle algorithm increases with the decrease of matrix size. The reason is that our design could achieve higher parallelism for SVDs of smaller matrices.

Thirdly, compared with the batched SVD with sizes of $m \geq n$, the W-cycle algorithm achieves higher speedup for SVDs with $m < n$. Because, for any matrix with $m < n$, its transpose is used to generate SVD, which could decrease the number of iterations.

Acknowledgments

The authors would like to thank all anonymous referees for their valuable comments and helpful suggestions. This work is supported by the National Natural Science Foundation of China, under grant No. (62172391, 61802369, 61972377, 62032023).

References

[1] Wajih Halim Boukaram, George Turkiyyah, Hatem Ltaief, and David E. Keyes. 2018. Batched QR and SVD algorithms on GPUs with applications in hierarchical matrix compression. *Parallel Comput.* 74 (2018), 19–33. https://doi.org/10.1016/j.parco.2017.09.001 Parallel Matrix Algorithms and Applications (PMAA'16).

[2] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. 2018. The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale. *SIAM Rev.* 60, 4 (2018), 808–865. https://doi.org/10.1137/17M1117732