Accelerating the Higher Order Singular Value Decomposition Algorithm for Big Data with GPUs

Poster summary

Yuhsiang M. Tsai
Institute of Applied Mathematical Sciences, National Taiwan University
yhmtsai@ntu.edu.tw

Jeewhan Choi (advisor)
IBM Research
jwchoi@us.ibm.com

Xing Liu (advisor)
IBM Research
xliu@us.ibm.com

Weichung Wang (advisor)
Institute of Applied Mathematical Sciences, National Taiwan University
wwang@ntu.edu.tw

1 SUMMARY

We tackle the problem of implementing efficient large-scale dense tensor decomposition on GPUs. We implement Higher-Order Singular Value Decomposition (HOSVD) [3], and Sequential-Truncated Higher-Order Singular Value Decomposition (STHOSVD) [4] using four different methods: Householder QR, modified Block QR [1], Tall Skinny QR (TSQR) [2], and Gram method. Our efforts were motivated by the fact that to apply Singular Value Decomposition (SVD) from cuSolver library, we need the entire matricized tensor in the GPU memory. This severely limits the size of the tensor that we can process, and led us to explore alternative method.

For the HOSVD method, we first use QR factorization to reduce the problem size, while streaming the data into the GPU to handle tensors of any size. We then apply SVD on the matrix produced by QR methods whose dimensions are typically much smaller than the original problem.

Householder-based QR method limits us to streaming the data vector-by-vector, and while this mitigates the tensor size limitation, it suffers from low throughput as result of BLAS-2 operations (computation) and low data granularity (bandwidth). To improve the efficiency of data transfer and computation, we implement modified Block QR that combines several Householder factors into two *matrices*, transforming the computation to more efficient BLAS-3 operations. However, when the matrix is extremely tall and skinny there is still not enough work to saturate the compute units. We then use TSQR instead, which partitions the matrix along the row and processes multiple blocks simultaneously. Both modified Block QR and TSQR methods use BLAS-3 operations which better utilize the GPUs, and among the three QR methods, we found TSQR to be fastest.

The Gram method transforms the SVD problem into an eigenvalue problem by computing the Gram matrix of the matricized tensor. It offers one advantage over QR methods: the Gram matrix calculation (i.e., DGEMM) can easily be parallelized over multiple GPUs while sustaining high throughput. The STHOSVD method provides an advantage over HOSVD method: the work required for calculating the singular/eigenvectors is greatly reduced by moving the tensor-times-matrix (TTM) to the inner loop. We implement and compare block QR and Gram methods for STHOSVD.

Figure 1: performance of all methods

The performance of all methods is summarized in Figure 1.
Testing environment:
CPU: full 24 threads of 2x Intel(R) Xeon(R) CPU E5-2670 v3
GPU: 1x or 2x NVIDIA Tesla K40c.

HOSVD with direct cuSolver SVD fails when the problem size exceeds $2048 \times 2048 \times 2048$ as it requires 64 GB of memory, much greater than any GPU device memory size. Householder QR method allows us to solve such large data sets, but achieves lower performance than implementations based on MKL and cuSolver for data sets that fit in GPU memory, due to the aforementioned issues. However, our improved implementation based on block QR and TSQR all achieve much greater performance, up to 8.57x speedup.

When comparing STHOSVD and HOVSD using block QR for data sets $2048 \longrightarrow R$, we see that the execution time remains constant with changing R for HOSVD, whereas it diminishes with R for STHOSVD. STHOSVD with block QR has good performance, up to 2.6x speedup over HOSVD with block QR.

We also implement those algorithms on multiple GPUs. For Block QR, we implement Householder QR and Block QR block-wise. The information also updates block-wise. These methods update simultaneously on multiple GPUs. We make one GPU update fewer blocks than the other GPUs and solve QR factorization of the next block when other GPUs still update the remaining blocks. With this schedule, the next round updating step does not need to wait for the QR factorization. For TSQR, we split a matrix into several tall and skinny blocks by the number of GPUs. We use TSQR to solve each block, so it is an independent process. And then we combine the results together in a single matrix and apply TSQR method on this matrix. For Gram, we split the matrix into several small blocks. For each block, we only need to calculate the product of its transpose and itself. The operations in each block are independent, so we can assign those works to multiple GPUs equally.

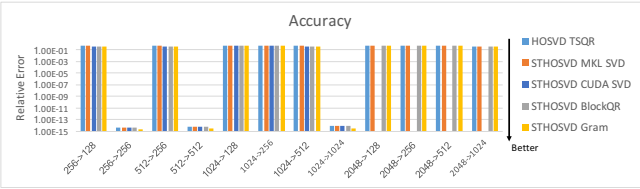

Figure 2: Accuracy

Overall, STHOSVD with Gram performs the best, up to 163.21x speedup over HOSVD with direct CUDA SVD, and 61.52x speedup over STHOSVD with direct CUDA SVD. While the Gram method shows comparable accuracy on our data sets (Figure 2), QR method is proved to be more accurate when the conditional number of the tensor is large, as the Gram method has the inherent disadvantage of losing accuracy.

REFERENCES

[1] Dan Campbell Andrew Kerr and Mark Richards. 2009. QR decomposition on GPUs. In *GPGPU'09 : Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. ACM, 71–78.

[2] Mark Hoemmen James Demmel, Laura Grigori and Julien Langou. 2012. Communication-optimal Parallel and Sequential QR and LU Factorizations. *SIAM Journal on Scientific Computing* 34, 1 (2012), A206–A239.

[3] Tamara G. Kolda and Brett W. Bader. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (2009), 455–500.

[4] Raf Vandebril Nick Vannieuwenhoven and Karl Meerbergen. 2012. A New Truncation Strategy for the Higher-Order Singular Value Decomposition. *SIAM Journal on Scientific Computing* 34, 2 (2012), A1027–A1052.