jh200023-NAHI

Hierarchical Low-Rank Approximation Methods on Distributed Memory and GPUs

Rio Yokota (Tokyo Institute of Technology)

Abstract

Hierarchical low-rank approximation of dense matrices can reduce the complexity of matrix multiplication and factorization from $\mathcal{O}(N^3)$ to $\mathcal{O}(N)$ while trading accuracy for speed. Modern processors are equipped with low-precision arithmetic, which yield much higher throughput if low accuracy can be tolerated. On such processors, performing exact $O(N^3)$ dense linear algebra operations through the use of LAPACK and BLAS libraries is a waste of Flops. Our goal is to replace these exact dense linear algebra libraries with our hierarchical low-rank approximation library. The functions needed for this are the capability to perform LU and QR factorization on GPUs and distributed memory. For FY 2020 we focused on the following critical steps to achieving this goal; Use of uniform basis as a nested basis block low-rank (BLR) matrix, QR factorization on TensorCores with refinement, Eigenvalue computation based on BLR-QR, GPU implementation of lattice H-matrix.

1 Basic Information

1.1 Collaborating JHPCN Centers

The University of Tokyo
Information Technology Center

Tokyo Institute of Technology
Global Scientific Information and Computing Center

Hokkaido University
Information Initiative Center

Kyoto University
Academic Center for Computing and Media Studies

Nagoya University
Information Technology Center

1.2 Research Areas

- Very large-scale numerical computation

1.3 Roles of Project Members

Rio Yokota (Tokyo Institute of Technology) Low-rank approximation using FMM and its GPU-MPI implementation

Ichitaro Yamazaki (Sandia National Laboratories) Development of distributed memory runtime ParSEC, and blocked BLAS library for GPU

Akihiro Ida (University of Tokyo) Feature extension of hybrid MPI/OpenMP $\mathcal{H}$-matrix code HACApK, and its integration with ParSEC and block MAGMA

Takeshi Iwashita (Hokkaido University) Application of HACApK to boundary integral solvers for electromagnetics, and optimization of $\mathcal{H}$-matrix-vector product

Takeshi Fukaya (Hokkaido University) Development of QR decomposition on TensorCores for low-rank approximation

Satoshi Oshima (Nagoya University) GPU implementation of HACApK and integration with MAGMA

Kengo Nakajima (University of Tokyo) Extend capability of HACApK within the ppOpen-HPC framework

Toshihiro Hanawa (University of Tokyo) Support for code optimization using FPGA, MPI, GPU

Tetsuya Hoshino (University of Tokyo) Optimization of batched operations on GPU

Tasuku Hiraishi (Kyoto University) Dynamic load-balancing of HACApK

Kazuki Osawa (Tokyo Institute of Technology) Generation of Hessian, Fisher, Covariance matrices

Hiroki Naganuma (Tokyo Institute of Technology) Experiments for deep learning optimization

Hiroyuki Ootomo (Tokyo Institute of Technology) Optimization of TensorCore implementation

Shun Iwase (Tokyo Institute of Technology) Visualization of results using 3-D rendering

Sameer Deshmukh (Tokyo Institute of Technology) Optimization of batched low-rank kernels on CPU

Peter Spalthoff (Tokyo Institute of Technology) Development of nested-bases, main developer of HiCMA code

Muhammad Ridwan Apriansyah Budikafa (Tokyo Institute of Technology) QR decomposition using BLR structure

Qianxiang Ma (Tokyo Institute of Technology) GPU implementation of H^2-matrix

Yuichiro Ueno (Tokyo Institute of Technology) Development of hierarchical AllReduce over NCCL

Hikaru Nakata (Tokyo Institute of Technology) Application to continual learning

Linsho Kaku (Tokyo Institute of Technology) Application to few-shot learning

Thomas Spendlhofer (Tokyo Institute of Technology) Development of novel low-rank compression schemes

Mikiya Shibuya (Tokyo Institute of Technology) Visualization of results using 3-D rendering

Takahiro Shohata (Tokyo Institute of Technology) Application to stochastic weight averaging

Hana Hoshino (Tokyo Institute of Technology) Generation of Hessian, Fisher, Covariance matrices

Aoyu Li (Tokyo Institute of Technology) Application to non-uniform sampling methods

Takumi Ito (Tokyo Institute of Technology) Generation of Hessian, Fisher, Covariance matrices

Sora Takashima (Tokyo Institute of Technology) Application to eigenvalue based generalization metrics

Xinyu Zhang (Tokyo Institute of Technology) Application to large language models

2 Purpose and significance of Research

The purpose of this research is to develop a scalable and highly optimized open source library for hierarchical low-rank approximation of dense matrices. Such large dense ma-

trices naturally appear in electromagnetic, seismic, quantum, and fluid simulations, in scientific computing. Large dense matrices also appear in machine learning, where the Hessian, Fisher, Covariance, and Gram matrices play an important role in determining the properties of optimization and generalization of deep neural networks. Unlike their dense counterparts which require $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory, $\mathcal{H}$-matrices can perform matrix multiplication and factorization in $\mathcal{O}(N)$ time and $\mathcal{O}(N)$ memory, have controllable arithmetic intensity, have asynchronous communication, and can exploit deep memory hierarchy.

During the previous JHPCN project we have extended the $\mathcal{H}$-matrix code to perform not only matrix-vector multiplications, but also matrix-matrix multiplication, LU factorization, and QR factorization. We have also extended the parallelization to support not only OpenMP and MPI, but also batched GPU kernels and task-based parallelization. We experimented with runtime systems such as OmpSs, StarPU, but found that the overhead was too large so we designed our own light-weight task scheduler. Another achievement is the lattice $\mathcal{H}$-matrix method, which combines the scalability of block-low-rank methods with the favorable arithmetic complexity of $\mathcal{H}$-matrices. The present JHPCN project extends our previous work in the direction of better scalability, higher GPU utilization, and better accuracy control. We also plan to experiment with matrices that arise in deep learning applications, though it is possible that such matrices may not have hierarchical low-rank structure.

3 Significance as JHPCN Joint Research Project

Hardware architecture is now moving towards low-precision arithmetic, backed by the increasing demand from the machine learning field. When such low-accuracy can be tolerated, exact dense linear algebra operations become unnecessary, and libraries such as BLAS and LAPACK, which are at the heart of HPC applications, can be replaced by hierarchical low-rank ($\mathcal{H}$-matrix) libraries that effectively do the same work in linear time. Due to the increasing diversity in hardware, it is becoming increasingly difficult to maintain a single codebase that achieves performance portability across all these different architectures. Our aim is to develop a $\mathcal{H}$-matrix library that can achieve performance portability across the different JHPCN centers. Therefore, collaboration with multiple JHPCN centers is essential.

4 Outline of Research Achievements up to FY2019

Up to FY2019 we have tackled various problems regarding hierarchical low-rank approximation and its parallel implementation. There are various derivatives of hierarchical low-rank approximation methods such as; BLR, HODLR, HSS, $\mathcal{H}$-matrix, and $\mathcal{H}^2$-matrix. We started from the most basic variant – BLR, which uses low-rank off-diagonal blocks, but not a hierarchical matrix. We started with the most basic operations such as matrix-vector and matrix-matrix multiplication. This was extended during FY2016 to LU factorization and implemented in OpenMP and MPI. In FY2017, we extended the matrix format to more complex HSS and $\mathcal{H}$-matrix structures, and extended the implementation to GPUs for the matrix-vector multiplication. We utilized batched MAGMA operations to process the matrix-vector multiplication efficiently on GPUs. In FY2018, we further extended the implementation of the LU factorization to multiple-GPUs using a hybrid MPI + OpenMP + CUDA code. In FY 2019 we extended the $\mathcal{H}$-matrix code to $\mathcal{H}^2$-matrix by using a nested basis. We also used a runtime for $\mathcal{H}$-LU on GPU, but found that such runtimes like StarPU and OmpSs incur too much overhead. For the inner kernels, we ported the QR decomposition to run on TensorCores. Finally, we implemented the QR decomposition using the BLR matrix.

5 Details of FY2020 Research Achievements

The four main goals for the fiscal year 2020 were

1. Use of uniform basis as a nested basis block low-rank (BLR) matrix (FY2020 1Q)
2. QR factorization on TensorCores with refinement (FY2020 2Q)
3. Eigenvalue computation based on BLR-QR (FY2020 3Q)
4. GPU implementation of lattice H-matrix (FY2020 4Q)

We give the details of each goal along with the details of the achievements here.

5.1 Use of uniform basis as a nested basis block low-rank (BLR) matrix (FY2020 1Q)

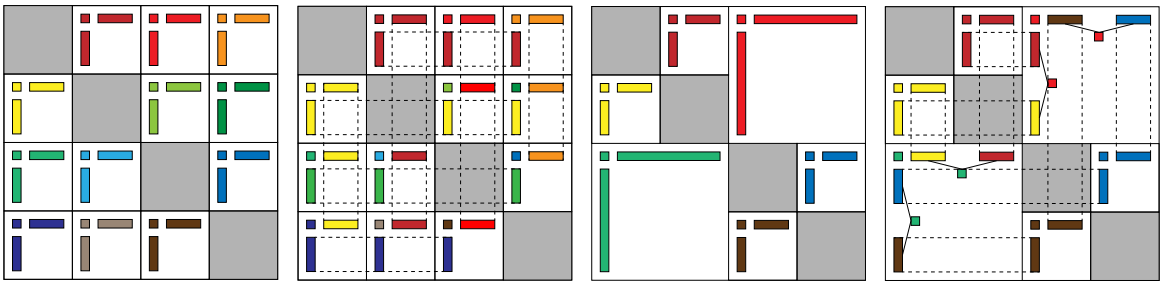
5.1.1 Research plan

In order to understand the significance of the present research plan, it is necessary to understand the subtle differences between the low-rank matrix structures shown in Fig. 1. The block low-rank (BLR) matrix shown in (a) is not-hierarchical and does not have a uniform basis. It is possible to have a uniform basis in BLR as shown in (b), though this has never been used in existing related work. The H-matrix shown in (c), has a hierarchical structure, but does not have a uniform basis. It is also called an hierarchical off-diagonal low-rank (HODLR) matrix when the off-diagonal blocks are not subdivided further, as shown in (c). The H^2-matrix shown in (d), has a hierarchical structure, and has a uniform basis, which is nested within a hierarchical structure for the basis. It is also called a hierarchically semi-separable (HSS) matrix when the off-diagonal blocks are not subdivided further, as shown in (d).

HACApK uses a non-nested H-matrix, and requires each block to store a long basis matrix. Since these basis matrices must be communicated during a distributed LU factorization, this results in a significant amount of communication. For a nested basis formulation, only the small translation matrix needs to be stored for each block. This reduces the amount of communication significantly. One disadvantage of the nested basis formulation is that it requires extra bookkeeping for the binary tree structure of the nested basis. Therefore, in FY2020 we plan to develop a simpler version of the nested basis called the uniform basis. The uniform basis does not use a hierarchical structure, and can be thought of as a nested basis equivalent for the (BLR) matrix. Since there is no hierarchical structure, the bookkeeping of the binary tree is unnecessary for the uniform basis.

5.1.2 Achievements

We have successfully implemented the uniform basis BLR during 1Q of FY2020. This is not only an important incremental step towards developing a nested basis H^2-matrix with superior scalability and complexity, but also a novel matrix structure that has both the parallel scalability of BLR and the low memory usage of the uniform basis. As far we know, this is the first implementation of such a structure, and we are preparing a submission to ACM TOMS based on the novel findings. A comparison of the LU factorization time between a dense matrix, BLR matrix, and BLR matrix with uniform basis is shown in Fig. 2. We can see that the LU factorization of BLR matrix is of course faster than that of a dense matrix. We also see that the BLR with uniform basis is much faster than the original BLR, and the complexity with respect to N is much better. The accuracy of the low-rank approximation is set to 10^{-10} for both the BLR and BLR(uniform), so the BLR can be even faster if less accuracy is required. Even for this high accuracy case, the BLR(uniform) can compute the LU factorization of a originally dense matrix of size $N = 32,768$ on a single CPU in about 1 second, which is over 100 times faster than the dense LU. This is speed up is partially due to the performance optimization of the inner kernels, which are now twice as fast compared to batched MKL [2].



(a) BLR (b) BLR (uniform basis) (c) H-matrix/HODLR (d) H^2-matrix/HSS

Fig. 1: Illustration of the different low-rank structured matrices. Blocks with the same color share the basis.

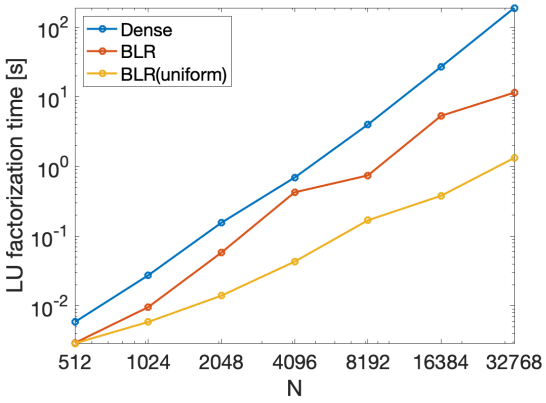


Fig. 2: Comparison of LU factorization time between a dense matrix, BLR matrix, and BLR matrix with uniform basis. The accuracy of the low-rank approximation is set to 10^{-10} for both the BLR and BLR(uniform).

5.2 QR factorization on TensorCores with refinement (FY2020 2Q)

5.2.1 Research plan

In order to achieve our goal for FY2019 of developing a highly optimized batched randomized SVD, we first developed a tall-skinny QR (TSQR) using TensorCores. Unlike many applications of the QR factorization, we use the QR factorization to perform the low-rank approximation, which does not require very high accuracy anyway. Therefore, we can tolerate the errors from the low-precision arithmetic up to a certain point. Randomized SVD makes use of the TSQR, so we are basically optimizing the TSQR operation. Therefore, our focus will be to extend the TSQR on TensorCores, to achieve higher accuracy. Our experiments in FY2019 showed that the use of TensorCores increased the residual error and orthogonality of the QR factorization to 10^{-2}, which is equivalent to using FP16. TensorCores perform the multiplication in FP16 but the accumulation is done with FP32. However, the conversion to FP16 for the input is what increases the error, and it does not matter that the accumulation is done with FP32. Therefore, we will develop a refinement method that stores the residuals of the FP16 operations in an auxiliary FP16 variable. This will allow us to still benefit from the speed of TensorCores, while retaining the accuracy of FP32 computations. Since the applications we are targeting can tolerate FP32 accuracy, this will make our TSQR on TensorCores usable in our final application.

5.2.2 Achievements

The use of floating point numbers with fewer bits such as single precision (FP32) and half precision (FP16) can cause the following problems.

- Roundoff error – where the least significant digits of the Mantissa are truncated
- Cancelation of significant digits – where the subtraction of two large numbers results in a cancellation of significant digits
- Loss of trailing digits – when a small number is added to a large number the trailing digits are lost

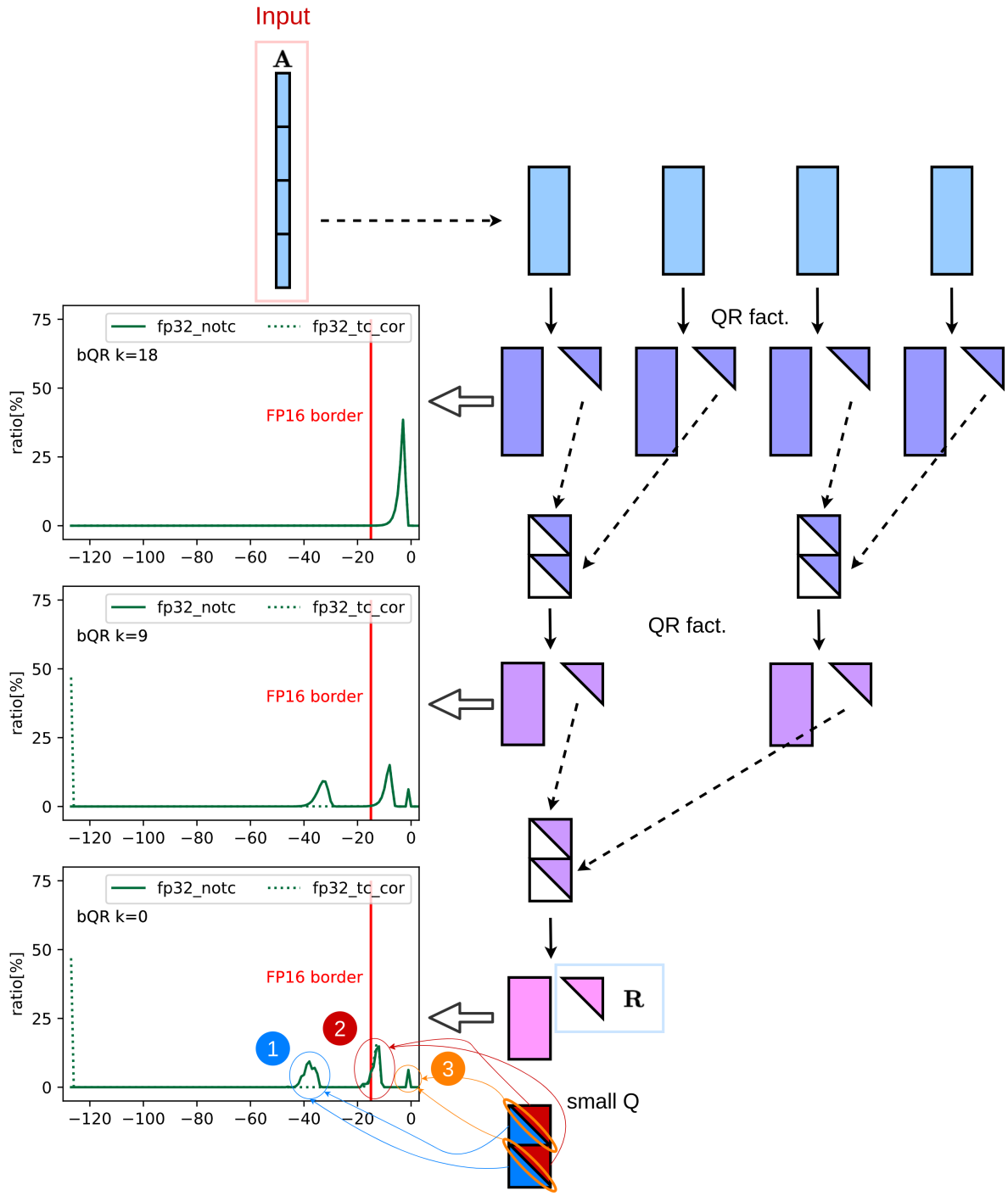


Fig. 3: Error of TSQR when using FP16

- Overflow/underflow of the exponent – when the value becomes too large/small

When performing operations on the Tensor-Core, the input matrix is converted to FP16. This causes roundoff error to the elements of the input matrix, and could also result in overflow/underflow if the value falls outside of the dynamic range supported by FP16.

As shown in Fig.3, TSQR performs a sequence of QR factorizations recursively. As the QR factorization is performed recursively, the dynamic range of the matrix elements changes. When the dynamic range crosses the boarder of the FP16 minimum, this results in a gradual underflow as defined by the subnormal numbers in IEEE754. This does not immediately produce the wrong result, but could cause the degradation of accuracy for the residual and orthogonality of the QR decomposition. It was also observed that certain parts of the matrix are more prone to underflow than other parts of the matrix.

The residual, orthogonality, computation time, and memory usage of the TSQR on TensorCores is shown in Fig. 4. "TC-noCor" is the TensorCore with no error correction, "TC-Cor" is the TensorCore with error correction, "noTC" is without TensorCore using FP32, and "cuSOLVER" is cuSOLVER using FP32. We see that the use of TensorCores without error correction results in a residual of 10^{-1} (about 10% error). With error correction, this can be reduced to 10^{-3}. This is only slightly worse than the case where the computation is done in FP32 without TensorCores, which has a residual of approximately 10^{-4}. cuSOLVER using FP32 has even better accuracy due to the different algorithm it uses. The orthogonality shows a similar trend. With respect to the computation time, we see that our implementation on TensorCores with error correction, is always faster than the FP32 computation without TensorCores. It is also faster than cuSOLVER when the matrix size is large enough. We also see that the memory consumption of cuSOLVER is larger than the other methods. These results were presented at ISC'20 [4].

5.3 Eigenvalue computation based on BLR-QR (FY2020 3Q)

5.3.1 Research plan

In FY2019 we have developed a QR factorization based on the BLR structure, but this alone could not be used in any application. In FY2020 we plan to use our BLR-QR inside a QR method to compute Eigenvalues. This will allow us to reduce the computational complexity of Eigenvalues of a dense matrix from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^{3/2})$. The original version of the BLR-QR in HACApK was based on the modified Gram-Schmidt algorithm and had poor orthogonality. This can be improved by performing another orthogonalization. We also plan to develop an alternative BLR-QR that uses the Householder projection, and is based on the tile-QR algorithm [1].

5.3.2 Achievements

In the 3Q of FY2020, we used the BLR-QR for the eigenvalue computation of a dense matrix arising from a electromagnetic scattering problem using a surface charge method. We used a BLR structure for the matrix and performed a BLR-QR using the technique developed in FY2019. The BLR matrix uses a weak admissibility where only the diagonal blocks are dense and all off-diagonal blocks are compressed to a low-rank matrix. The block size of the BLR matrix was set to $\mathcal{O}(N^{0.5})$, since this is known to yield optimal complexity. The QR decomposition uses a modified block Gram Schmidt method. Then, this QR decomposition is used inside a QR method to obtain the eigenvalues, where the following iteration is performed until convergence

$$Q_n, R_n = QR(A_n) \tag{1}$$

$$A_{n+1} = R_n * Q_n \tag{2}$$

We measured the maximum relative norm of the difference between the approximated eigenvalue e_n and the exact eigenvalue e_{exact} from the following equation

$$max((e_n(i) - e_{exact}(i))/e_{exact}(i)) \tag{3}$$

We also shift the eigenvalues to avoid instability by subtracting $I * A_n(N, N)$ from A_n. The maximum relative norm error is shown in Fig. 5. We observe that the error drops initially but reaches a stagnation point. This is caused by the approximation error in the BLR-QR computation.

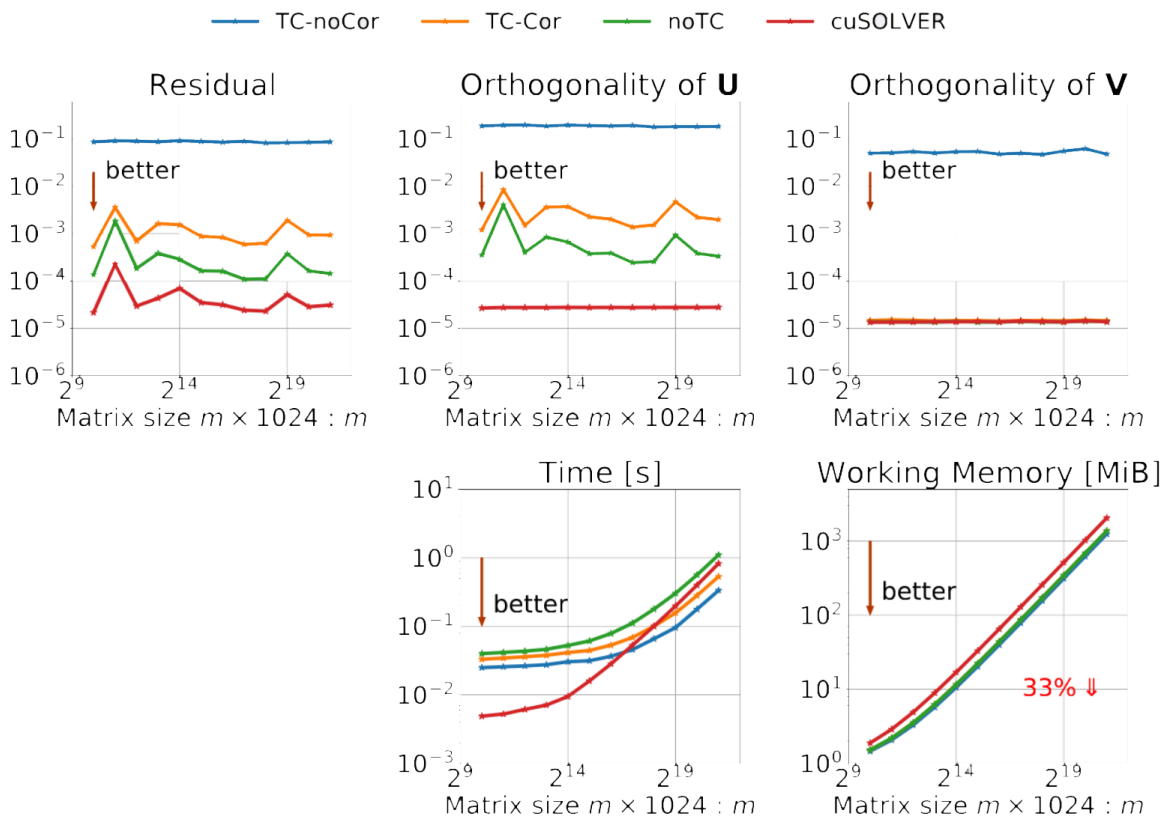


Fig. 4: Residual, orthogonality, time, and memory of randomized SVD on TensorCores.

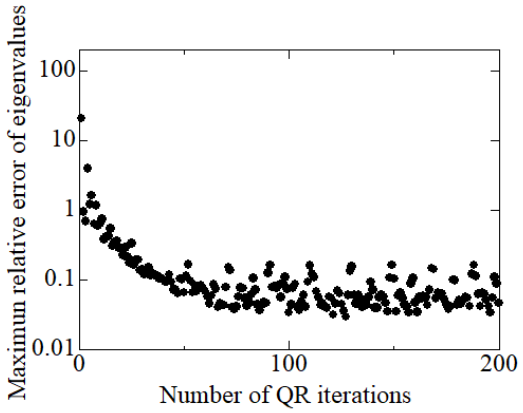


Fig. 5: Maximum relative norm error of eigenvalues during QR method iteration.

5.4 GPU implementation of lattice H-matrix (FY2020 4Q)

5.4.1 Research plan

In FY2019 we have shown that combining the scalability of the BLR matrix with the computational complexity of the H-matrix results in a scalable method that has $\mathcal{O}(Nlog^2N)$ complexity for LU factorization, which we named "lattice H-matrix". Although this method was implemented as an extension of HACApK, which has a GPU implementation for matrix-vector multiplications, the lattice H-matriix was done in a separate branch of the code, so it does not have a GPU implementation. In FY2020 we plan to extend the lattice H-matrix to include the GPU implementation.

5.4.2 Achievements

The lattice H-matrix is working on the CPU and the plain H-matrix is working on GPUs. The lattice H-matrix has been implemented, but it is still going through a debugging phase. There are still some cases where it returns a larger error than expected.

6 Progress during FY2020 and Future Prospects

6.1 Use of uniform basis as a nested basis block low-rank (BLR) matrix (FY2020 1Q)

For the uniform basis BLR work, we achieved the original research objectives. The shared basis BLR is a simple but novel structure that has a good balance between the arithmetic complexity and the amount of parallelism. Using this shared basis BLR structure, we were able to achieve over 100 times speed up for a LU decomposition compared to the dense LU in MKL for a matrix size of N=32,768 (as shown in Fig. 2).

6.2 QR factorization on TensorCores with refinement (FY2020 2Q)

For the QR factorization on TensorCores, we achieved the original research objectives. We extended our work on error corrected matrix multiplication on TensorCores. Careful consideration of the dynamic range of the matrix elements allowed us to achieve much higher accuracy than any other implementation of QR decomposition on TensorCores.

6.3 Eigenvalue computation based on BLR-QR (FY2020 3Q)

For the eigenvalue computation using BLR-QR, we were able to achieve our original research objectives. We have shown that the BLR-QR method can be used along with the QR method to obtain eigenvalues. This is the first work to demonstrate an eigenvalue solver using the BLR matrix, which has $\mathcal{O}(N^2)$ complexity instead of $\mathcal{O}(N^3)$.

6.4 GPU implementation of lattice H-matrix (FY2020 4Q)

For the GPU implementation of lattice H-matrix, we were not able to achieve our original research objective. The implementation has been completed, but there remains some unexplained behavior in the code, which is causing a large error in the solution. We believe this can be resolved in time by constructing unit tests for each component and checking their validity.

7 List of Publications and Presentations

7.1 Journal Papers

1. M. R. Aripansyah, R. Yokota, QR Decomposition of Block Low-Rank Matrices, ACM Transactions on Mathematical Software (in preparation).

7.2 Conference Papers

2. S. Deshmukh, R. Yokota, G. Bosilca, Efficient Batched Low Rank Matrix Multiplication Using Register Blocking, ACM Transactions on Mathematical Software (in preparation).

7.3 Oral/Poster Presentations

3. S. Deshmukh and R Yokota. Distributed Memory Task-Based Block Low Rank Direct Solver, ISC High Performance 2020 (Research Poster), June, 2020.
4. H. Ootomo, R. Yokota, Randomized SVD on TensorCores, ISC High Performance 2020, (Research Poster), June, 2020.