jh210024-NAH

Hierarchical Low-Rank Approximation Methods on Distributed Memory and GPUs

Rio Yokota (Tokyo Institute of Technology)

Abstract

The purpose of this research is to develop a scalable and highly optimized open source library for hierarchical low-rank approximation of dense matrices. 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. The four main goals for the fiscal year 2021 are: 1) Improving the complexity of the $\mathcal{H}$-matrix LU decomposition from $\mathcal{O}(N \log^2 N)$ to $\mathcal{O}(N)$, 2) Comparison of runtimes such as ParSEC, QUARK, and StarPU for the asynchronous execution of $\mathcal{H}$-matrix LU, 3) Benchmarking of our $\mathcal{H}$-matrix code against other open source libraries, 4) Development of an $\mathcal{H}$-matrix LDL decomposition. We were able to achieve our research goal for all 4 objectives. Above that, we were able to develop a $\mathcal{O}(N)$ LU factorization for dense matrices without a dependency on trailing sub-matrices. This was not in our original plan, but is a disruptive technology that makes runtime systems irrelevant, and is possibly the ultimate form of hierarchical low-rank approximation methods.

1 Basic Information

1.1 Collaborating JHPCN Centers

Hokkaido, Tokyo, Tokyo-Tech, Nagoya, Kyoto

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 Tensor-Cores 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

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

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

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

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

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

Hana Hoshino (Tokyo Institute of Technology) Application to reinforcement learning

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

Sora Takashima (Tokyo Institute of Technology) Application to vision transformer optimization

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

Tomoya Takahashi (Tokyo Institute of Technology) Application to autonomous driving models

Kai Okawa (Tokyo Institute of Technology) Application to visual SLAM

Sixue Wang (Tokyo Institute of Technology) Application to second order optimization in deep learning

2 Purpose and significance of Research

2.1 Purpose 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 matrices 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 $\mathcal{O}(Nlog^2N)$ time and $\mathcal{O}(N)$ memory, have controllable arithmetic intensity, have asynchronous communication, and can exploit deep memory hierarchy, which makes them an ideal solver/preconditioner for the Exascale era.

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. In particular, we will extend the previous $\mathcal{O}(N \log^2 N)$ method to a new $\mathcal{O}(N)$ method. We also plan to study the effect of mixed precision in the context of hierarchical low-rank matrices.

2.2 Significance of Research

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. There is still ample room for investigation regarding the use of such low-precision in scientific computing applications, where methods such as iterative refinement have recently gained interest. $\mathcal{H}$-matrices can be used as a scalable preconditioner for such problems, and we aim to quantify the advantage over existing state-of-the-art methods in this JHPCN project.

Furthermore, batched operations on GPUs are becoming popular and libraries such as MAGMA and CUBLAS are providing low-level functions that can process many small dense matrix operations in large batches. $\mathcal{H}$-matrices can benefit greatly from such batched dense linear algebra libraries, and in doing so will be able to extract a large portion of the performance of the latest GPU and many-core architectures including Tensor Cores. Since libraries like MAGMA and CUBLAS are optimized to use Tensor Cores, we do not have to do the implementation ourselves.

3 Significance as JHPCN Joint Research Project

Each member of this project has different expertise, all of which are essential for the development and verification of a high performance $\mathcal{H}$-matrix library. R. Yokota' s group is currently developing a C++-based $\mathcal{H}$-matrix code FRANK that uses advanced C++ features to provide a collection of primitives for performing $\mathcal{H}$-matrix computation with hybrid parallelism for MPI, OpenMP, and CUDA over half of the project members are students in his group. A. Ida and T. Iwashita are developers of HACApK – a hybrid MPI-OpenMP-CUDA implementation of the $\mathcal{H}$-matrix. T. Hiraishi has experience in load-balancing for distributed memory $\mathcal{H}$-matrix codes. I. Yamazaki is the developer of dense linear algebra libraries such as MAGMA and PLASMA. S. Oshima, T. Hanawa and T. Hoshino have expertise in tuning solvers for GPUs and Xeon Phi. K. Nakajima has expertise in parallel preconditioned iterative solvers. The combination of these expertise is necessary for achieving the goals mentioned above. There are a few existing $\mathcal{H}$-matrix implementations, but they are limited to shared memory and have not been ported to GPUs. To our knowledge, HACApK and HiCMA are the only multi-GPU $\mathcal{H}$-matrix codes available at the moment. This could only have been done through a JHPCN international collaboration between the experts in each area.

4 Outline of Research Achievements up to FY2020

Up to FY2020 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 ma-

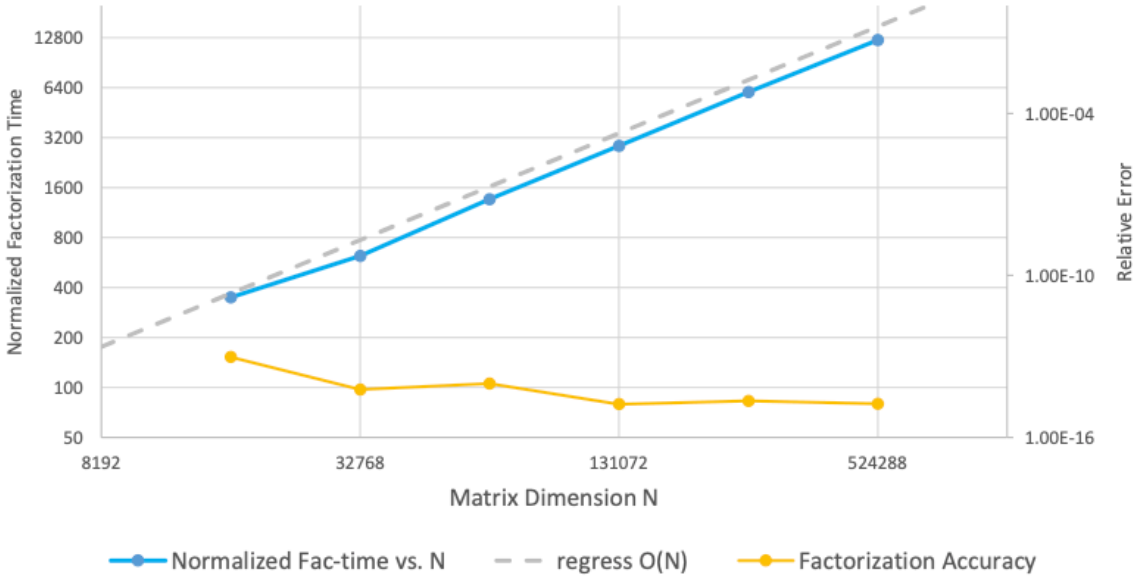

Fig. 1 Computation time (blue) of LU decomposition as a function of N and the approximation error (yellow)

trix. 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 FY2019 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, and implemented the QR decomposition using the BLR matrix.
- In FY2020 we implemented the uniform basis BLR, and QR factorization on TensorCores with error correction. We also developed a Eigenvalue computation based on BLR-QR, and developed a GPU implementation of the lattice $\mathcal{H}$-matrix.

5 Details of FY2021 Research Achievements

5.1 Improving the complexity of the $\mathcal{H}$-matrix LU decomposition from $\mathcal{O}(N \log^2 N)$ to $\mathcal{O}(N)$

5.1.1 Research plan

Until FY2020 the cost of an $\mathcal{H}$-matrix LU decomposition was believed to be $\mathcal{O}(N \log^2 N)$. We have discovered a novel algorithm that reduces the cost of $\mathcal{H}$-matrix LU decomposition to $\mathcal{O}(N)$. We are not aware of any open source libraries that implement this $\mathcal{O}(N)$ technique. Therefore, our focus during FY2021 1Q is to implement the $\mathcal{O}(N)$ $\mathcal{H}$-matrix LU decomposition algorithm and show empirical evidence that our method is indeed $\mathcal{O}(N)$.

5.1.2 Progress

We were able to finish the development of the $\mathcal{O}(N)$ algorithm. The results are shown in Fig. 1, where the LU factorization time in shown in the blue line, while the relative error of the factorization is shown in the yellow line. The factorization time is shown in milliseconds. As can be seen from the figure, our new method shows ideal $\mathcal{O}(N)$ complexity. We can compute the LU factorization of a 500k by 500k dense matrix in 12.8 seconds on a single CPU. Furthermore, the error shown in the yellow line is close to the double precision rounding error. In these experiments, we are using a large enough rank to achieve such accuracy. These results were achieved during the first half of FY2021.

In the second half of FY2021, we have extended this $\mathcal{O}(N)$ dense direct solver to distributed memory. For HSS matrices, it is possible to remove the dependency on the trailing matrices during Cholesky/LU factorization, which results in a highly parallel algorithm. However, the weak admissibility of HSS causes the rank of off-diagonal blocks to grow for 3-D problems, and the method is no longer $\mathcal{O}(N)$. On the other hand, the strong admissibility of $\mathcal{H}^2$-matrices allows it to handle 3-D problems in $\mathcal{O}(N)$, but introduces a dependency on the trailing matrices. In the present work, we pre-compute the fill-ins and integrate them into the shared basis, which allows us to remove the dependency on trailing-matrices even for $\mathcal{H}^2$-matrices. Comparisons with a block low-rank factorization code LORAPO showed a maximum speed up of 140x for a 3-D problem with complex geometry, as shown in Fig. 2.

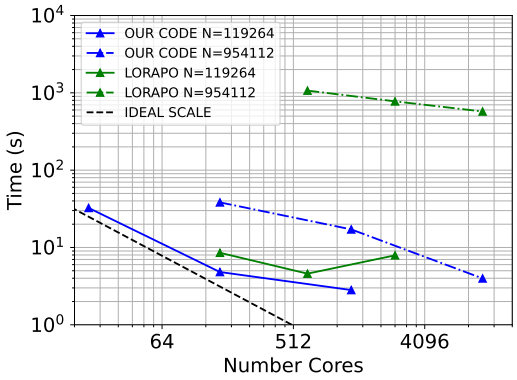

Fig. 2 Strong scaling experiments on multiple nodes for different problem sizes of the hemoglobin boundary element problem using our code and LORAPO.

5.2 Comparison of runtimes such as ParSEC, QUARK, and StarPU for the asynchronous execution of $\mathcal{H}$-matrix LU

5.2.1 Research plan

The $\mathcal{H}$-matrix LU/QR decomposition is an extremely challenging algorithm to parallelize. It is more challenging that dense matrix decomposition since the existence of low-rank blocks makes the workload highly unbalanced. It is more difficult than sparse matrix factorization, since the hierarchical structure of the matrix creates non-trivial dependencies across multiple levels. Runtimes such as ParSEC, QUARK, and StarPU are now commonly used in linear algebra libraries. These runtimes use a directed acyclic graph to identify which tasks can be handled in parallel in an asynchronous manner. However, the interface, capabilities, and overhead of these runtimes vary, and it is not obvious which runtime is suitable for $\mathcal{H}$-matrix LU/QR decomposition. Therefore, we aim to create a unified interface to these runtimes so that we may switch between them easily. Then, we can compare the performance of these runtimes on various matrix structures and hardware architectures.

5.2.2 Progress

During the first half of FY2021, we were preparing the implementation of our code in ParSEC, QUARK, and StarPU. However, these runtime systems have non-negligible overhead, and extracting good performance from these tool is not a straightforward task. In the second half of FY2021, we have decided to use a unified framework which allows us to compare different runtime systems called AL4SAN. This is a lightweight library for abstracting the APIs of task-based runtime engines, developed by our collaborators at KAUST. This has significantly im-

prove the productivity of our code development, but we are still investigating the optimal configurations for each of these runtime systems, to work with our $\mathcal{H}$-matrix library. Moreover, the distributed $\mathcal{H}^2$-ULV method described in the previous sub-section, removes the dependencies for the trailing submatrices, so there is no need to use runtime systems. Therefore, there is no need to compare the performance of these runtime systems anymore.

5.3 Benchmarking of our $\mathcal{H}$-matrix code against other open source libraries

5.3.1 Research plan

There are over 20 different implementations of structured low-rank approximation methods. The matrix structure used in these implementations slightly differs between BLR, HODLR, HSS, $\mathcal{H}$-matrix, and H2-matrix. Throughout our JHPCN project we have continuously developed a code that can use any of these structures. This is made possible through the use of a generic "Hierarchical" class in C++, which can handle any type of hierarchical low-rank structure. However, there have been no direct comparisons between these different implementations, which makes it difficult for the user to select the optimal library for their task of interest. To address this problem, we will perform a thorough benchmark between the representative implementations, and show which method is faster, scalable, accurate, robust, etc.

5.3.2 Progress

During the first half of FY2021, we had identified GOFMM, STRUMPACK, and MatRox to be the most competitive implementations. However, we have found that all of these codes have serious issues. First of all MatRox only has the capability to perform $\mathcal{H}$-matrix-vector multiplication and cannot perform the $\mathcal{H}$-matrix LU factorization that we are interested in. STRUMPACK does not have a distributed matrix construction scheme so it cannot handle the matrix size we are using in our experiments. GOFMM does not seem to compile in the environment we are using. Therefore, we decided to compare with an alternative state-of-the-art implementation LORAPO, which can handle $\mathcal{H}$-matrix LU factorization on distributed memory and compiles in our environment. The results of this comparison are shown in Fig. 2.

5.4 Development of an $\mathcal{H}$-matrix LDL decomposition

5.4.1 Research plan

During FY2020 we have developed a QR decomposition using the BLR structure, which is a non-hierarchical low-rank matrix structure that has $\mathcal{O}(N^2)$ cost for the QR decomposition. In FY2021 4Q, the original plan was to extend this to the QR decomposition of $\mathcal{H}$-matrices to further improve the complexity [1]. The long term goal of this project is to compute the eigenvalues of a dense matrix, and the plan was to use the QR method. However, after careful consideration of alternative methods to compute eigenvalues, we have decided to switch to the LDL decomposition, which can be used to compute eigenvalues by slicing the spectrum. The application we have in mind is the eigenvalue computation in electronic structure computations.

5.4.2 Progress

We have identified similarities between the LU decomposition and LDL decomposition that can be exploited for making the transition between the two methods easier. We have extended the $\mathcal{O}(N)$ LU decomposition implementation to the LDL decomposition with only minor modifications to the code. We used the LDL decompostion during the slicing of the spectrum using a binary search to obtain the nth eigenvalue of a dense matrix in $\mathcal{O}(N \log N)$ time. The computational time for different matrix sizes is shown in Fig. 3.

6 Progress during FY2021 and Future Prospects

The four main goals for the fiscal year 2021 are:

(1) Improving the complexity of the $\mathcal{H}$-matrix LU decomposition from $\mathcal{O}(N \log^2 N)$ to $\mathcal{O}(N)$
(2) Comparison of runtimes such as Par-

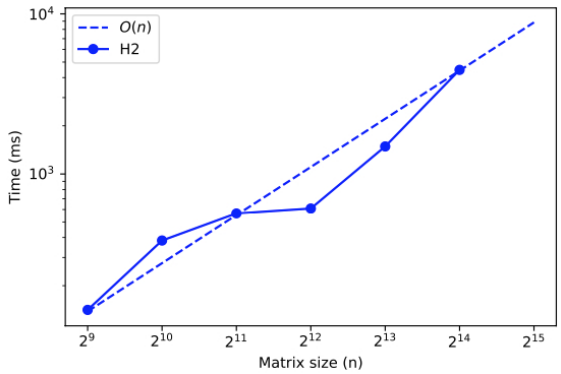

Fig. 3 Normalized time to compute the 8th smallest eigenvalue for various matrix sizes using the $\mathcal{H}^2$-LDL factorization to recursively split the spectrum.

SEC, QUARK, and StarPU for the asynchronous execution of $\mathcal{H}$-matrix LU

(3) Benchmarking of our $\mathcal{H}$-matrix code against other open source libraries

(4) Development of an $\mathcal{H}$-matrix LDL decomposition.

For (1), we were able to show perfect $\mathcal{O}(N)$ scaling with our $\mathcal{H}^2$-ULV method, while achieving double precision accuracy for the LU factorization of a dense matrix, as shown in Fig. 1. We have also extending this work to develop an $\mathcal{O}(N)$ LU factorization for dense matrices without a dependency on trailing sub-matrices, which we have submitted to SC22 and received positive reviews at this point.

For (2), we used a unified framework which allows us to compare different runtime systems such as StarPU, QUARK, and PaRSEC, which is provided by our collaborators at KAUST. However, the distributed $\mathcal{H}^2$-ULV method described in (1), removes the dependencies for the trailing sub-matrices, so there is no need to use runtime systems. Therefore, there is no need to compare the performance of these runtime systems anymore.

For (3), we have benchmarked against other state-of-the-art libraries such as LORAPO. Our latest code shows a 140 fold speed up over LORAPO as shown in the previous section. Therefore, we consider this task to be completed with more progress than originally planned.

For (4), we have successfully developed a $\mathcal{H}$-matrix LDL decomposition, and used it while slicing the spectrum using a binary search to obtain the nth eigenvalue of a dense matrix in $\mathcal{O}(N \log N)$ time. This algorithm has application in our Kiban B project, were we apply it to an electronic structure computation.

7 List of Publications and Presentations

Journal Papers (Refereed)

- M. R. Aripansyah, R. Yokota, 'QR Decomposition of Block Low-Rank Matrices', ACM Transactions on Mathematical Software, accepted.

Oral/Poster Presentations

- R. Yokota., 階層的低ランク近似法に関するレビュー, 第 40 回計算数理工学フォーラム, 9 月, 2021.
- R. Yokota., Overview of structured low-rank approximation methods, IUTAM Symposium on Computational Methods for Large-Scale and Complex Wave Problems, Jun., 2021.