jh230009

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 2022 are: 1) Application of the runtime system PaRSEC developed at UTK to our $\mathcal{H}$-matrix library, 2) Application of the LDL factorization to find the k-th eigenvalue in electronic structure calculations, 3) Extending the $\mathcal{O}(N)$ $\mathcal{H}$-matrix LU factorization to multi-GPU, 4) Completing the GPU implementation for BLR-QR in the HACApK library, 5) BLR tridiagonalization for computing the full eigenspectrum. We were able to achieve our research goal for all 5 objectives. This year's results were published in top journals like IJHPCA and top conferences such as ICPP.

1 Basic information

1.1 Collaborating JHPCN centers

- Hokkaido University
- The University of Tokyo
- Tokyo-tech
- Nagoya University
- Kyoto University
- Osaka University

1.2 Theme area

- Large-scale computational science

1.3 Research area

- Very large-scale numerical computation

1.4 Project members and their roles

Rio Yokota (Overall coordination)

Ichitaro Yamazaki (Overall advise)

Akihiro Ida (Lattice $\mathcal{H}$-matrix)

Takeshi Iwashita (Electromagnetics App.)

Takeshi Fukaya (QR)

Satoshi Ohshima (GPU optimization)

Kengo Nakajima (Preconditioning)

Toshihiro Hanawa (GPU optimization)

Tetsuya Hoshino (GPU optimization)

Tasuku Hiraishi (MPI)

Sameer Deshmukh (PaRSEC)

Muhammad Ridwan Apriansyah (QR)

Qianxiang Ma (Algorithm)

Thomas Spendlhofer (Iterative Refinement)
Kai Okawa (Visualization)
Shukai Nakamura (CPU optimization)
Shota Nakamura (MPI)
Hiro Ishii (Hessian matrix)
Zhaoqing Wang (GPU optimization)
Cong Bai (Hessian matrix)
Clement Bazan (Fisher matrix)
Ishikawa Satoki (Auto-tuning)
Tomokazu Saito (FMM)

2 Purpose and Significance of the Research

2.1 Purpose of Research

The purpose of this research is to develop a scalable and highly optimized open source library for structured low-rank approximation of dense matrices, *e.g.* $\mathcal{H}$-matrix, $\mathcal{H}^2$-matrix, HSS, HODLR, BLR. In this project report we will simply call these various types of structured low-rank approximations as $\mathcal{H}$-matrices. Such large dense matrices naturally appear in electromagnetic, seismic, quantum, and fluid simulations, in scientific computing. 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.

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 Hatrix 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 FY2022

Up to FY2022 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 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 computa-

tion based on BLR-QR, and developed a GPU implementation of the lattice $\mathcal{H}$-matrix.

- In FY2021 we improved the complexity of the $\mathcal{H}^2$-matrix LU decomposition from $\mathcal{O}(N \log^2 N)$ to $\mathcal{O}(N)$, compared various runtime systems, benchmarked against other libraries such as STRUMPACK and LORAPO, and extended to LDL decomposition.
- In FY2022 we extended the GPU implementation of $\mathcal{H}$-matrices to use Tensor Cores, extended the $\mathcal{O}(N)$ $\mathcal{H}$-matrix LU factorization to distributed memory, and extended the $\mathcal{O}(N)$ $\mathcal{H}$-matrix LU factorization to LDL factorization.

5 Details of FY2023 Research Achievements

The five main goals for the fiscal year 2023 are

1. Application of the runtime system PaRSEC developed at UTK to our $\mathcal{H}$-matrix library
2. Application of the LDL factorization to find the k-th eigenvalue in electronic structure calculations
3. Extending the $\mathcal{O}(N)$ $\mathcal{H}$-matrix LU factorization to multi-GPU
4. Completing the GPU implementation for BLR-QR in the HACApK library
5. BLR tridiagonalization for computing the full eigenspectrum

We were able to complete all five tasks. Task 1 was published in [6], task 2 was published in [5], task 3 has been published in [1], task 4 has been published in [9,10,12], task 5 has been published in [3,8]. Other publications in Section 7 are results from previous years that were published in FY2023.

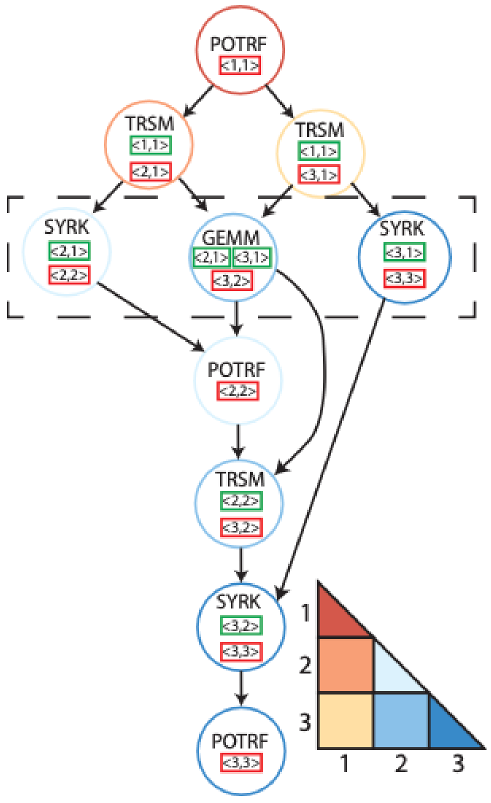

Fig. 1 DAG of the Cholesky decomposition on HSS-matrix.

5.1 Application of the runtime system PaRSEC developed at UTK to our $\mathcal{H}$-matrix library

5.1.1 Research plan

LAPACK has evolved into ScaLAPACK for MPI, PLASMA for many-core, and MAGMA for GPUs, which are all developed at Jack Dongarra’ s group at UTK. The latest invention from the UTK group is SLATE, which will be replacing the above libraries in the near future. SLATE is one of the core libraries being developed as art of the Exascale Computing Project (ECP). SLATE extracts parallelism on distributed memory systems through the use of runtime systems such as PaRSEC. Our goal is to use the same runtime system for $\mathcal{H}$-matrices instead of dense matrices. Unlike, dense matrices which have equal workload between the subblocks, $\mathcal{H}$-matrices contain subblocks with varying rank. Therefore, $\mathcal{H}$-matrices are much more difficult to load-balance, so its parallelization through PaRSEC will require considerable effort.

5.1.2 Progress

We have proposed a ULV factorization for HSS matrices, and provided an implementation, HATRIX-DTD, using the PaRSEC runtime system. The PaRSEC runtime system, can asynchronously execute tasks by resolving the dependencies using a directed acyclic graph (DAG) as shown in Fig. 1. We have showed that factorization of structured dense matrices arising from a diverse set of kernels. This is achieved as a result of the asynchronous runtime system and the lower computational intensity of the HSS-ULV factorization. Using HATRIX-DTD, we show that our implementation has much faster solution time than established state-of-the-art implementations such as STRUMPACK and LORAPO as shown in Fig. 2 [6].

5.2 Application of the LDL factorization to find the k-th eigenvalue in electronic structure calculations

5.2.1 Research plan

During FY2022 we have extended the LU factorization in our Hatrix library to LDL factorization. The D matrix in LDL factorization is a diagonal matrix that can be used to perform a slicing the spectrum approach to compute the k-th Eigenvalue, where 'k' is a prescribed value defined by the user. In electronic structure computations, it is necessary to compute the k-th Eigenvalue. However, in FY2022 we were only able to apply to LDL factorization to simple test matrices. The goal for FY2023 2Q is to apply our LDL factorization to the actual matrices arising from electronic structure calculations.

5.2.2 Progress

We develop a generalized LDL decomposition of $\mathcal{H}^2$-matrices and combine it with the bisection eigenvalue algorithm to compute the k-th eigenvalue with controllable accuracy. In addition, if more than one eigenvalue is required, some of the previous computations can be reused to compute the other eigenvalues in parallel. Numerical experiments show that our method is more efficient than the state-of-the-art dense eigenvalue solver in LAPACK/ScaLAPACK and ELPA as shown in Fig. 3 [5].

5.3 Extending the O(N) $\mathcal{H}$-matrix LU factorization to multi-GPU

5.3.1 Research plan

During FY2022 we developed a novel $\mathcal{H}$-matrix LU factorization algorithm with no trailing sub-matrix dependencies, which has presented at SC22. This method is groundbreaking in the sense that it can perform LU factorization of dense matrices in an embarrassingly parallel fashion without waiting for the upper-left blocks to finish. However, the SC22 paper only had results for a flat

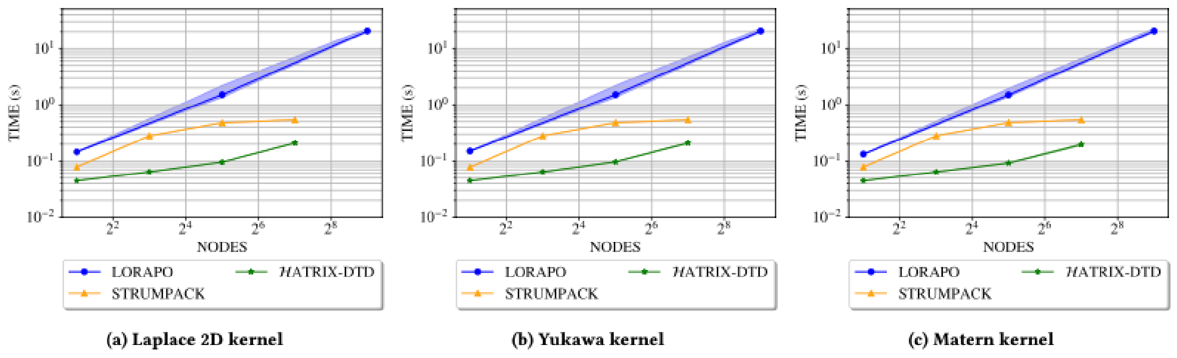

Fig. 2 Comparison between LORAPO and STRUMPACK for Cholesky factorization.

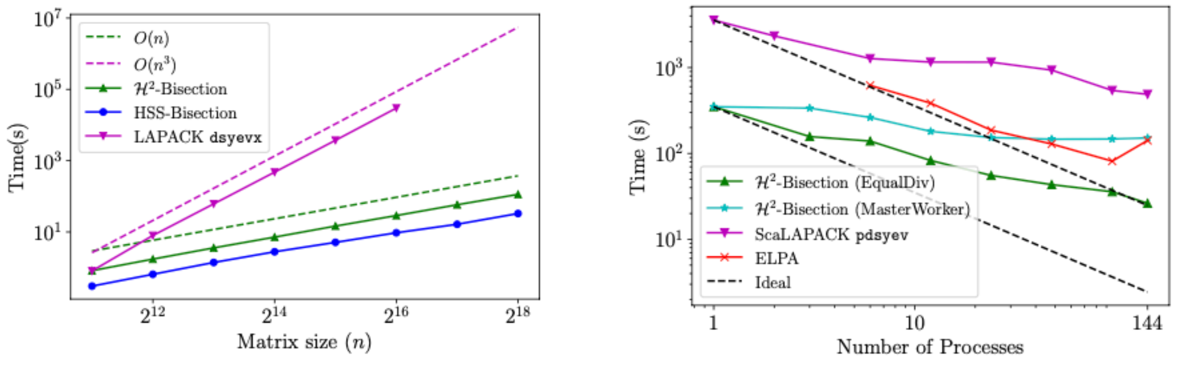

Fig. 3 Comparison between LAPACK, ScaLAPACK, and ELPA for k-th eigenvalue problem.

MPI implementation and no GPU implementation. The embarrassingly parallel nature of our algorithm makes it a perfect candidate for batched operations on GPUs, which can extract the full potential of Tensor Cores even when the matrices are small.

5.3.2 Progress

We have developed a highly scalable algorithm for an $\mathcal{O}(N)$ Cholesky factorization for rank-structured dense matrices and its multi-GPU implementation. By pre-computing the fill-ins and including them in the shared basis, we are able to devise an inherently parallel factorization algorithm even for strongly admissible $\mathcal{H}^2$-matrices. We also develop a novel algorithm for performing the forward and backward substitution in an inherently parallel manner. This inherently parallel factorization and substitution algorithms allows us to use batched kernels in cuBLAS and cuSOLVER, and extract the full potential GPUs even for small block sizes. Our method shows superiority to simpler low-rank matrix formats such as BLR and HSS in arithmetic complexity, and the ability to handle higher dimension geometry. We are able to solve a 3-D Yukawa potential problem of $N = 29,242,368$ under 1 second using 512 NVIDIA V100 GPUs as shown in Fig. 4 [1].

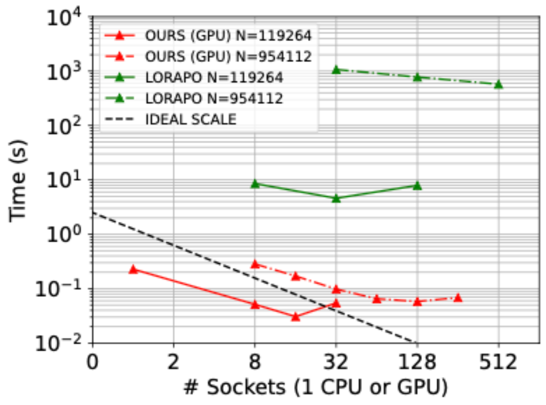

Fig. 4 Comparison with LORAPO for Cholesky factorization on GPUs.

5.4 Completing the GPU implementation for BLR-QR in the HACApK library

5.4.1 Research plan

The HACApK library is written in Fortran and its porting to GPUs is not straightforward. In the previous JHPCN projects we gradually started to port each subroutine in HACApK's BLR-QR factorization code to CUDA one by one. When only part of the BLR-QR is ported to CUDA, there will be many CPU-GPU data copy between the parts that run on the CPU and parts that run on the GPU. This causes a huge overhead, so the porting of the entire BLR-QR is necessary for BLR-QR to achieve decent performance on GPUs.

5.4.2 Progress

During FY2023, we have completed the porting of all Fortran subroutines in HACApK to CUDA functions. This allows us to keep all the data during the BLR-QR factorization on the GPU, instead of copying parts of the data back and forth [9,10,12].

5.5 BLR tridiagonalization for computing the full eigenspectrum

5.5.1 Research plan

Eigenvalue decomposition of dense matrices are performed by libraries such as ELPA or EigenExa. However, the dense matrix must first be tridiagonalized before the eigendecomposition is performed. This tridiagonalization can be done in $\mathcal{O}(N \log N)$ time if $\mathcal{H}$-matrices are applied. During FY2023, we have developed a novel algorithm for applying $\mathcal{H}$-matrices during the tridiagonalization phase of a dense eigenvalue decomposition.

5.5.2 Progress

We developed a fast tridiagonalization method based on the block low rank (BLR) structure, that reduces the complexity of the tridiagonalization from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^{7/3})$ [???]. Block Householder vectors are also formed using BLR-matrices. The procedure for forming a block tridiagonal structure is shown in Fig. 5. In numerical experiments of a string free vibration problem with known analytical solutions, for large eigenvalues, the calculated eigenvalues using the proposed method converge toward the analytical ones in accordance with the theoretical convergence curves. Owing to the reduced complexity, an eigenvalue decomposition of a matrix was solved with about N = 300,000, which is significantly larger than the limit of conventional methods for dense matrices, within a reasonable amount of time on CPU cores. For the calculation time, the proposed method was faster than the conventional method when the matrix size N was larger than a few tens of thousands [3,8].

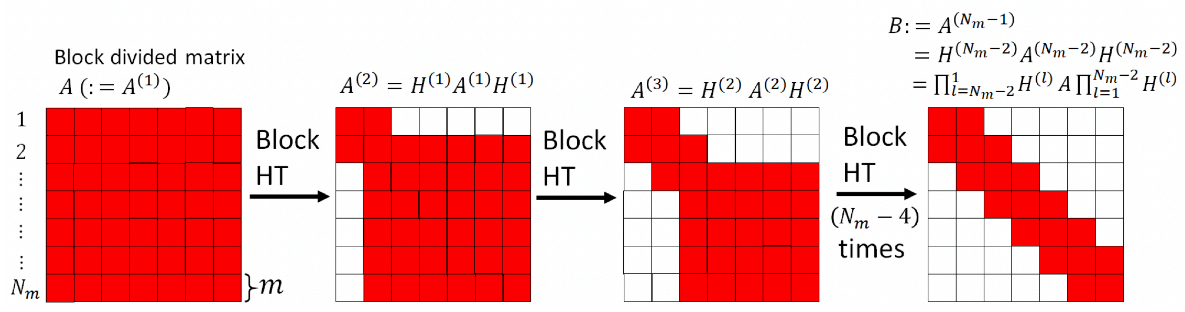
Block divided matrix
$A\ (:= A^{(1)})$
1
2
N_m
m
Block HT
$A^{(2)} = H^{(1)}A^{(1)}H^{(1)}$
Block HT
$A^{(3)} = H^{(2)}A^{(2)}H^{(2)}$
Block HT
$(N_m - 4)$ times
$B := A^{(N_m-1)}$
$= H^{(N_m-2)}A^{(N_m-2)}H^{(N_m-2)}$
$= \prod_{l=N_m-2}^{1} H^{(l)}\, A \prod_{l=1}^{N_m-2} H^{(l)}$

Fig. 5 A block-divided dense matrix A is transformed into a block tridiagonal matrix B using the block Householder transformation.

6 Self-review of Current Progress and Future Prospects

The five main goals for the fiscal year 2023 are

1. Application of the runtime system PaRSEC developed at UTK to our $\mathcal{H}$-matrix library
2. Application of the LDL factorization to find the k-th eigenvalue in electronic structure calculations
3. Extending the $\mathcal{O}(N)$ $\mathcal{H}$-matrix LU factorization to multi-GPU
4. Completing the GPU implementation for BLR-QR in the HACApK library
5. BLR tridiagonalization for computing the full eigenspectrum.

We were able to complete all five tasks. Task 1 was published in [6], task 2 was published in [5], task 3 has been published in [1], task 4 has been published in [9,10,12], task 5 has been published in [3,8]. Other publications in Section 7 are results from previous years that were published in FY2023.

7 List of publications and presentations

Journal Papers (Refereed)

1. Q. Ma, R. Yokota, “An Inherently Parallel $\mathcal{H}^2$-ULV Factorization for Solving Dense Linear Systems on GPUs”, International Journal of High Performance Computing Applications, 2024.
2. H. Ootomo, K. Ozaki, R. Yokota, “DGEMM on Integer Matrix Multiplication Unit”, The International Journal of High Performance Computing Application, 2024.
3. A. Ida, “Algebraic Partition Construction Method for Hierarchical Matrices”, IEEE Transactions on Magnetics, Vol. 60, pp. 1–4, 2024.
4. S. Deshmukh, R. Yokota, G. Bosilca, “Cache Optimization and Performance Modeling of Batched, Small, and Rectangular Matrix Multiplication on Intel, AMD, and Fujitsu Processors”, ACM Transactions on Mathematical Software, 2023.

Proceedings of International Conference Papers (Refereed)

5. M. R. Apriansyah, R. Yokota, Computing the k-th Eigenvalue of Symmetric $\mathcal{H}^2$-Matrices, International Conference on Parallel Processing (ICPP), Aug. 2023.
6. S. Deshmukh, R. Yokota, G. Bosilca, $\mathcal{O}(N)$ Distributed Direct Factorization of Structured Dense Matrices Using Runtime Systems, International Conference on Parallel Processing (ICPP), Aug. 2023.
7. H. Ootomo, H. Manabe, K. Harada, R. Yokota, Quantum Circuit Simulation by SGEMM Emulation on Tensor Cores and Automatic Precision Selection, ISC High Performance, May 2023.
8. A. Ida, An Algebraic Partition Construction Method for Hierarchical Matrices, COMPUMAG 2023, the 24th International Conference on the Computation of Electromagnetic Fields, 2023.

Presentations at International conference (Non-refereed)

9. S. Ohshima, A. Ida, N. Kawai, R. Yokota, I. Yamazaki(+), "Acceleration of BLR-QR Using CUDA Fortran+MIG+UVM", SWoPP, Aug. 2023.
10. S. Ohshima, A. Ida, N. Kawai, R. Yokota, I. Yamazaki(+), "Large Scale Acceleration of BLR-QR Factorization Using CUDA Fortran+MIG+UVM", SWoPP2023, July, 2023.
11. S. Ohshima, A. Ida, R. Yokota, I. Yamazaki(+), "GPU Performance Optimization and Autotuning in the 10,000 Core Era", ATTA2023, Dec. 2023.
12. S. Ohshima, "Considering multi process calculations on current GPU", ATAT in HPSC 2024, National Center for High-Performance Computing in Hsinchu Science Park, Mar. 2023.