jh170057-NAHI

Hierarchical Low-Rank Approximation Methods on Distributed Memory and GPUs

Rio Yokota（Tokyo Institute of Technology）

Hierarchical low-rank approximation methods such as H-matrix, H^2-matrix, and HSS can compress a dense matrix with $O(N^2)$ elements into a hierarchical matrix with $O(N)$ elements. By using such compressed matrices it is possible to perform matrix-matrix multiplication, LU decomposition, and eigenvalue computation in near-linear time. They are most commonly used in boundary integral problems where the matrix to be solved is dense. Hierarchical matrices can also be applied to Schur complements that arise in sparse direct solvers, so their applicability extends to fluid, structure, and electromagnetic simulations. However, these hierarchical algorithms are rather new and highly optimized implementations do not exist at the moment. A highly optimized distributed memory GPU implementation is needed to extract the potential parallelism of these methods.

1. Basic Information

(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

(2) Research Areas

- ☑ Very large-scale numerical computation
- ☐ Very large-scale data processing
- ☐ Very large capacity network technology
- ☐ Very large-scale information systems

(3) Roles of Project Members

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

Ichitaro Yamazaki (University of Tennessee)
Development of distributed memory runtime –ParSEC and blocked BLAS library for GPU –block MAGMA

Akihiro Ida (University of Tokyo)
Feature extension of hybrid MPI/OpenMP 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 H-matrix-vector product

Takayuki Aoki (Tokyo Institute of Technology)
Application of HACApK to Poisson solvers for multiphase flows

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

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

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

Jack Dongarra (University of Tennessee)
Development of distributed memory runtime –ParSEC and blocked BLAS library for GPU – block MAGMA

2. Purpose and Significance of the Research

H-matrices can reduce the arithmetic complexity of dense matrix-multiplication and factorization from $O(N^3)$ to $O(N\log^2 N)$ but still attain high Flop/s on GPUs by making use of batched BLAS operations. Conventional fast algorithms with low arithmetic complexity such as FFT, sparse linear algebra, and multigrid methods have low arithmetic intensity and are memory-bound on most modern architectures. Conversely, methods with high arithmetic intensity like dense linear algebra and N-body methods can remain compute-bound on modern architectures, but tend to have a high arithmetic complexity and waste many Flop/s. H-matrices have a rare combination of high arithmetic intensity and low arithmetic complexity, which makes them an interesting alternative to many existing algorithms on future architectures.

H-matrices were initially applied to boundary integral problems in electromagnetics, seismic, and fluid simulations. However, H-matrices have recently been growing in popularity in new fields that can benefit from approximate dense linear algebra operations such as machine learning and statistics/big data. The broad applicability of H-matrices makes it a worthwhile algorithm to heavily optimize on many-core and accelerator architectures.

One of the goals of this project is to facilitate the transition to the algorithm of the future, by providing a highly optimized H-matrix library that users can simply call from their existing framework. An important aspect of this approach is that our code will be optimized on CPU, GPU, and Xeon Phi, which represents the range of architectures for JHPCN platforms during the coming years.

3. Significance as a 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 H-matrix library. R. Yokota is the developer of exaFMM, which is a highly scalable and GPU equipped FMM code have the same data structure as an HLRA code. A. Ida and T. Iwashita are developers of HACApK – a hybrid MPI-OpenMP implementation of the HLRA. T. Hiraishi has experience in load-balancing for distributed memory H-matrix codes. I. Yamazaki and J. Dongarra are developers of dense linear algebra libraries such as MAGMA and PLASMA. T. Aoki has expertise in large scale fluid dynamics simulations that make use of distributed memory and GPUs. S. Oshima has expertise in tuning solvers for GPUs and Xeon Phi. K. Nakajima has expertise in parallel preconditioned iterative solvers and their application in CFD with AMR. The combination of these expertise is necessary for achieving the goals mentioned above.

Furthermore, each member already has highly optimized code for each component, which gives us an advantage over other groups that are writing an H-matrix code

from scratch. There are a few existing H-matrix implementations, but they are limited to shared memory and have not been ported to GPUs. To our knowledge, HACApK is the only multi-GPU H-matrix code available at the moment. This could only have been done through a JHPCN international collaboration between the experts in each area.

4. Outline of the Research Achievements up to FY2016

Our goal is to have a GPU implementation of our H-matrix code, and extend it to LU factorizations and use it as a preconditioner. To this end, understanding the relation between H-matrices and FMM is critical, because FMM is known to perform well on GPUs and have recently possessed the ability to be used not only as a mat-vec, but also a preconditioner [4]. Therefore, during the first half of FY2016 we focused on understanding the relation between H-matrices and FMM, because this is the shortest path to achieving our objective.

FMM and H-matrices lie at the opposite ends of the spectrum of hierarchical low-rank approximation methods , which are shown in Figure 1. ``Compressed operators" in Figure 1 represents a method which recompresses the FMM translation matrix by using SVD. This method was originally designed to accelerate the translation of multipole expansion in FMM. When this technique is used the FMM becomes very similar to a H^2-matrix or HSS matrix.

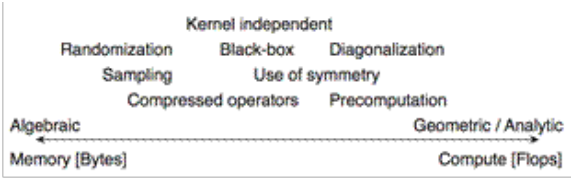


Figure 1. Compute-memory tradeoff in hierarchical low-rank approximation method

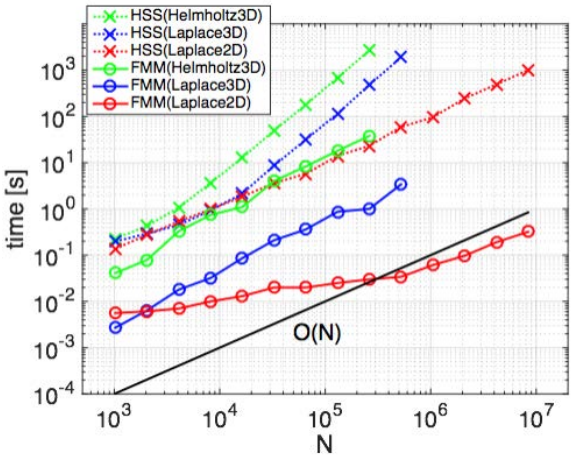


Figure 2 Calculation time for a single matrix-vector multiplication including setup time for the Green's function matrix of a 2-D Laplace, 3-D Laplace, and 3-D Helmholtz equation

Therefore, we perform a direct comparison between FMM and HSS. We use the 2-D Laplace, 3-D Laplace, and 3-D Helmholtz equations as the problem of interest, and generate the matrices from the Green's function. A single core of a 12 core Ivy Bridge (E5-2695v2) is used for the calculations. In Figure 2, we show the calculation time for the FMM and HSS. The x-axis is the size of the matrix, and the y-axis is the calculation time in seconds. FMM has a constant overhead so for small N it does not show O(N) behavior. For sufficiently large N, both FMM and HSS show O(N) behavior. It can be seen that FMM is about 1000 times faster than HSS. This is due to the large calculation cost of the algebraic compression that HSS does, whereas the FMM does not.

Another important aspect of our work in the first half of FY2016 is the load-balancing of the distributed memory implementation. The decomposition of a hierarchical matrix is shown in Figure 3. In the present work, we improve the load-balance by introducing a

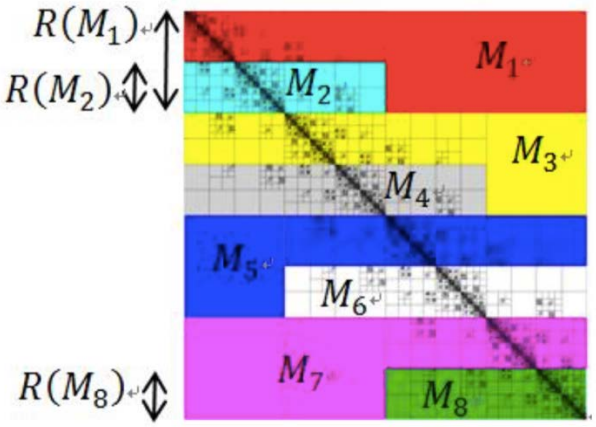


Figure 3. Domain decomposition of H-matrix

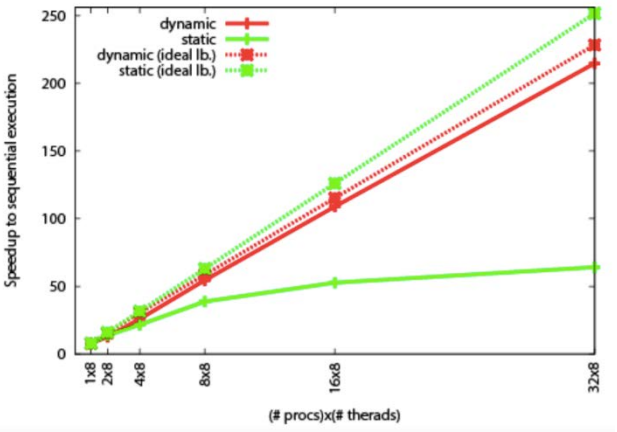


Figure 4. Parallel speedup of H-matrix

dynamic load-balancing scheme that predicts the ranks of each block. We use a test problem that gave poor load-balance for a static load-balancing scheme, to validate our new dynamic load-balancing scheme. We use dynamic task scheduling in OpenMP. The solid line in Figure 4 is the result when we use dynamic load-balancing. We can see that all threads are calculating similar number of elements.

For improving the scalability of the distributed memory H-matrix codes, we considered the possibility of importing various techniques from FMM. We have studies the communication patterns of FMM

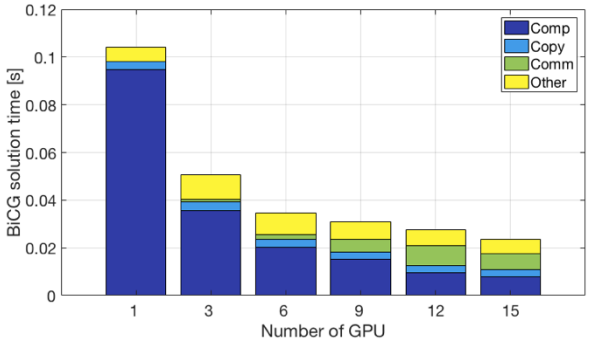


Figure 5. Performance of BiCGSTAB using HACApK for the mat-vec on multiple GPUs.

extensively and have constructed a performance model that predicts the behavior on the largest supercomputers such as Mira, Titan, and Shaheen. This performance model was used to construct an asymptotically superior communication scheme for FMM in FY2017 [13]. The communication pattern of H-matrices are identical to FMM so these techniques should be directly applicable to these algebraic variants of FMM.

The GPU implementation of HACApK was achieved through the use of MAGMA. Since one of the developers of MAGMA – Ichitaro Yamazaki was a collaborator in this project, the integration of MAGMA with HACApK was done in a very short amount of time. We were therefore able to have a multi-GPU version by the end of FY2016.

The performance of HACApK on multi-GPU is shown in Figure 5, where ``Comp" is the computational kernels, ``Copy" is the CUDA memory copy time, ``Comm" is the MPI communication. The MPI communication time eventually becomes the bottleneck because the mat-vec computation part takes very little time on the GPU.

5. Details of FY2017 Research Achievements

5.1 H-matrix-vector multiplication on multi-GPUs

During FY2017, we have made significant progress in the use of batched MAGMA with HACApK, and its multi-GPU implementation. We have validated its performance by applying it to the matrix-vector multiplication in a BiCGSTAB solver.

Though the low-rank compression reduces the cost of the matrix multiply, in many cases, the BiCG's iteration time is still dominated by this Hierarchical Matrix Vector multiply (HMVM). To reduce the iteration time using a distributed-memory computer, HACApK distributes the contiguous, but not disjoint, rows of the matrix among the processes (see Figure 3 for an illustration). Then, each process performs HMVM with its local submatrix. With this parallelization scheme, the only required inter-process communication is the all-gather needed after HMVM to form the global vector on each process (using MPI_Allgatherv).

This parallelization scheme is motivated by two performance properties of the solver: 1) the BiCG's computation time is dominated by HMVM, while the time needed for the remaining vector operations is insignificant in the computation time and 2) the redundant computation of the vector operations avoids the global all-reduces needed to compute the six dot-products for each BICG iteration, which can be much more expensive compared with the arithmetic operations. Hence, by redundantly performing the vector operations, this parallelization scheme aims to balance out two conflicting performance factors: distributing the computation with a minimum inter-process communication.

We conducted all the experiments in double precision, and used the matrices from electrostatic field simulations with perfect conductors of two particular shapes:

· Sphere: pairs of perfect conductors with the shape of a sphere. For each pair, one sphere has its electric potential set to be 1 Volt, while the other has the electric potential of −1 Volt. We use the boundary value of 0 Volt at infinity and analyze the induced electrical charge on the surface of the spheres.

· Human: perfect conductors with the shape of a humanoid who is standing on a uniform 2D grid on a uniform electric field. The surface of the humanoid is divided into 2, 359, 680 triangular elements and the induced electrical charge on the humanoid's surface was calculated using an indirect BEM with a single layer potential formulation and step functions as the base function for the BEM.

Table 1 lists our test matrices. The large-scale matrices 8ms and 20ms were used for the inner-iteration to precondition the linear system.

Table 1. Test matrices where "compression%" is the ratio of the total number of numerical values in the compressed matrix over $n\log_2(n)$.

	Sphere objects			
name	100ts	288ts	338ts	1ms
size, n	101,250	288,000	338,000	1,004,400
compress%	16.0	16.7	16.9	17.6

	Sphere objects (precond)		Human objects		
name	8ms	20ms	hum2	hum4	hum6
size, n	7,996,800	20,736,000	78,656	314,624	707,904
compress%	1.7	1.7	17.3	22.9	31.7

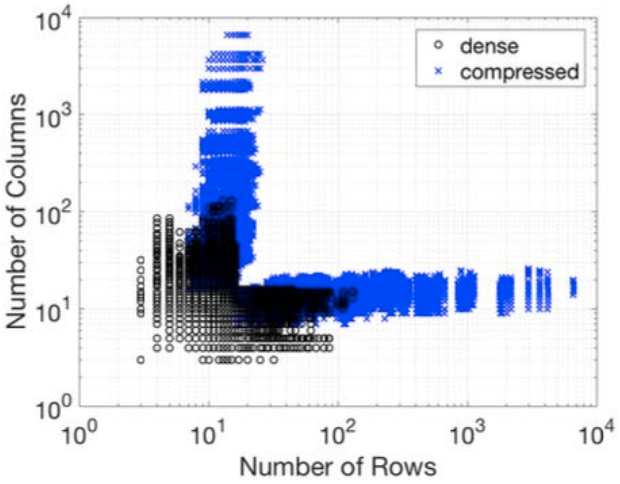


Figure 6. Block sizes in test matrix ``100ts''

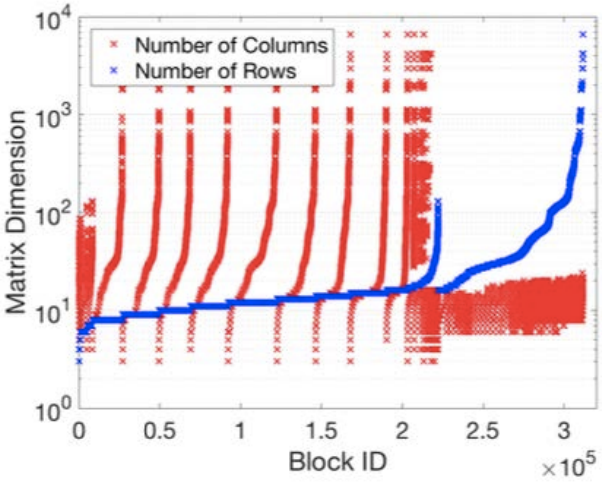


Figure 7. Matrix 100ts sorted by the number of rows

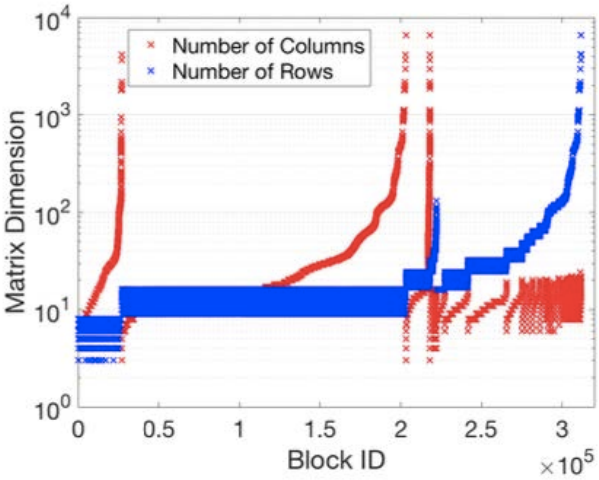


Figure 8. Matrix 100ts sorted by number of rows, and then by number of columns within group.

All of the compressed blocks have rank one and we fixed the number of inner iterations to be 20 for our experiments. Figure 6 shows the sizes of the blocks in the matrix 100ts. We see a wide range of the block sizes where all the blocks on the diagonal are square and dense, while off-diagonal blocks can be either dense or compressed and are either tall-skinny or wide-short. To utilize the GPU, each process divides its local dgemv tasks into several batches (e.g., a batch with a fixed number of dgemv's), and then calls dgemv_vbatched for each batch. dgemv_vbatched can execute dgemv's with variable matrix sizes in a single kernel launch. However, the performance of dgemv_ vbatched can be much lower than its fixed-size counterpart. This is especially true when there is a wide range of matrix sizes in the single batch. In order to improve the performance of dgemv_vbatched, we examined several schemes to sort the blocks of A, on which dgemv's operate.

In Figure 7, the blocks were sorted in the ascending order of their numbers of rows, and in Figure 8, we first grouped the blocks according to the number of rows (the k-th group contains the block with the number of rows in the range between $8(k-1)+1$ and $8k$), and then we order the blocks in the same group according to their numbers of columns.

Figures 9 and 10 show the effects of these two sorting schemes on the kernel performance for the matrix 100ts. The figure also shows the performance of the fixed-size batched kernel for each block size, which we consider as the upper bound on the performance of the variable-size kernel. We clearly see that the performance can be

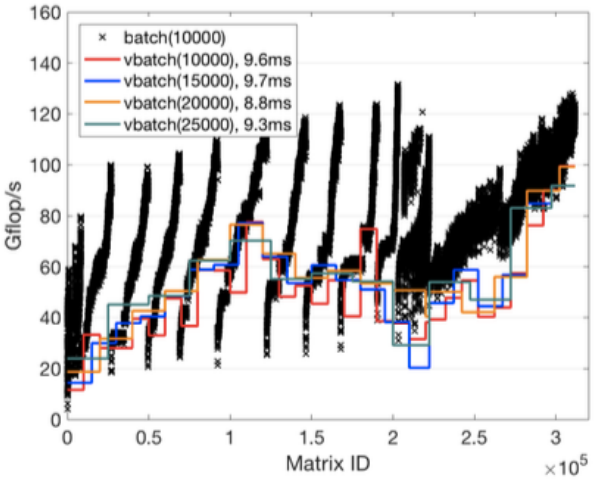


Figure 9. Sorting scheme of Figure 7 with fixed batch count.

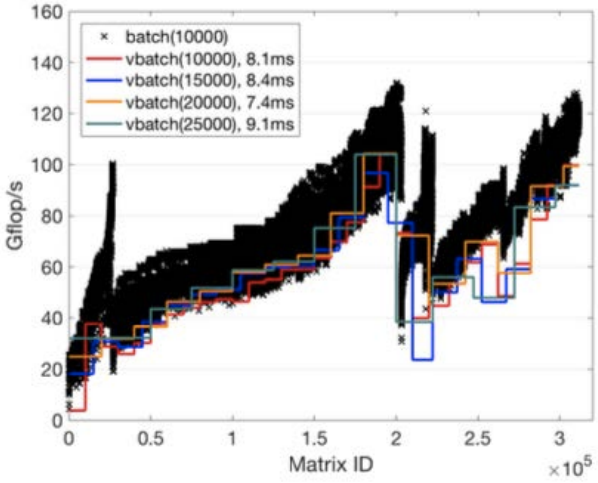
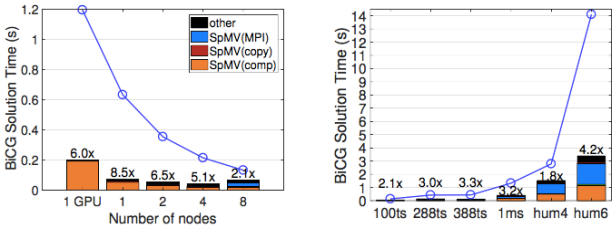


Figure 10. Sorting scheme of Figure 8 with fixed batch count.

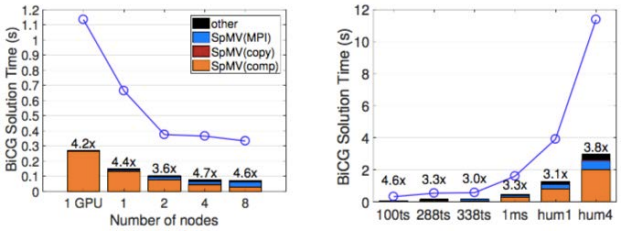
significantly improved by properly sorting the blocks (speedups of up to 2.5×) and the variable size kernel may obtain the performance closer to that of the fixed-size kernel.

Figures 11 and 12 show the effects of the GPU kernels on the BiCG performance on Tsubame-3 and Reedbush-H, respectively. For the performance without the GPUs, we bind each process to a socket and launch one OpenMP thread on each of the available cores of the socket. We found this process/thread configuration typically gives the best



(a) Strong scaling 100ts (b) On 8 nodes

Figure 11. Performance on Tsubame-3. The blue markers show the solution time with original HACApK without GPUs, while the bars are with the GPUs.



(a) Strong scaling 100ts (b) On 8 nodes

Figure 12. Performance on Reedbush-H. The blue markers show the solution time with original HACApK without GPUs, while the bars are with the GPUs.

performance of the hybrid MPI/OpenMP implementation. With the GPUs, we launch one process per GPU (i.e., four or two processes per node on Tsubame-3 or Reedbush-H). The figures clearly show that the GPUs have reduced the iteration time significantly, obtaining the speedups of about 4.2× and 4.5× on eight nodes of Tsubame-3 and Reedbush-H, respectively (in Figures 11(b) and 12(b)).

5.2 H-matrix LU-decomposition on distributed memory

We propose a novel method to parallelize the factorization of a hierarchical low-rank matrix (H-matrix) on distributed memory computers. This problem is much more difficult than the parallelization of a dense matrix factorization due to the hierarchical

block structure of the matrix, and it is much more difficult than the H-matrix vector multiplication due to the dataflow of the factorization. Getting rid of the hierarchy, the Block Low Rank (BLR) format not only simplifies the parallelization, but also increases concurrency. However, this comes at a price of loosing the near linear complexity of the H-matrix factorization. In the present work, we propose a "lattice H-matrix" factorization that combines the parallel scalability of BLR with the near linear complexity of H-matrix. To the extent of our knowledge, there have been no such attempts to balance the complexity and concurrency of two structured low-rank approximation methods.

We can generate any of the low-rank structures shown in Figure 13 by controlling the admissibility condition and the branch truncation of the block cluster tree during the construction of the H-matrix. Hence, all structured low-rank matrices can be regarded as special types of the H-matrices, as illustrated in Figure 13 for a given problem.

Each partitioning structure has different pros and cons. For instance, the H-matrix is the most general low-rank matrix structure, and it is an effective way of compressing the matrix with small numerical ranks. However, it usually has a complicated partitioning structure as shown in Figure 13(a), which makes it challenging to perform some matrix operations on a distributed-memory computer (e.g. LU factorization).

To improve the parallel scalability of the matrix operations, simpler partitioning structures have been proposed. For instance, by setting a weak admissibility condition, we can construct the partitioning structure

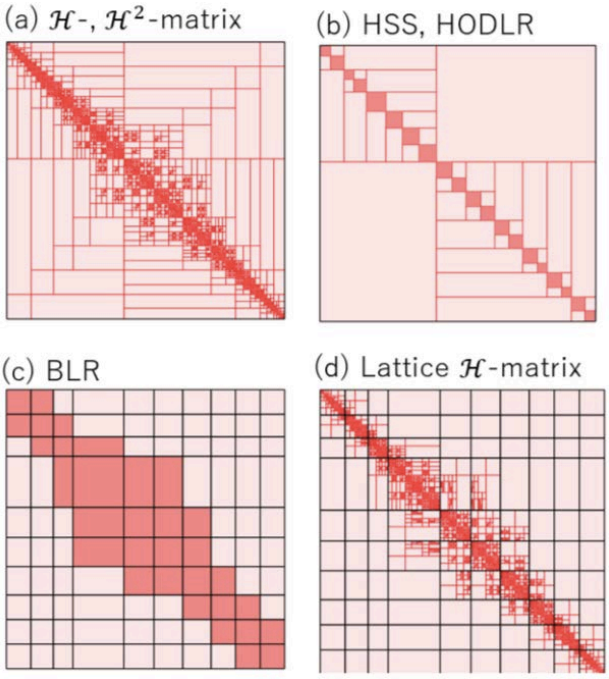


Figure 13 Illustration of popular low-rank structures generated for the same problem: (a) general H-matrix and its conversion to (b) Hierarchically Off-Diagonal Low-Rank (HODLR) layout, (c) Block Low-Rank (BLR) layout, and (d) lattice H-matrix layout. Blocks painted in deep red show dense submatrices, and blocks in light red indicate low-rank submatrices.

shown in Figure 13(b), which can be seen in the Hierarchical Semi-Separable (HSS) matrix or in the Hierarchically Off-Diagonal Low-Rank (HODLR) matrix. Compared with the H-matrix structure, this structure is simpler and more convenient for performing certain matrix operations on distributed-memory. However, this matrix structure assumes a weak admissibility condition, where all off-diagonal blocks are assumed to be low-rank. When the weak admissibility condition is applied to a 3D or higher dimensional problem, this structure leads to a higher asymptotic complexity due to the larger ranks of off-diagonal blocks.

Block Low Rank (BLR), shown in Figure 13(c), is another simpler matrix structure. Though the construction of the BLR matrix does not require the block cluster tree used to construct the H-matrix, it can be constructed by truncating all branches of the cluster tree at a certain depth level. The partitioning structure of BLR is then given by the induced blocks in the block cluster tree. The admissible condition is verified on each block after the structure is defined. The memory complexity of the BLR matrix is $O(n^{1.5})$ and is higher than O(nlogn) of the H-matrix. However, the BLR matrix is a simple, nonhierarchical, and effective low-rank layout, especially for the distributed-memory. In particular, the BLR structure has the block layout similar to the 2D block layout used in many dense matrix operations, and it can use many of the high-performance optimization techniques developed for the dense matrix operations.

Figure 13(d) shows the partitioning structures of the lattice H-matrix that we use in this paper. It combines the structures of BLR with the H-matrix by introducing the lattice structure on top of the H-matrix. In other words, the lattice H-matrix utilizes the H-matrix format for each block of the BLR matrix. These lattices are then distributed among the processes in a 2D block cyclic fashion. It is designed to balance the advantages of the H and BLR matrices: the high compressibility of the H- matrix, which reduces the memory and computational costs, and the parallel scalability of the BLR matrix. Using the lattice H-matrix, the complex matrix operations originating from the H-matrix structure are performed using the threaded computational kernels.

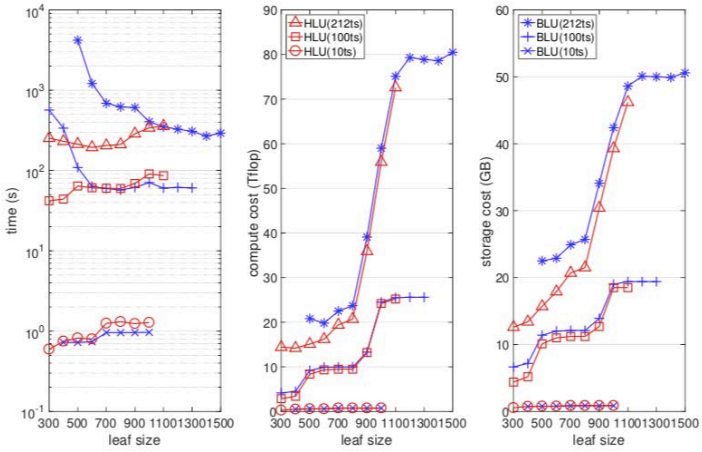
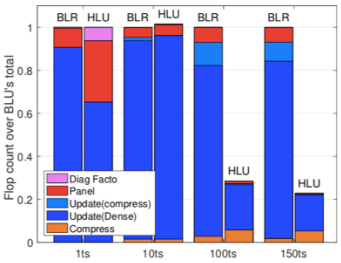


Figure 14 Effects of the leaf sizes on the factorization time, and the computational and storage costs of the factorization.

Taken a large enough lattice size, the lattice H-matrix can reduce the memory complexity from $O(n^{1.5})$ of the BLR matrix to O(nlogn) of the H-matrix. At the same time, the lattice matrix has the same communication pattern as the BLR matrix, and it can utilize the high-performance parallel algorithms and the efficient communication schemes developed for the dense matrix operations.

We conducted our experiments on the on the Reedbush-L supercomputer at the University of Tokyo, and the TSUBAME3.0 supercomputer at the Tokyo Institute of Technology, and also the Edison super-computer at the National Energy Research Scientific Computing Center (NERSC). We used Intel MPI compiler mpiifort and mpiicpc of version 18.1.163. On all systems, the code was compiled using the -O3 optimization flag, and linked to sequential MKL version 2018.1.163.

We first study the performance of the threaded BLR LU (BLU) and threaded H-matrix LU (HLU) factorization on the shared-memory CPUs. One of the critical parameters that affect the factorization performance is



(a) Breakdown of flop count.

# of thread	1ts	10ts	100ts
1	0.06	16.2	892
6	0.03	3.0	172
12	0.02	1.6	94
24	0.02	0.9	63

(b) Thread scalability of BLU.

# of thread	1ts	10ts	100ts
1	0.07	14.4	426
6	0.05	2.8	82
12	0.06	1.5	45
24	0.06	1.0	32

(c) Thread scalability of HLU.

Figure 15 Performance of the shared-memory factorization. Left graph shows the breakdown of the flop counts needed for different phases of factorization (diagonal factorization, computation of off-diagonal tiles in the panel, updating either compressed or dense blocks, and ACA), relative to the total flop count for BLR. Right tables shows the factorization time in seconds using different numbers of threads on one node.

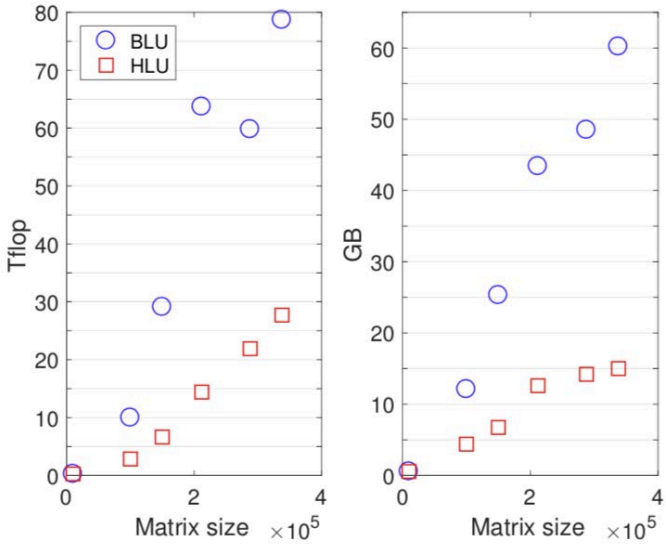


Figure 16 Computational and storage costs with varying matrix sizes. The low-rank compression greatly reduces the costs of the factorization, e.g., for the matrices in this figure, the dense factorization would require the computational costs of 0.7, 667, 2250, 6352, and 25743 Tflops, and the storage costs of 0.8, 80, 180, 360, and 914 GB.

the leaf size. Hence, we first seek for an optimal leaf size that obtains the shortest factorization time. Figure 14 shows the effects of the leaf size on the factorization performance. A larger leaf size tends to increase both the computational and storage costs of the BLU factorization, but it also improves the performance of the BLAS and LAPACK subroutines used for the local computation. In many cases, BLU obtained the fastest factorization time using the leaf size that is larger than that obtained the minimum storage or computation. The optimal leaf size also tends to increase with the increase in the matrix dimension. In contrast, HLU's factorization time was less sensitive to the leaf size. Overall, for our test matrices, the leaf size of $O(\sqrt{kn})$ with k = 5 was a good choice for BLU, and that is what we use for the rest of the experiments.

For HLU, we use the fixed leaf size of 300. With these choices of the leaf sizes that obtain the near-optimal performance of each algorithm, BLU needed a much larger leaf size and had much higher storage and computational costs than HLU, especially for a large matrix.

Figure 15(a) shows the relative flop counts for different phases of the factorization. We see that the flop count is dominated by the trailing submatrix update that is well suited for the parallelization. Figure 15(b) and 15(c) show the resulting thread scalability of the two factorization algorithms. Since we used a fixed leaf size for HLU, its scalability was lower than BLU's for a small matrix. For instance, for the matrix 1ts, there were only three diagonal blocks for HLU. On the other hand, for a large matrix, HLU was often

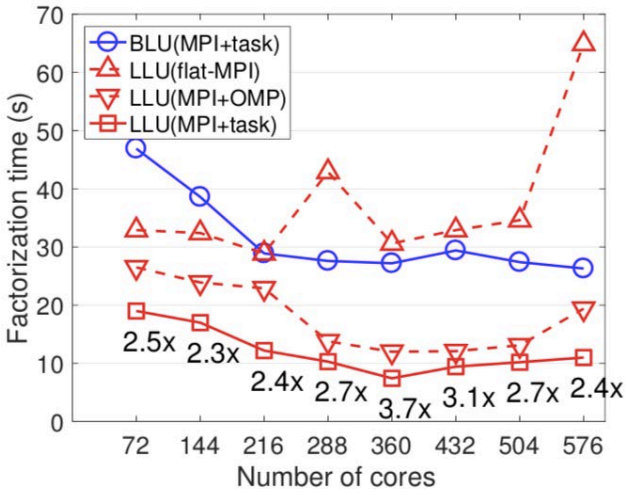
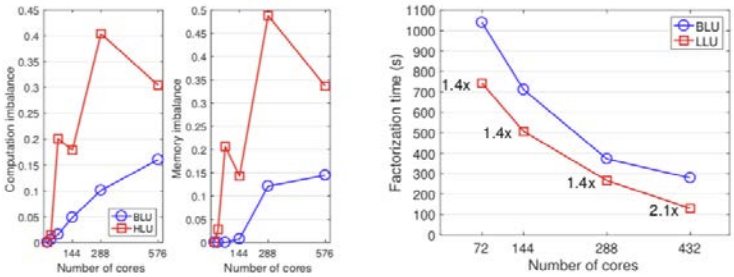


Figure 17 Effects of MPI/OpenMP configurations on the factorization time for the matrix 100ts with 18 threads per process on Reedbush-L.



(a) Imbalance. (b) Factorization time.

Figure 18 Strong parallel scaling for 332ts with 18 threads per process on Reedbush-L. The imbalance is the ratio of the difference between the maximum and the minimum loads among the processes over the average load.

faster than BLU due to HLU's lower computational cost.

Figure 16 visualizes the computational and storage costs of BLU and HLU for varying matrix dimension. The difference between the costs of the two algorithms tends to increase with the increase in the matrix dimension. The cost of the new lattice LU is between those of BLU and HLU depending on the lattice size used.

We now compare the performance of BLU and the new LLU (lattice LU) on the distributed-memory computer. We first study the effects of the lattice size with the increasing number of processes (e.g., the storage or computational cost per process). For LLU, we used the fixed leaf size used for HLU (i.e., 300). As we reduce the lattice size, the H-matrix is split into smaller lattices, becoming closer to BLR. The factorization costs did not significantly change using different leaf sizes.

Figure 17 shows the performance of LLU using three different MPI/OpenMP configurations: i.e., 1) flat-MPI with one process per core, 2) MPI+OpenMP with one process per socket and one thread per core but with a synchronization among the local threads before each phase of factorization, and 3) MPI+OpenMP tasks. We see that the hybrid MPI/OpenMP programming often reduces the cost of inter-process communication, performing better than the flat-MPI. The performance can be further improved using tasks that avoid the artificial synchronizations and obtain a better core utilization.

To accommodate the large storage costs of BLU, we conducted the remaining experiments on Reedbush-L. Figure 16(a) shows the load imbalance among the processes for computing BLU and HLU. Since HLU's lattice size is larger than BLU's tile size, HLU had a greater load imbalance, especially with a larger process count. One of our motivations for using OpenMP task is to reduce the effects of the load imbalance on the parallel scalability of HLU or BLU.

6. Progress of FY2017 and Future Prospects

6.1 Progress of FY2017 including self-evaluation with respect to the research plan

The two main goals of the fiscal year 2017 are 1) Implement the GPU version of HACApK using block MAGMA, which can calculate streams of small matrices efficiently on GPUs, 2) Extend HACApK to perform a LU decomposition of a hierarchical version of H-matrices instead of the current non-hierarchical version. We have achieved our goals with satisfactory performance, where the former has been accepted to IPDPS'18 [5,6] and the latter has been submitted to SC'18.

For item 1), we have made significant progress as shown in section 5.1. We ported the hierarchical-matrix BiCG solver onto GPU clusters [5], and investigated several techniques to improve the performance of the batched GPU kernel (sorting the tasks, dividing them into multiple batches of different batch sizes [9], and using GPU streams to execute multiple batches in parallel) [16]. We hope that these techniques will be integrated into the future generation of the batched kernel along with a runtime analysis and tuning. We have also studied several techniques to reduce the inter-GPU communication. As the heterogeneous node architecture becomes increasingly complex and the cost of the inter-GPU communication increases, these techniques likely become critical to many applications running on GPUs. We have observed a great potential of the sophisticated MPI communication strategies that complement our studies [12,20], and we plan to investigate its performance on different architectures (e.g., non-blocking collective on Summit). We would also like to investigate other algorithmic techniques to address the communication costs (e.g., 2D partitioning, empty off-diagonal blocks with rank zero). Though we ported only the linear solver onto the GPUs, we plan to port other parts of the simulation (e.g., generation or compression of the matrix). We have also extended HACApK to run on the Xeon Phi Knights Landing [7,17,25]. This port is more straightforward than that of the GPU [8,22], since it is merely changing the BLAS library to the AVX512 optimized version and compiling for the Knights Landing. We have also ported HACApK to run on FPGAs using OpenCL [18,19].

For item 2), we have proposed a novel method that hybridizes the BLR and H-matrix to achieve optimal balance between the complexity and concurrency on distributed memory architectures. The algorithm partitions the matrix into 2D H-submatrices, called lattices, and distributes the lattices in a 2D cyclic pattern. Compared with the standard H-matrix factorization, the lattice simplifies both the communication pattern and the parallel computation, while compared with the block low-rank (BLR) layout, the lattice reduces both the storage and computational costs. Our experimental results demonstrate that the lattice- based LU (LLU) can obtain significant speedups over BLR, while using a smaller storage.

We not only developed the H-matrix library in FY2017 but also applied it to various scientific problems including electro-magnetics [1,24], micromagnetics [3,26], which required the parallelization of BEM [2,13]. A new and exciting application that we have extended H-matrices to is machine

learning where we observed significant speedup by using low-rank approximations [10, 11, 14].

6.2 Future prospects

We are working to improve the parallel performance of LLU (e.g., accumulating multiple updates before compression, using task-priority for reducing idling time, and examining the potential of other runtime systems). Though the low-rank compression reduces the communication volume, the communication can still be the parallel performance bottleneck. We are looking to reduce this bottleneck by integrating other techniques (e.g., 2.5D factorization). The selection of the optimal lattice size is critical to obtain good performance of LLU, especially with an increase in the process count. We are studying theoretical or experimental way of picking a proper lattice size. Another critical parameter is the leaf size. For our experiment with LLU, we used the leaf size that obtained a good performance of HLU. We are examining the effects of the lattice size on the optimal leaf size for HLU's performance. Other future work including using the factors as a preconditioner and accelerating the factorization process using GPUs. Our focus is on HACApK that generates a particular matrix partition and compression, and we cannot directly use existing algebraic linear solver packages. However, we would like to extend our study by examining other solvers for the applications of our interests. In fact, our layered interface allows any partitioning structure to be imported into our solver, and our solver can be used as an algebraic solver.

7. List of Publications and Presentations

(1) Journal Papers

1. N. Tominaga, T. Mifune, A. Ida, Y. Sogabe, T. Iwashita, N. Amemiya, ``Application of hierarchical matrices to large-scale electromagnetic field analyses of coils wound with coated conductors", IEEE Transactions on Applied Superconductivity, Vol. 28, No. 3, pp.1-5 (2018).
2. T. Iwashita, A. Ida, T. Mifune, Y. Takahashi, ``Software Framework for Parallel BEM Analyses with H-matrices Using MPI and OpenMP", Procedia Computer Science 108C, pp.2200–2209 (2017).
3. A. Ida, T. Ataka, T. Mifune, Y. Takahashi, T. Iwashita, A. Furuya,``Application of Improved H-matrices in Micromagnetic Simulations", IEEE Transactions on Magnetics, Vol. 54, No. 3, 2018.
4. H. Ibeid, R. Yokota, J. Pestana, D. Keyes, ``Fast Multipole Preconditioners for Sparse Matrices Arising from Elliptic Equations", Computing and Visualization in Science, pp. 1—17, 2017. https://doi.org/10.1007/s00791-017-0287-5

(2) Conference Papers

5. I. Yamazaki, A. Abdelfattah, A. Ida, S. Ohshima, S. Tomov, R. Yokota, J. Dongarra, ``Analyzing Performance of BiCGStab with Hierarchical Matrix on GPU clusters," 32nd IEEE International Parallel & Distributed Processing Symposium, IPDPS2018, Vancouer, Canada, 21-25 May (2018).
6. A. Ida, ``Lattice H-Matrices on Distributed-Memory Systems", 32nd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2018), Vancouver, Canada, 21-25 May (2018).
7. *T. Hoshino*, Akihiro Ida, *T. Hanawa*, K. Nakajima, ``Design of Parallel BEM Analyses

Framework for SIMD Processors", The International Conference on Computational Science 2018 (ICCS 2018), Wuxi, China, 11-13 Jun. (2018).

8. S. Oshima, I. Yamazaki, A. Ida, R. Yokota, `` Optimization of Hierarchical Matrix Computation on GPU", Proceedings of the 4th Asian Conference, Lecture Notes in Computer Science, Vol. 10776, pp. 274-292, Singapore, 26-29 Mar. (2018).
9. A. Ida, *H. Nakashima*, M. Kawai, ``Parallel Hierarchical Matrices with Block Low-rank Representation on Distributed Memory Computer Systems" International Conference on High Performance Computing in Asia-Pacific Region, Tokyo, Japan, 28-31, Jan. (2018)
10. K. Oosawa, R. Yokota, ``Evaluating the Compression Efficiency of the Filters in Convolutional Neural Networks ", Proceedings of the 26th International Conference on Artificial Neural Networks, Sardinia, Italy, 11-14 Sep. (2017).
11. K. Oosawa, A. Sekiya, H. Naganuma, R. Yokota, "Accelerating Matrix Multiplication in Deep Learning by Using Low-Rank Approximation", Proceedings of the 2017 International Conference on High Performance Computing & Simulation, Genoa, Italy, 17-21 Jul. (2017).
12. M. AbdulJabbar, G. Markomanolis, H. Ibeid, R. Yokota, D. Keyes, ``Communication Reducing Algorithms for Distributed Heirarchical N-Body Methods", ISC High Performance, Lecture Notes in Computer Science, Vol. 10266, pp. 79–96, Frankfurt, Germany, 18-22, Jun. (2017).
13. T. Iwashita, A. Ida, T. Mifune, Y. Takahashi, ``Software Framework for Parallel BEM Analyses with H-matrices Using MPI and OpenMP", Tools for Program Development and Analysis in Computational Science, ICCS2017, Zürich, Switzerland, 12-14 Jun. (2017).
14. K. Oosawa , A. Sekiya , H. Naganuma , R. Yokota, ``Accelerating Convolutional Neural Networks Using Low-Rank Approximation", Proceedings of the 22nd Conference of Japan Computational Engineering Society, 31 May – 2 Jun. (2017).

(3) Oral Presentations

15. A. Ida, ``Low Rank Approximation Methods Used in Hierarchical Matrices", ATAT in HPC 2018, National Cheng Kung University, Tainan, Taiwan, 27 Mar. (2018)
16. A. Ida, ``Efficient Low-rank Solver for Integral Equations on Distributed Memory Systems" SIAM Conference on Parallel Processing for Scientific Computing, Tokyo, Japan 7-10 Mar. (2018).
17. *T. Hoshino*, A. Ida, *T. Hanawa*, ``Performance Evaluations and Optimizations of H-Matrices for Many-Core Processors", SIAM Conference on Parallel Processing for Scientific Computing, Tokyo, Japan 7-10 Mar. (2018).
18. *T. Hanawa*, A. Ida, *T. Hoshino*, ``Hierarchical Matrix Operations on FPGA Using OpenCL", The 163rd HPC Workshop, Ehime, Japan, 2 Mar. (2018).
19. *T. Hanawa*, A. Ida, *T. Hoshino*, ``Implementation of Hierarchical Matrix Operations on FPGAs", The 16th HPC Research Presentation, Hakodate, Japan, 20 Sep. (2017).
20. M. AbdulJabbar, M. Al Farhan, R. Yokota, D. Keyes, ``Performance Evaluation of Computation and Communication Kernels of the Fast Multipole Method on Intel Manycore Architecture", 23rd EUROPAR, Galicia, Spain, 29 Aug. - 1 Sept. (2017).
21. *T. Katagiri*, M. Yang, W. Wang, A. Ida,

``Performance Evaluation of Integration of Multiple Randomized Low-Rank Singular Value Decompositions", Summer United Workshops on Parallel, Distributed and Cooperative Processing, Akita, Japan, 26-28 Jul. (2017).

22. S. Ohshima , I. Yamazaki , A. Ida , R. Yokota, ``Optimization of Hierarchical Matrix Computations on a Cluster of GPUs", Summer United Workshops on Parallel, Distributed and Cooperative Processing, Akita, Japan, 26-28 Jul. (2017).
23. R. Yokota, ``Hierarchical Low-Rank Approximations at Extreme Scale", 32nd International Conference, ISC High Performance, Frankfurt, Gemany, 18-22 Jun. (2017).

(4) Others

24. N. Tominaga, T. Mifune, A. Ida, Y. Sogabe, T. Iwashita, N. Amemiya, ``Application of hierarchical matrices to large-scale electromagnetic field analyses of coils wound with coated conductors", 25th International Conference on Magnet Technology (MT25), Amsterdam, Netherlands, 31 Aug. (2017).
25. *T. Hoshino*, S. Ohoshima, *T. Hanawa*, K. Nakajima, A. Ida, ``Pascal vs KNL : Performance Evaluation with ICCG Solver", ISC17 Poster, Frankfurt, Germany (2017).
26. A. Ida, T. Ataka, T. Mifune, Y. Takahashi, T. Iwashita, A. Furuya, ``Application of Improved H-matrices in Micromagnetic Simulations", 21st International Conference on the Computation of Electromagnetic Fields (Compumag 2017), Daejeon, Korea(20th June 2017).