jh200051-NAHI

Scalable Multigrid Poisson solver for AMR-based CFD applications in Nuclear Engineering

Naoyuki Onodera（Japan Atomic Energy Agency）

We develop a multigrid preconditioned conjugate gradient (MG-CG) solver for the pressure Poisson equation in a two-phase flow CFD code JUPITER. The MG-CG solver is redesigned to realize efficient CFD simulations including complex boundaries and objects using an adaptive mesh refinement (AMR) method based on a block-structured Cartesian grid system. Here, we propose a new MG preconditioner with the cache reuse successive over relaxation (CR-SOR) smoother, which has high arithmetic intensity using the shared memory and enables continuous memory access on the block-structured grid. We measured the kernel performance of the MG-CG solver on GPU and CPU on the TSUBAME supercomputer, and A64FX on the FLOW supercomputer. The performances on the GPU and CPU are reasonable, but there are significant performance degradation on A64FX. The numerical experiments are conducted for two-phase flows in a fuel bundle of a nuclear reactor. Thanks to the block-structured data format, grids inside fuel pins are removed without performance degradation, and the total number of grids is reduced to 2.26×10^9, which is about 70% of the original uniform Cartesian grid system. The MG-CG solver with the CR-SOR smoother reduces the number of iterations to less than 9% of the original preconditioned CG method, leading to 5.9-times speedup on the TSUBAME supercomputer. In the strong scaling test, the MG-CG solver with the CR-SOR smoother is accelerated by 2.1 times between 64 and 256 GPUs.

1. Basic Information

(1) Collaborating JHPCN Centers

- The University of Tokyo
- Tokyo Institute of Technology
- Nagoya University

(2) Research Areas

■ Very large-scale numerical computation

(3) Roles of Project Members

- Naoyuki Onodera (JAEA, Representative, Development of a Poisson solver)
- Audit Edouard (CEA France, Deputy-Representative, Advice and support)
- Takayuki Aoki (Tokyo Tech, Deputy-Representative, Advice and support)
- Yasuhiro Idomura (JAEA, Advice and support for large-scale computation on TSUBAME)
- Yuta Hasegawa (JAEA, Development of a Poisson solver)
- Susumu Yamashita (JAEA, Development of a nuclear engineering code)
- Yuichi Asahi (JAEA, Porting codes on CPU/GPU platforms)
- Takashi Shimokawabe (Tokyo, Advice for large-scale computations on Oakbridge-CX)
- Satoshi Oshima (Nagoya, Advice for large-scale computations on Flow)

2. Purpose and Significance of Research

(1) Research purpose

This project aims at developing exascale computing technologies for multi-scale CFD simulations on exascale platforms. For this purpose, we promote a collaborative research with respect to the following three main subjects, a) High performance sparse matrix solvers on accelerated computing platforms, b) Adaptive Mesh Refinement (AMR) methods for multi-scale turbulence problems with complicated boundaries, c) Performance portability of CFD codes based on advanced frameworks and programming models.

The target CFD applications in this project are the

3D multi-phase multi-component thermal hydraulic CFD code JUPITER [Yamashita et al., Nuclear Engineering and Design 2017] and the 3D compressible hydrodynamic astrophysics CFD code ARK [Padioleau et al., Astrophysical Journal 2019]. JAEA has promoted the development of high-performance pressure Poisson solvers based on various Communication-Avoiding (CA) algorithms on the Oakforest-PACS [Idomura, ScalA18@SC18] and on the Summit [Ali, ScalA19@SC19] (Fig.1), and their strong scaling was demonstrated up to ~8,000 KNLs/V100s. In the FY2019 project, the development of a new AMR framework for JUPITER was conducted. On the other hand, CEA has developed ARK based on an AMR-based compressible fluid model, and its performance portability between KNL and V100 was demonstrated using a programing model with MPI+Kokkos. ARK requires high performance matrix solvers for gravity Poisson solvers and implicit matrix solvers.

In this project, we accelerate the development of common exascale computing technologies for AMR-CFD simulations by sharing the knowledges on high performance matrix solvers, performance portability, and AMR methods. The developed technologies will contribute to various science and engineering CFD applications.

(2) Significance of research

a) High performance sparse matrix solvers on accelerated computing platforms

Sparse matrix solvers for the pressure Poisson equation, the gravity Poisson equation, and implicit time integration are critical bottlenecks for high performance CFD simulations. In this project, we extend our CA algorithms such as CA multi-grid (CA-MG) solvers and CA-Krylov solvers to block (or patch) structured AMR grid data, and apply them to various nuclear engineering problems and astrophysics problems in JUPITER and ARK, respectively. By analyzing required precision in each problem, we explore optimum mixed precision approaches, which were already implemented in

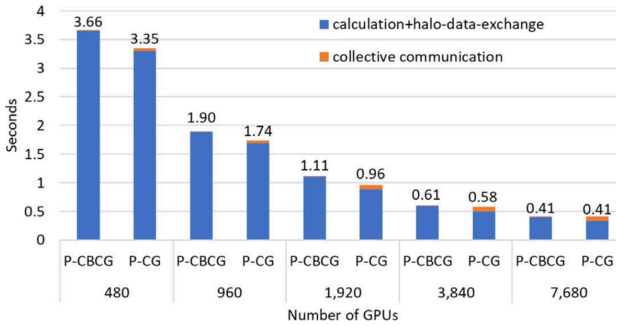


Fig.1: Strong scaling tests up to 7,680 GPUs on the Summit. A large JUPITER matrix with N = 1,280 × 1,280 × 4,608 is computed using the Preconditioned CG method and the Preconditioned Chebyshev Basis Communication-Avoiding CG method.

JUPITER. If such an efficient implementation works for AMR-CFD simulations, it has a great impact on various engineering and science fields.

b) AMR methods for multi-scale CFD problems

AMR methods are promising solutions for multi-scale turbulence problems. Their parallel implementations have been designed to keep the balance of computing loads, e.g., via space filling curves. However, on accelerated computing platforms, the communication cost often becomes comparable or exceeds the computing cost. In such a situation, one needs to design a new parallel implementation by taking account of both computation and communication costs. In this work, we develop a new performance model of AMR-CFD simulation, which is of critical importance for various exascale AMR applications.

c) Performance portability on multi-platforms

Performance portability between CPU and GPU has been a long-standing issue. In addition, several new CPU and GPU architectures are emerging towards exascale supercomputers. This makes the performance portability issue more serious. In this work, first, we port miniapps from JUPITER and ARM via native programing models such as CUDA and OpenMP and via more portable programing models such as Kokkos, and investigate performance bottlenecks in the latter approach. From this performance study, we clarify the most efficient porting strategy and apply it to JUPITER

and ARK.

3. Significance as JHPCN Joint Research Project

The goals in this project are to develop extreme scale AMR-CFD simulations and to establish performance portability on the latest CPU and GPU platforms. For this purpose, we need to use several supercomputing systems, which are based on state-of-the-art CPUs and GPUs and have a large number of computing nodes. Such computing needs can be satisfied only by a JHPCN joint research project.

4. Outline of Research Achievements up to FY2019

Not applicable.

5. Details of FY2020 Research Achievements

5.1 JUPITER Code

(1) Incompressible Two-Phase Flow Solver

We solve two phase flows using an incompressible fluid model based on the Navier-Stokes equation. The Navier-Stokes equation consists of advection, diffusion, and external force terms with an explicit time integration method. Since the divergence-free condition on the velocity field is essential for mass conservation, the pressure Poisson equation is solved iteratively. As a pressure-velocity correction algorithm, the simplified marker and cell (SMAC) method is applied to JUPITER. After solving the pressure Poisson equation, the velocity field is modified with the updated pressure gradient. The time evolutions of the above Navier-Stokes equation.

The dynamics of the gas and liquid phases are described by a convection equation of the volume of fluid (VOF) function. A physical property in a cell is determined by the linearly weighted average of the VOFs for the gas and liquid phases.

(2) Poisson Solver

The main computational cost comes from the pressure Poisson equation discretized by the second order accurate centered finite difference scheme with seven stencils. Since the matrix is symmetric and diagonal-dominant, the Poisson solver is normally computed using the P-CG method.

In two-phase flow simulations, a large density ratio between gas and liquid increases the contrast of each matrix component. In addition, large-scale simulations have multi-scale features such as thin gas-liquid interfaces and complex object boundaries. These features deteriorate the convergence property, and thus, the improvement of preconditioning is of critical importance.

(3) Geometric Multigrid Preconditioner

The MG method is one of the most efficient preconditioners to reduce the computational cost and improve the convergence property in multi-scale problems. Since JUPITER is based on the block-structured Cartesian grid system, we can use the geometric multigrid (GMG) method with a three-stage V-cycle shown in Fig. 2. In the GMG method, an approximate equation is solved at each MG level. The interpolation coefficients between MG levels are calculated geometrically.

The choice of the smoother at each MG level is very important in terms of the computational cost and the convergence property. The red and black SOR (RB-SOR) method is an efficient smoother, which has high thread parallelism. However, it requires multiple

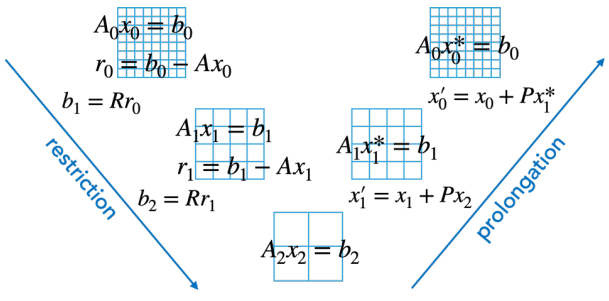


Fig.2: The geometric multigrid preconditioner based on a three-stage V-cycle with fine, middle, and coarse MG levels. The MG levels are defined within each leaf, in which the number of grids is reduced as 8^3, 4^3, and 2^3. A fixed iteration number is used for smoothers at each level.

memory accesses for the red and black phases alternatively to eliminate the dependency between adjacent grids. In this research, we propose a cache reuse SOR (CR-SOR) method, which can effectively reduce the residuals of local subdomain by multiple sub-iterations.

(4) Block-Structured Cartesian Grid

The original JUPITER code was developed based on a Cartesian grid system, whose resolution was chosen to reproduce the shape of the VOF function or the gas-liquid interface with enough accuracy. However, there was a lot of memory wastage for nuclear engineering problems which contain complex non-fluid objects. In the block-structured AMR grid system, grid resolution is adaptively changed in each subdomain depending on the local scale of multi-scale solutions. Even without the change of grid resolution, the block-structured grid is beneficial, when the boundary is not aligned to the Cartesian grid and/or complex structures exist inside the computational domain. In this work, we use the block-structured grid with uniform grid resolution and reduce the grids outside the boundary and inside the object.

A data structure is very important from the viewpoint of the computational efficiency on CPU/GPU platforms. A block-structured grid has an efficient data structure, in which the whole domain is decomposed into block units named "Leaf". Here, the leaves are connected by the forest-of-octree data structure. Since the leaf consists of a regular Cartesian grid system with n^3 grids, continuous memory access is enabled. In this research, the size of leaf is set to n=8 by taking account of the above tradeoff.

The block-structured grid can be easily extended to the MG method. It is possible to use the same forest-of-octree structure as long as there is a grid in the leaf. We adopt the MG method with a three-stage V-cycle, in which the number of grids at fine, middle, and coarse MG levels are set to 8^3, 4^3, and 2^3, respectively.

5.2 Implementation and Optimization

(1) Implementation of JUPITER Code on Block-Structured Cartesian Grid

JUPITER is written in C++ and CUDA. The connection of the block-structured grid is managed by the forest-of-octree data format, in which each array is based on the Structure of Array (SoA) memory layout. The same block-structure grid is used also in the MG method, where the offset index changes to 1/8 and 1/64 of the original resolution depending on the MG level.

It is important to assign CPU, GPU and A64FX threads to grids in stencil computation. We implement a kernel as shown in Fig. 3. Outer loop as "FOR_EACH1D_BLOCKIDX" corresponds to the leaves, and inner loop as "FOR_EACH3D" is assigned to 8^3 grids in the leaf. Although the block-structured Cartesian grid system keeps continuous memory access as in the standard Cartesian grid system, special care has to be taken in the treatment of the leaves. Stencil computation for the block-structured grid requires the offset indices of the neighbor leaves, which are normally assigned to the register memory. One leaf refers to the offset indices of 26 surrounding leaves, and these indices can be shared by threads belonging to the same leaf.

```
#ifdef __CUDA_ARCH__
  #define FOR_EACH1D_BLOCKIDX(L, NL) \
    const auto L = blockIdx.x

  #define FOR_EACH3D(I,J,K, NX,NY,NZ) \
    const auto I = threadIdx.x \
    const auto J = threadIdx.y \
    const auto K = threadIdx.z
#else
  #define FOR_EACH1D_BLOCKIDX(L, NL) \
    _Pragma("omp parallel for") \
    for(int L=0; L<NL; L++)

  #define FOR_EACH3D(I,J,K, NX,NY,NZ) \
    PRAGMA_FOR_SIMD \
    for(int K=0; K<NZ; K++) \
    PRAGMA_FOR_SIMD \
    for(int J=0; J<NY; J++) \
    PRAGMA_FOR_SIMD \
    for(int I=0; I<NX; I++)
#endif

void cfd_function(...)
{
  FOR_EACH1D_BLOCKIDX(l, num_amr_leaves){
    FOR_EACH3D(i,j,k,NX_LEAF,NX_LEAF,NX_LEAF){
      // operations at stencil (i,j,k, l)
    }
  }
}
```

Fig.3: Pseudocode for stencil computation to call CPU and GPU instructions.

(2) Cache Reuse SOR Method

The calculation speed of smoother is mainly limited by memory read/write time rather than the cost of floating-point operations. The RB-SOR method requires global memory access twice for the red and black phases, which causes performance degradation. To resolve this issue, we propose a new CR-SOR method [Onodera et al., HPC Asia 2021]. The computational domain is divided hierarchically in order to resolve dependencies between leaves and grids inside the leaf. One is coarse domain decomposition for MPI parallelization (MPI domain), and the other is fine block decomposition for cache optimization (cache block). In the outer iteration, each MPI domain is processed for *n* times with a two-color ordering. In the inner iteration, the RB-SOR method within each cache block is computed for *m* times to accelerate the convergence.

5.3 Numerical Experiment

(1) CG Kernel Performances on TSUBAME and FLOW

We evaluated the kernel performance of CG solvers in JUPITER on multiple platforms. Table 1 shows hardware metrics of GPU and CPU on TSUBAME, and A64FX on FLOW. Table 2 shows the kernel performance analysis against the modified roofline model [Shimokawabe et al., SC'10, 2010].

AXPY and inner product kernels are memory intensive kernels without neighbor stencil access. Since the block-structured data format enables continuous memory access in the leaf, these kernels achieved ideal sustained performance with the performance ratio against the roofline model $t_R/t > 0.5$.

SpMV kernel is a very important kernel, because SOR kernels also have similar memory access. The performances on GPU and CPU are reasonable, but there is a significant performance degradation on A64FX. According to the optimization report, the innermost loop was not SIMD-optimized due to inline functions and a conditional branch in the loop. Since Fujitsu C++ compiler cannot optimize any function call in a loop, it was necessary to perform manual inline expansion for those functions and to replace the conditional branch with integer flags. Thanks to these optimization techniques, finally, the performance on A64FX became comparable that on CPU and GPU.

CR-SOR kernel is computationally intensive kernel on each MG level. Memory accesses are almost the same as SpMV kernel, but this kernel computes multiple SpMV processes on a local cache. The performances on GPU and CPU were reasonable, $t_R/t > 0.35$ on GPU and $t_R/t > 0.15$ on CPU. On the other hand, the performance on A64FX was quite low, $t_R/t < 0.02$, while we expected the similar sustained performance as

Table 1: Hardware metrics of the TSUBAME and Flow platforms.

	TSUBAME NVIDIA P100	TSUBAME Broadwell (2 sockets)	FLOW A64FX (1 CMG)
Performance [GFlops]	4700	851.1	844.8
Bandwidth [GByte/s]	732	153.6	256
Compiler: compiler options	nvcc -O3 -restrict -use_fast_ math	icpc -O3 -qopenmp -AVX2 -ipo -restrict	mpiFCCpx -O3 -Kfast, parallel, openmp, restp=all

Table 2: Kernel performance analysis: Floating point operation f [Flop/grid], Memory access b [Byte/grid], Elapse time t [ns/grid], Roofline time $t_{RL} = f/F + b/B$ [ns/grid]. Here F is Peak performance [Flops], and B is memory band-width [Byte/s]. (Note: The symbol * indicates the performance after optimization.)

PCG kernels		TSUBAME NVIDIA P100		TSUBAME Broadwell (2 sockets)		FLOW A64FX (1 CMG)	
	f / b	t	t_{RL}/t	t	t_{RL}/t	t	t_{RL}/t
AXPY	3 / 24	0.045	0.75	0.26	0.62	0.13	0.74
Inner product	2 / 16	0.033	0.68	0.22	0.50	0.09	0.72
SpMV	13 / 72	0.141	0.72	1.19	0.41	7.45 *0.51	0.04 *0.58
CR-SOR (16) Lv.0	257 / 80	0.447	0.37	6.03	0.14	72.7	0.01
CR-SOR (8) Lv.1	129 / 80	0.368	0.37	3.75	0.18	26.4	0.02
CR-SOR (8) Lv.2	129 / 80	0.472	0.29	3.09	0.22	25.3	0.02

SpMV kernel. Although we investigated this issue with our collaborators from Fujitsu, the performance degradation was not resolved even with the above manual optimization. Possible causes for this issue may be deeper nested loops due to inner iterations and the two-color ordering or very tight scheduling for on-cache computing.

(2) Convergence Histories of MG-CG Method

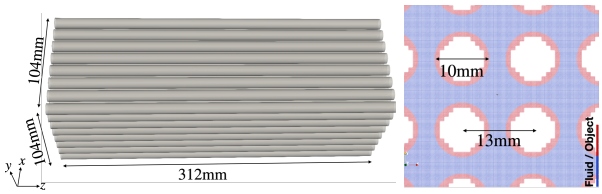
The numerical experiments were conducted for two-phase flows in a fuel bundle of a nuclear reactor shown in Fig. 4. Since computational grids inside fuel pins are removed, the total number of grids is reduced to 2.26×10^9, which is about 70% of the original uniform Cartesian grid system.

Figure 5 shows convergence histories of the P-CG solver and the MG-CG solvers with the RB-SOR smoother (MG-CG) and with the CR-SOR smoother (CRMG-CG). Although the P-CG solver required more than 1,300 iterations until convergence, the number of iterations in the MG-CG solver was 194, which was less than 15% of the P-CG solver. The CRMG-CG solver showed faster convergence with 114 iterations, which was less than 9% of the P-CG solver.

(3) Strong Scaling Test on GPU

The strong scaling of JUPITER with the MG-CG and CRMG-CG solvers is summarized in Fig. 6. In the strong scaling test, we use 64, 128, 192, and 256 GPUs. The labels of “Precondition”, “CG”, and “NS” indicate the preconditioner, the CG method including the inner product, and the Navier-Stokes solver with the explicit time integration, respectively.

The breakdown of computational time on 64 GPUs shows that Precondition accounts for more than 90% of the total time, the remaining cost comes from CG, and NS is negligible. A similar breakdown was observed up to 256 GPUs. In the MG-CG and CRMG-CG solvers, global collective communication in the CG method is still negligible, and halo data communication gives a dominant communication cost. The performance gains of the CRMG-CG solver from 64 GPUs are 1.5×, 1.9



(A) (B)

Fig.4: (A) shows the computational domain for a 8×8 fuel bundle. The size of the whole domain is 104 mm×104 mm ×312 mm. (B) shows the block-structured grid. Blue and red leaves show fluid and boundary regions, which are allocated in the simulation. Leaves inside the pins (white area) are not allocated.

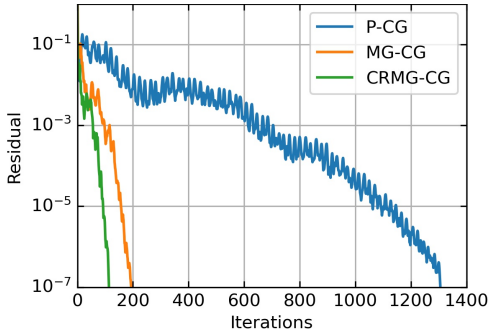


Fig.5: Convergence histories of the P-CG, MG-CG, and CRMG-CG solvers. A large matrix from a two phase flow problem in a 8×8 fuel bundle is computed using 64 GPUs. The P-CG, MG-CG, and CRMG-CG solvers are respectively converged with 1,306, 194, and 114 iterations. The calculation time until convergence of the P-CG, MG-CG, and CRMG-CG solvers are respectively 446, 142, and 76 seconds.

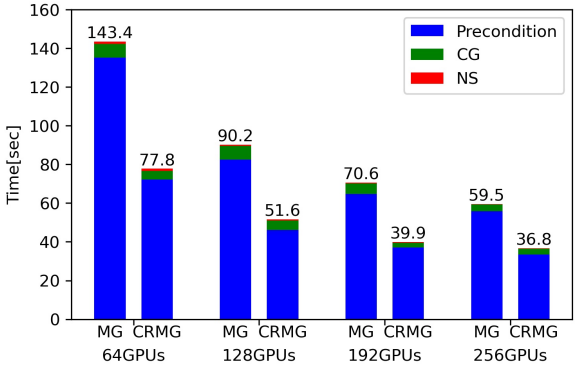


Fig.6: Strong scaling tests of JUPITER with the MG-CG and CRMG-CG solvers up to 256 GPUs. The computational condition is the same as Fig. 9. It is noted that the total time of JUPITER with the P-CG solver is 447.5 second on 64 GPUs.

×, and 2.1× on 128, 192, and 256 GPUs, respectively. It is noted that the calculation speed of the CRMG-CG solver on 256 GPUs is 12 times faster than that of the P-CG solver on 64 GPUs.

6. Progress during FY2020 and Future Prospects

This project developed a new MG-CG solver to accelerate a three-dimensional two-phase flow CFD code JUPITER. The MG-CG solver is designed to achieve high performance on the block-structured Cartesian grid.

We measured the kernel performance of the MG-CG solver on CPU and GPU on TSUBAME, and A64FX on FLOW. Although AXPY, inner product, and SpMV kernels achieved reasonable performances on all platforms, CR-SOR kernels showed a significant performance degradations on A64FX. In the future project, this issue will be investigated on FLOW and BDEC under collaboration with Information Technology Center, Nagoya Univ. and Information Technology Center, Univ. Tokyo.

We discussed the convergence property for two-phase flows in a fuel bundle. The number of iterations of the MG-CG and CRMG-CG methods were respectively less than 15% and 9% compared to the P-CG method.

The strong scaling of JUPITER was measured from 64 to 256 GPUs. The performance gain of the CRMG-CG solver from 64 GPUs to 256 GPUs was 2.1×. We conclude that the redesigned JUPITER code is highly efficient and enables large-scale simulations of two-phase flows on GPU based supercomputers.

7. List of Publications and Presentations

(1) Journal Papers (Refereed)

(2) Proceedings of International Conferences (Refereed)

- N. Onodera, Y. Idomura, Y. Hasegawa, S. Yamashita, T. Shimokawabe, T. Aoki, “GPU Acceleration of Multigrid Preconditioned Conjugate Gradient Solver on Block-Structured Cartesian Grid”, Proceedings of HPC Asia 2021, 2021/01, 査読有り
- N. Onodera, Y. Idomura, Y. Ali, S. Yamashita, T. Shimokawabe, T. Aoki, “GPU-acceleration of locally mesh allocated two phase flow solver for nuclear reactors”, Proceedings of SNA+MC2020, 2020/05, 査読有り

(3) International conference Papers (Non-refereed)

- N. Onodera, Y. Idomura, Y. Asahi, Y. Hasegawa, S. Yamashita, T. Shimokawabe, T. Aoki, “Multigrid Poisson Solver for a Block-Structured Adaptive Mesh Refinement Method on CPU and GPU supercomputers”, Compsafe 2020, 2020/12, 査読無し

(4) Presentations at domestic conference (Non-refereed)

- 小野寺 直幸, 井戸村 泰宏, 長谷川雄太, 山下晋, 下川辺 隆史, 青木 尊之, “ブロック型適合細分化格子を用いた気液二相流体解析手法の開発”, 原子力学会 2021 年春の年会, 2021/3, 査読無し
- 小野寺 直幸, 井戸村 泰宏, 朝比 祐一, 下川辺 隆史, 青木 尊之, “ブロック型適合細分化格子での Poisson 解法の GPU・CPU・ARM プロセッサに対する性能測定”, 第 34 回数値流体力学シンポジウム, 2020/12, 査読無し
- 小野寺 直幸, 井戸村 泰宏, ユスフ アリ, 下川辺 隆史, 青木 尊之, “ブロック型適合細分化格子での Poisson 解法の GPU 高速化”, 計算工学講演会論文集, 2020/06, 査読なし