Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures
Final Report for JHPCN Joint Research of FY 2025

jh250037

Practical acceleration methods to achieve high performance for large-scale applications

Takashi Shimokawabe（The University of Tokyo）

Abstract In recent years, many supercomputers have adopted accelerators such as GPUs to maximize computational performance under constraints of power consumption and physical space. This project aims to port CPU-based applications running on supercomputers to accelerator-equipped systems and to establish practical methodologies for such porting. While targeting high performance on accelerators, the project emphasizes the use of standard, general-purpose approaches rather than relying heavily on accelerator-specific languages, enabling broader accessibility across various research fields. In this year, we conducted performance evaluation and optimization of the seismic wave propagation code OpenSWPC on the GH200 architecture. We also evaluated the performance of an N-body simulation code using Kokkos in comparison with other programming approaches, and extended the use of Kokkos to port Fortran-based applications. In addition, performance evaluation of a fluid dynamics code based on Kokkos was carried out to assess its effectiveness.

1. Basic Information

(1) Collaborating JHPCN Centers

Tohoku University

The University of Tokyo

Nagoya University

(2) Theme Area

Large scale computational science area

(3) Project Members and Their Roles

- Takashi Shimokawabe (The University of Tokyo): Development and optimization of fluid codes
- Akira Nukada (University of Tsukuba): Porting and optimization of OpenSWPC
- Yuuichi Asahi (CEA): Development and optimization of molecular density functional theory (MDFT) code
- Takashi Furumura (The University of Tokyo): Advice on OpenSWPC
- Hiroyasu Hasumi (The University of Tokyo): Advice on COCO
- Takao Kawasaki (The University of Tokyo): Advice on COCO
- Takateru Yamagishi (RIST): Advice on COCO
- Taisuke Boku (University of Tsukuba): Advice on accelerators
- Daisuke Takahashi (University of Tsukuba): Advice on accelerators
- Yohei Miki (The University of Tokyo): Development and optimization of N-body simulation codes
- Kazuya Yamazaki (The University of Tokyo): Advice on Fortran
- Toshihiro Hanawa (The University of Tokyo): Advice on large-scale simulations
- Kengo Nakajima (The University of Tokyo): Advice on computational science
- Masatoshi Kawai (Tohoku University): Advice on Fortran
- Tetsuya Hoshino (Nagoya University): Advice on accelerators
- Hayato Shiba (University of Hyogo): Advice on accelerators
- Julien Bigot (CEA): Advice on Kokkos
- Naoyuki Onodera (Prometech Software Inc.): Advice on accelerators and fluid dynamics
- Yuta Hasegawa (Prometech Software Inc.): Advice on accelerators and fluid dynamics
- Ziheng Yuan (The University of Tokyo): Code development
- Tao Wang (The University of Tokyo): Code development
- Kagetora Kevin Miyamoto (The University of Tokyo): Code development
- Yize Yang (The University of Tokyo): Code development
- Jipeng Su (The University of Tokyo): Code development
- Taiyo Ito (The University of Tokyo): Code development
- Hiroki Kaneko (The University of Tokyo): Code development
- Choi Yeon Kyu (University of Tsukuba): Code development

2. Purpose and Significance of the Research

Supercomputers have increasingly adopted GPUs and other accelerators to enhance performance under power and space constraints. In the latest TOP500 rankings, 9 of the top 10 systems are equipped with GPUs. This trend means the use of accelerators will be essential for high performance on future supercomputers. Joint Center for Advanced High Performance Computing (JCAHPC), a collaboration between the University of Tokyo and University of Tsukuba, will launch the Miyabi supercomputer, on January 14, 2025. Miyabi-G will be equipped with NVIDIA GH200, each comprising a H100 GPU and a Grace CPU. Many CPU-optimized applications run on both centers' supercomputers. Establishing guidelines and methods for efficiently porting these applications to Miyabi-G is important.

This research project focuses on porting CPU applications to accelerator-equipped supercomputers and developing methodologies for this transition. To achieve high performance with accelerators, we will accomplish the porting without using accelerator-specific languages. We will use standard, general-purpose methods so that researchers in fields other than high-performance computing (HPC) can easily use them.

Last year, we ported several applications, including the seismic wave propagation code OpenSWPC (https://github.com/OpenSWPC), to GPU using language standard methods in preparation for Miyabi. This year, we will verify their performance and optimize them for Miyabi-G. Previous research has demonstrated that, while standard, directive-based methods are highly productive, their performance is lower than that of using accelerator-specific languages. Therefore, this year, we will also aim to clarify the performance and productivity that applications based on Kokkos (https://kokkos.org) can achieve. Kokkos is a prominent performance portability library that provides high productivity and performance on GPUs from various vendors.

3. Significance as JHPCN Joint Research Project

This project, conducted as a JHPCN joint research initiative, demonstrated the importance of cross-disciplinary collaboration in advancing accelerator-based high-performance computing. By integrating the expertise of domain scientists and HPC specialists, it established a practical and general-purpose methodology for porting existing applications to modern accelerator architectures. Domain experts contributed codes, datasets, and knowledge of data structures, ensuring that the original design and sustainability of applications such as OpenSWPC, fluid dynamics, and N-body simulations were preserved. HPC experts applied optimization techniques with a focus on the Kokkos library, achieving high performance while maintaining code readability and portability. The use of Miyabi-G (NVIDIA GH200) and comparisons with Wisteria-Aquarius (A100) enabled valuable evaluation of performance portability. These efforts highlight the project's strong relevance to JHPCN by promoting effective collaboration and developing reusable methodologies. These outcomes are expected to significantly contribute to the future advancement of accelerator utilization.

4. Outline of Research Achievements until FY2024

This continuing research project, initiated in FY2022, focuses on porting applications to GPUs and improving performance portability. The project aims to port the OpenSWPC seismic wave propagation code and the COCO ocean model, both written in Fortran, to GPUs. In FY2022 and early FY2023, OpenSWPC was successfully ported using the DO CONCURRENT Fortran parallelization, achieving performance of about 8 CPUs per GPU on the Wisteria-Aquarius node. Despite lower inter-node communication performance compared to GPU-specific languages like CUDA, a method for explicitly reserving device memory improved performance. From the second half of FY2023 to the present, we have been working on GPU porting of complex COCO codes. Additionally, N-body and fluid computation codes written in C++ have been parallelized and implemented on GPU using standard C++ syntax and/or SYCL. The N-body code was implemented using both CUDA and SYCL. The SYCL implementation achieved the highest performance on the NVIDIA H100 GPU, demonstrating its high-performance portability. The project is also working on developing a FFT performance portability library with Kokkos.

5. Details of FY2025 Research Achievements

This section summarizes the research achievements obtained in FY2025. It presents the performance evaluation and optimization of the seismic wave propagation code OpenSWPC on the GH200 architecture, as well as the performance evaluation of an N-body simulation code using Kokkos in comparison with other programming approaches. In addition, it describes the porting of Fortran-based applications using Kokkos and reports on the performance evaluation of a fluid dynamics code developed with Kokkos.

5.1 Performance evaluation and optimization of OpenSWPC for the GH200

On the NVIDIA GH200 Grace Hopper Superchip, the NVLink-C2C interconnect enables a hardware-coherent shared memory space between the CPU and GPU. This year, we investigated a specific execution mode enabled by the compiler option -gpu=mem:unified:nomanagedalloc.
Unlike the standard Managed Memory mode, which explicitly treats allocatable arrays as managed objects, this mode keeps allocations in their standard state while allowing the GPU to transparently access all memory addresses.
Our comparative analysis between this "Hardware-Coherent Unified Memory" and the traditional "Managed Memory" revealed critical differences in data movement:
Managed Memory (allocatable as managed): This mode relies on "Eager Migration." When the GPU accesses a page, the OS/driver immediately migrates that page from CPU memory to GPU memory. This ensures that the GPU kernel benefits from the high local bandwidth of HBM3. However, the migration itself incurs a latency penalty during the initial access, and subsequent CPU access will pull the page back to the host side.
Unified Memory with nomanagedalloc: In this mode, pages do not necessarily migrate upon the first GPU access. Instead, the hardware allows the GPU to perform "Remote Access" to the CPU's memory (LPDDR5X) across the NVLink-C2C. While this avoids the immediate

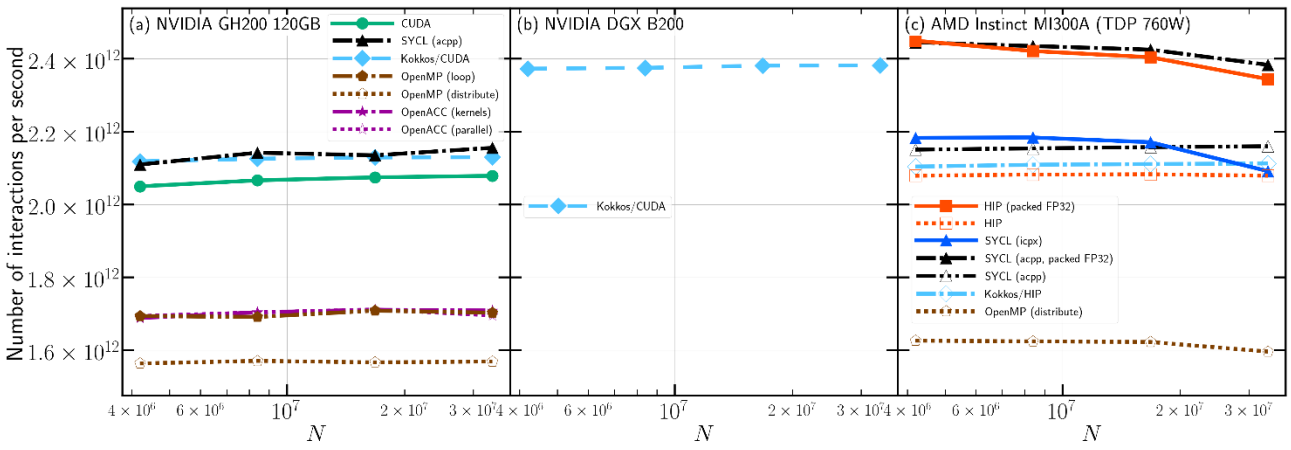

Figure 1 Performance comparison of the direct N-body code on NVIDIA GH200, NVIDIA B200, and AMD MI300A. The figure shows the best measured number of interactions per second as a function of particle number N for each implementation.

overhead of page migration, it can limit performance to the interconnect bandwidth rather than the peak GPU memory bandwidth. Interestingly, we observed that page migration in this mode is "Conservative"—triggered only by high access frequency or specific hardware heuristics—and once a page resides in GPU memory, it remains there even during CPU-side inspections.

Our evaluation with OpenSWPC using Fortran Stdpar showed that while Managed Memory remains highly effective for data-intensive kernels, the Unified Memory mode (nomanagedalloc) offers a unique advantage for complex workflows where the CPU frequently monitors or interacts with the same data structures without wanting to trigger expensive "ping-pong" migrations.

5.2 Performance evaluation of N-body code based on Kokkos and other programming approaches

As a representative study within our broader project on practical GPU acceleration for large-scale scientific applications, we carried out a systematic comparison of GPU programming methods using a direct N-body code as a compute-intensive benchmark. The code was implemented and optimized with CUDA, HIP, SYCL, Kokkos, and directive-based offloading through Solomon (OpenACC/OpenMP target), and evaluated on NVIDIA GH200 (Miyabi-G), NVIDIA B200, and AMD MI300A. As shown in Figure 1, performance-portable and vendor-neutral approaches achieved performance comparable to, and in some cases higher than, vendor-specific low-level implementations. On NVIDIA GH200, the best result was obtained by the SYCL implementation compiled with AdaptiveCpp, which reached 2.16×10^{12} interactions/s at $N = 33{,}554{,}432$, 3.7% higher than the CUDA C++ implementation. The Kokkos implementation also reached 2.13×10^{12} interactions/s, exceeding CUDA by 2.5%. These results show that performance portability can be realized on Miyabi-G without sacrificing performance.

We also examined portability across GPU generations and vendors. On NVIDIA B200, the Kokkos implementation reached 2.38×10^{12} interactions/s at $N = 33{,}554{,}432$, corresponding

to a 1.12× speedup over GH200, which is very close to the theoretical peak-performance ratio of 1.13×. This indicates that high performance can be retained across NVIDIA generations with only general tuning, such as team size and loop-unrolling parameters, without major architecture-specific rewriting. On AMD MI300A, the highest performance was 2.45 × 10^{12} interactions/s at N= 4,194,304, obtained by HIP and SYCL implementations using packed FP32 instructions. Even without AMD-specific tuning, the standard SYCL and Kokkos implementations delivered 2.16 × 10^{12} and 2.11 × 10^{12} interactions/s, respectively, which are close to the GH200 results. On GH200, directive-based implementations through Solomon still sustained about 82% of CUDA performance for the OpenACC and OpenMP loop variants and 76% for the OpenMP distribute variant, while greatly reducing porting cost and preserving greater commonality with CPU-oriented source code. To support more effective use of computer resources, we also tuned block or team size, loop unrolling, instruction-level parallelism, reciprocal-square-root operations, and shared-memory usage for each backend. The FY2025 study therefore provided both scientific results and practical guidance for efficient use of accelerator-based JHPCN systems.

5.3 Porting Fortran applications using Kokkos

We have ported a mini-application of molecular density functional theory (MDFT) code from Fortran to Kokkos. As this code heavily relies on Fast-Fourier Transform (FFT), we have developed a FFT interface for the Kokkos eco-system. Compared to the original Fortran version parallelized with OpenMP, the Kokkos version achieves the speed-ups of 31.2, 46.2, 12.3, and 12.6 on V100, A100, MI250X and PVC, respectively.

In FY2025, we extended Kokkos-FFT to support distributed FFTs. The performance portability of our distributed-FFT library comes for free as part of Kokkos ecosystem. In stead of spending time to implement across multiple backends, we focus on developing the unique features such as batched distributed transforms on data structures up to 7 dimensions and an interface to third parity distributed FFT libraries. To evaluate the performance and scalability of our library, we develop a benchmark application that solves 3D Navier-Stokes equation with Fourier spectral method. We measured the strong scaling of this application on NVIDIA A100, AMD MI250X and Intel PVC GPUs from 2 nodes to 8 nodes using the GPU-Aware MPI backend. With 8 nodes, we found almost the same performance on all GPU platforms wherein the MPI communications account for about 80% to 90% of the total execution time. This implies that GPU kerels are appropriately accelerated on all the platforms. We have also compared the performance of our native implementation against the cuFFTMp. We found that the cuFFTMp is 50 % times faster than our implementation at scale which likely results from the accelerated communications by NVSHMEM. Integration of NVSHMEM as our communication backend remains as a future task. In this work, we also improved the performance of multi-dimensional parallelization in Kokkos which benefits the entire Kokkos community.

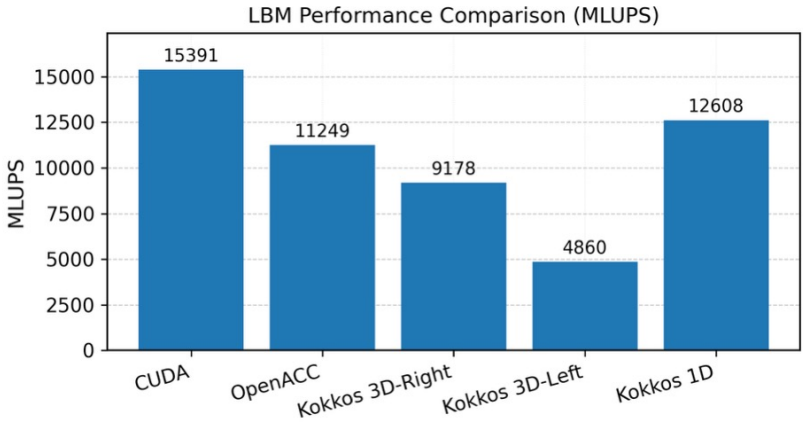

Figure 2 Performance comparison of various implementations of the Lattice Boltzmann Method on NVIDIA GH200.

5.4 Performance evaluation of fluid dynamics code based on Kokkos

In FY2050, we conducted a performance comparison of Lattice Boltzmann solvers implemented using CUDA, OpenACC, and Kokkos with different data layouts and execution policies.

The experimental results indicate that the Kokkos 1D implementation achieves the highest performance among the Kokkos variants and closely approaches the CUDA baseline. As shown in Figure 2, the CUDA implementation reaches approximately 15,400 MLUPS, while the Kokkos 1D implementation attains around 12,600 MLUPS, demonstrating competitive performance. In comparison, OpenACC achieves roughly 11,200 MLUPS. In contrast, the multi-dimensional Kokkos implementations exhibit significantly lower performance. The Kokkos 3D (MDRangePolicy) version achieves only 9,178 MLUPS, indicating a substantial degradation relative to both CUDA and Kokkos 1D. This highlights the performance limitations associated with multi-dimensional abstractions in this context.

Furthermore, analysis of the runtime breakdown shows that the communication time is consistently low and nearly identical across all implementations. This confirms that MPI communication overhead is not a dominant factor in overall performance differences. Instead, the streaming–collision kernel accounts for the majority of execution time, with noticeable variation across implementations.

These observations indicate that performance differences are primarily governed by kernel-level efficiency, particularly memory access patterns in the streaming step, rather than communication costs. Consequently, optimizing memory efficiency within the compute kernel is critical for achieving high performance in 3D Lattice Boltzmann simulations.

6. Self-review of Current Progress and Future Prospects

Overall, we judge the FY2025 progress to be good, and the main objectives for this fiscal year have been substantially achieved. According to the proposal and the progress report, the key targets for FY2025 were: (i) performance evaluation and optimization of OpenSWPC on Miyabi-G (GH200), (ii) performance evaluation of an N-body simulation code using Kokkos and comparison

with other programming approaches, (iii) porting of Fortran applications using Kokkos, and (iv) performance evaluation of a fluid dynamics code based on Kokkos.

6.1 Performance evaluation and optimization of OpenSWPC for the GH200

During FY2025, we clarified the interplay between memory management policies and performance on the NVIDIA GH200 Grace Hopper Superchip. Using OpenSWPC, a Fortran-based seismic wave propagation code, we demonstrated that the hardware-coherent interconnect (NVLink-C2C) significantly simplifies GPU porting via Fortran Standard Parallelism (Stdpar).

A key finding was the effectiveness of the Unified Memory mode (-gpu=mem:unified:nomanagedalloc). Unlike standard Managed Memory, which triggers eager page migrations, this mode utilizes hardware coherency to allow the GPU to access host-side descriptors and data without redundant management overhead. We discovered that this mode can effectively "pin" data in the high-bandwidth GPU memory despite CPU-side inspections, preventing the expensive "ping-pong" migration cycles common in traditional managed environments.

Moving forward, we aim to develop a control layer or specialized library that interfaces with Fortran Stdpar to provide "residency hints." By strategically utilizing the hardware-coherency of the GH200, we hope to achieve a hybrid execution model where data resides in the GPU memory for performance-critical loops while remaining transparently accessible to the CPU for auxiliary tasks. This will further solidify the role of standard language constructs like do concurrent in high-end scientific computing.

6.2 Performance evaluation of N-body code based on Kokkos and other programming approaches

The key targets for FY2025 were: verification and optimization of GPU-ported codes on Miyabi-G and related accelerator systems, and clarification of the attainable balance between performance and productivity for performance-portable frameworks, especially Kokkos, relative to low-level and directive-based approaches. For the representative *N*-body application, both targets were achieved quantitatively. On GH200, SYCL/AdaptiveCpp and Kokkos reached 103.7% and 102.5% of the CUDA performance, respectively, while directive-based implementations still maintained about 76–82% of the CUDA result with substantially lower porting cost. On B200, Kokkos preserved portability across generations with a 1.12× speedup over GH200, close to the 1.13× theoretical expectation. On MI300A, the best optimized implementations reached 2.45×10^{12} interactions/s, while the standard vendor-neutral implementations remained close to the GH200 level. These outcomes provide concrete evidence that high performance on current accelerator systems can be obtained without relying exclusively on accelerator-specific languages. In this sense, the methodological target set for FY2025 can be regarded as largely achieved.

6.3 Porting Fortran applications using Kokkos

In FY 2025, we have ported a mini-application of molecular density functional theory (MDFT) code from Fortran to Kokkos and achieved a reasonable performance on

NVIDIA, AMD and Intel GPUs. We have extended Kokkos-FFT library to support distributed FFTs. This work is presented in the international workshop. Our optimization effort of Kokkos itself also benefits the Kokkos users. We have also started to evaluate the potential of AI frameworks like JAX/Pytorch for HPC simulations. As preliminary results, we confirmed that we can achieve quite competitive performance with these frameworks at least for toy examples. Overall, we have achieved most of the subjects in FY2025.

#Section 7, achievements of this FY, should be input on the JHPCN website.

6.4 Performance evaluation of fluid dynamics code based on Kokkos

In this study, we implemented and compared Lattice Boltzmann solvers using CUDA, OpenACC, and Kokkos, and demonstrated that the Kokkos 1D implementation can achieve performance close to CUDA. This indicates that a certain level of performance portability is attainable. However, the results also reveal that multi-dimensional Kokkos implementations suffer from significant performance degradation, mainly due to inefficiencies in the streaming–collision kernel rather than communication overhead. For future work, we plan to evaluate newer versions of Kokkos, where performance improvements are expected, and assess their impact on our application. In addition, we will focus on further kernel-level optimizations, particularly in memory access patterns, and consider hybrid approaches that combine Kokkos with CUDA to achieve both portability and high performance.