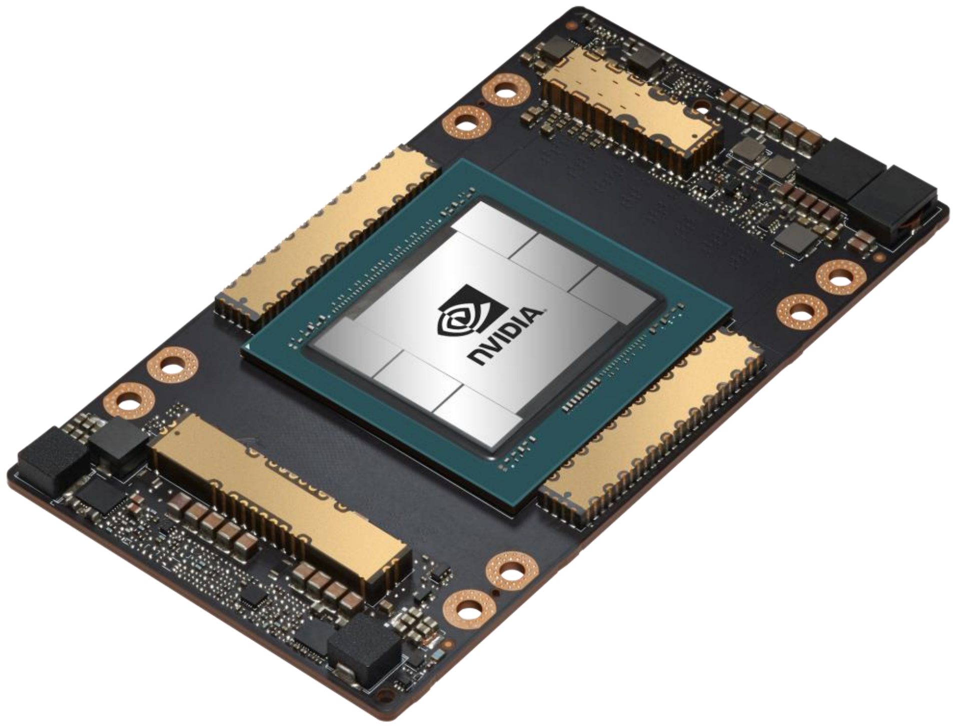
GPU acceleration of OpenMX with OpenACC and CUDA

Hiroyuki Kawai

Department of Physics, Niigata University

What is a GPU?

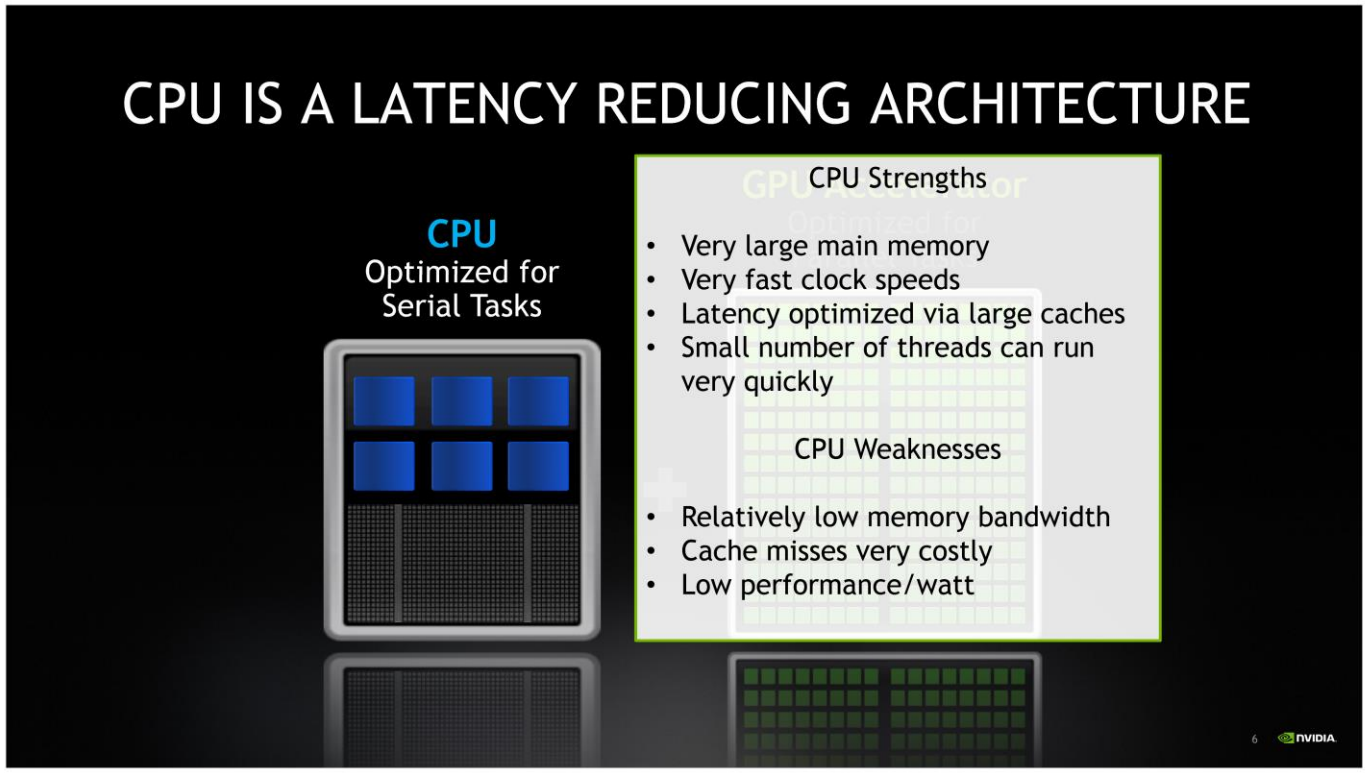
- GPU (Graphics Processing Units) are processors that originally specialized in image processing.
- Recently, the application of GPU computing resources to general-purpose computing purposes other than image processing has become popular.



NVIDIA A100, the GPU used in this study

Comparison of CPU and GPU

Pros and cons of CPU



CPU IS A LATENCY REDUCING ARCHITECTURE

CPU
Optimized for
Serial Tasks

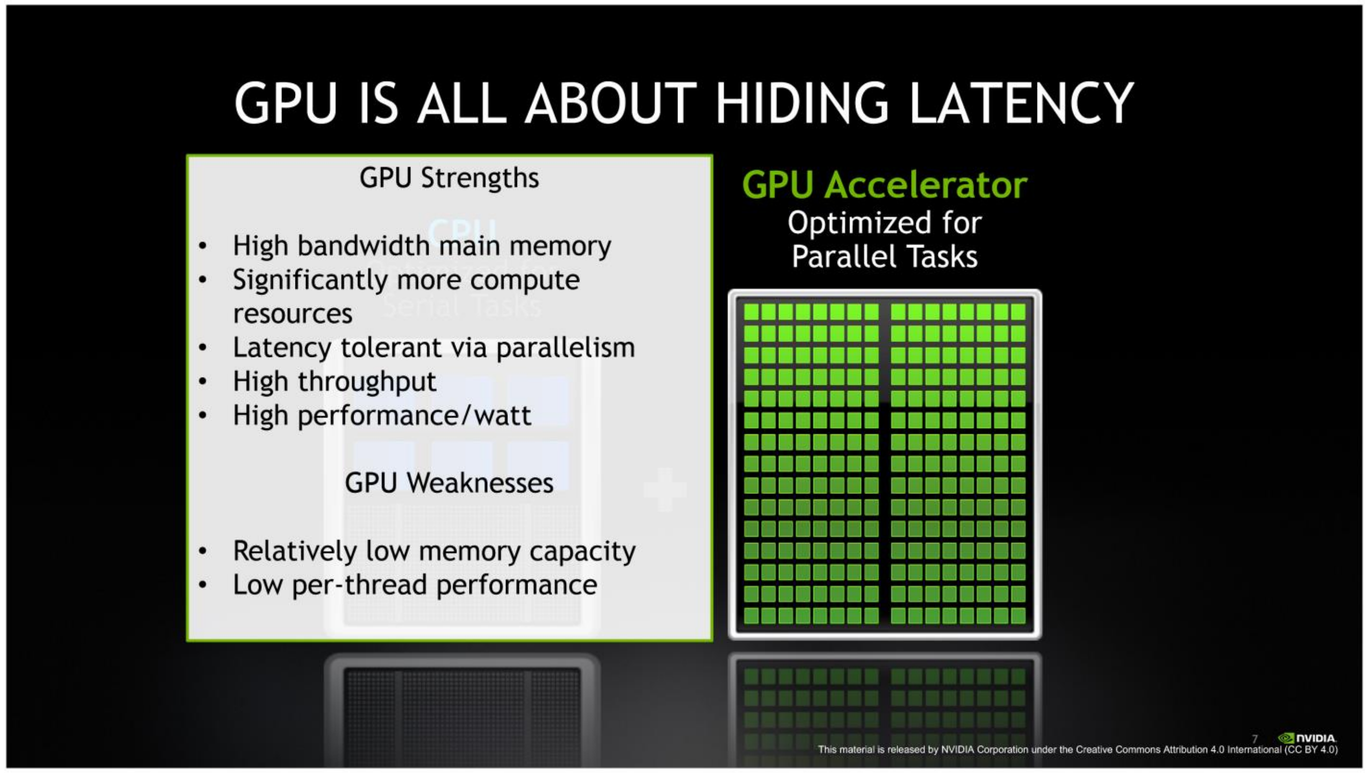
CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

Pros and cons of GPU



GPU IS ALL ABOUT HIDING LATENCY

GPU Strengths

- High bandwidth main memory
- Significantly more compute resources
- Latency tolerant via parallelism
- High throughput
- High performance/watt

GPU Weaknesses

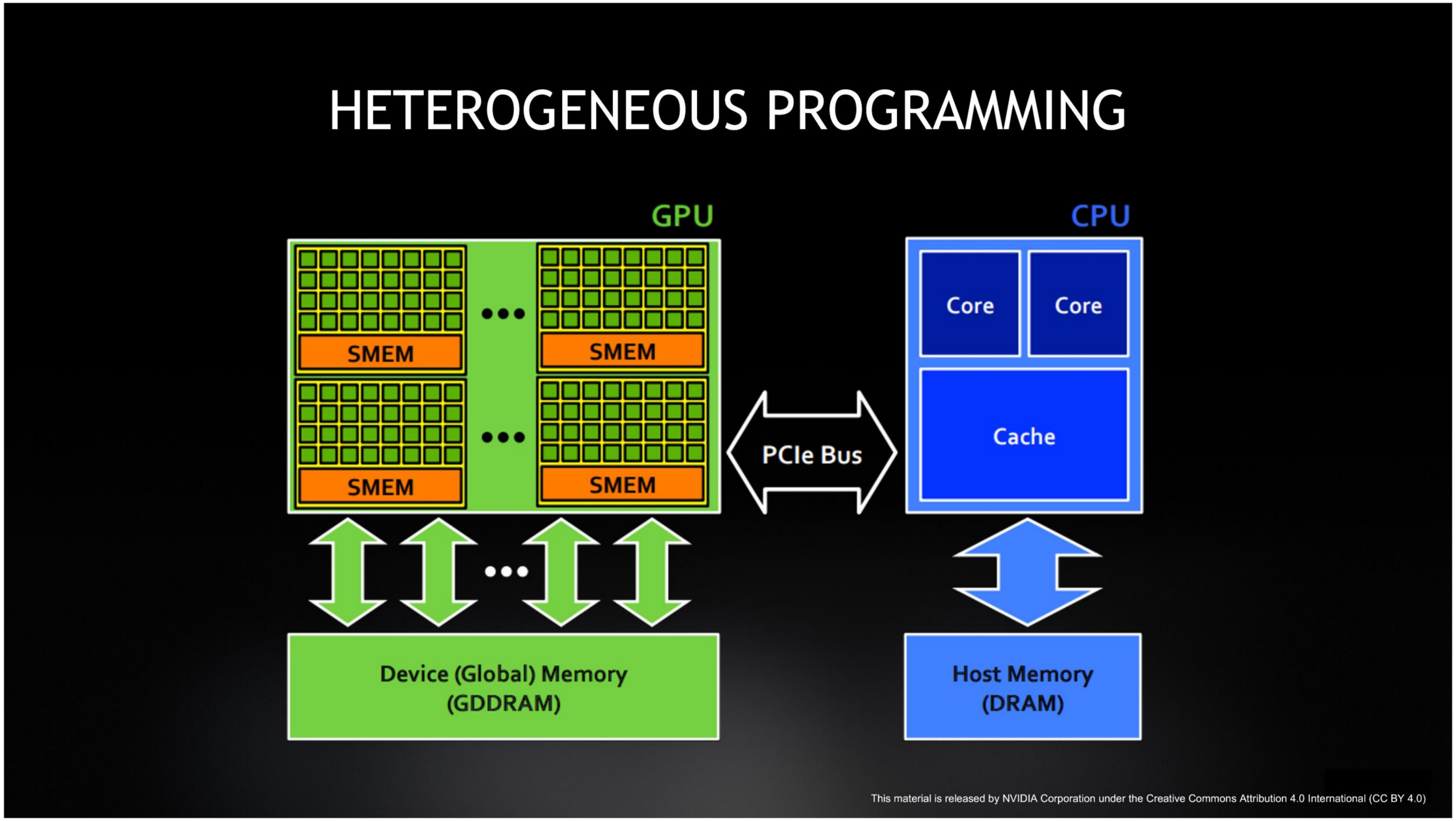
- Relatively low memory capacity
- Low per-thread performance

GPU Accelerator
Optimized for
Parallel Tasks

CPUs and GPUs have their own advantages and disadvantages, so it is best to use a combination of both. Overall efficiency can be improved by assigning processing to the processor most suited for the application.

[1] https://www.cc.u-tokyo.ac.jp/events/lectures/207/20230621-1.pdf

CPU-GPU heterogeneous programming


HETEROGENEOUS PROGRAMMING
GPU
CPU
SMEM
Core
Core
Cache
PCIe Bus
Device (Global) Memory (GDDRAM)
Host Memory (DRAM)
This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

Heterogeneous programming is programming on a computer system built by combining different types of processors.

GPU supercomputers currently in operation

- Japanese domestic
  - ISSP, System C kugui (AMD EPYC 7763 + NVIDIA A100)
  - Information Technology Center, The University of Tokyo, Wisteria/BDEC-01 Aquarius (Intel Xeon Platinum 8360Y + NVIDIA A100)
- Overseas
  - OLCF, Frontier (AMD EPYC 7453s + AMD Radeon Instinct MI250X)
  - CSC, LUMI (AMD EPYC 7A53 + AMD Radeon Instinct MI250X)
  - CINECA, Leonardo (Intel Xeon 8358 + NVIDIA custom Ampere GPU)

Research Objectives

- Currently, many first-principles electronic structure calculation software such as ABINIT, Quantum ESPRESSO, SIESTA, and VASP are GPU accelerated.
- As for OpenMX, there has been only one report of an attempt to accelerate with GPUs[2], and this report was made in the old days when GPUs did not have sufficient performance, so it is hard to say that it showed the superiority of GPUs over conventional CPU-based parallelization (MPI parallelization).

- In this study, we aim to accelerate the bottleneck in OpenMX 3.9, which is the diagonalization of matrices, using GPU. The acceleration through GPU involves the use of CUDA math libraries (cuBLAS and cuSOLVER) and OpenACC. Subsequently, we assess the performance through benchmark calculations.

- For GPU acceleration, Wisteria/BDEC-01 Aquarius from information technology center, the University of Tokyo, was used. For benchmark calculations, Wisteria/BDEC-01 Aquarius and ISSP's system B ohtaka were used.

[2] J. H. Parq, E. Sevre, and S. M. Lee, Int. j. comput. appl. **98**, 20 (2014).

What is CUDA?

- CUDA (Compute Unified Device Architecture) is a GPU program development environment developed by NVIDIA.
- CUDA enables high-speed parallel processing using the GPU's multiple arithmetic units, using C-like program descriptions.
- Programming with CUDA can take GPU performance to the extreme. On the other hand, programming with CUDA is very time-consuming and difficult.
- Also, CUDA only works on NVIDIA GPUs (CUDA does not work on AMD GPUs or Intel GPUs).
- GPU versions of BLAS and LAPACK are also included in CUDA in the form of cuBLAS and cuSOLVER.

What is OpenACC?

- OpenACC is a standard for parallel programming on accelerator devices, not limited to GPUs, using a directive-based programming method (i.e., inserting directive lines in the source) like OpenMP.
- OpenACC can easily program GPUs with #pragma acc directive.
- If you aim to maximize performance to the utmost on GPU, CUDA is preferable. However, if achieving reasonably good performance on GPU is sufficient, then using OpenACC would be a good choice.
- The code written in OpenACC is expected to run on GPUs from AMD and Intel as well, but unfortunately, it seems that AMD and Intel do not intend to support OpenACC. OpenACC is supported only on NVIDIA GPUs, similar to CUDA.
- GPU versions of BLAS and LAPACK are not included in OpenACC. If you need a GPU version of BLAS or LAPACK, you can use cuBLAS or cuSOLVER. Alternatively, you can use an external library called MAGMA[3] instead of cuBLAS or cuSOLVER.
- [3] https://icl.utk.edu/magma/

More about GPU acceleration

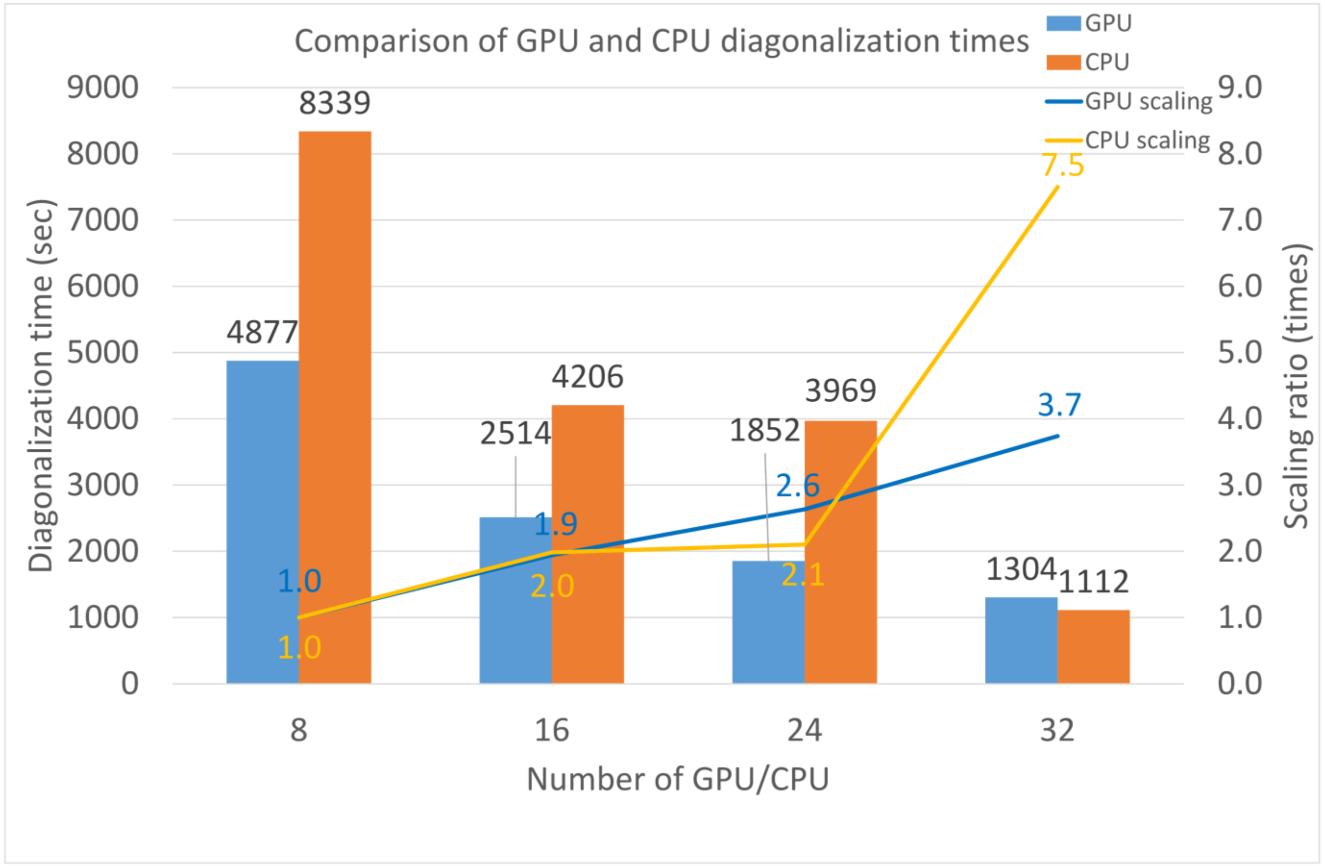
- The typical $O(N^3)$ band calculations (collinear, non-collinear), divide conquer (DC) method, divide-conquer method with localized natural orbitals (DC-LNO) method and NEGF calculations were performed on GPU.
- We cannot say for sure because we have not done enough benchmark calculations, but it appears that the calculation speed of the GPU exceeds that of the CPU on large systems of 5000 basis or more.
- Similarly, in the NEGF calculation, the GPU does not seem to perform well unless the system is large, although benchmark calculations are not sufficient.
- The most successful GPU speedups were achieved with the DC and DC-LNO methods.

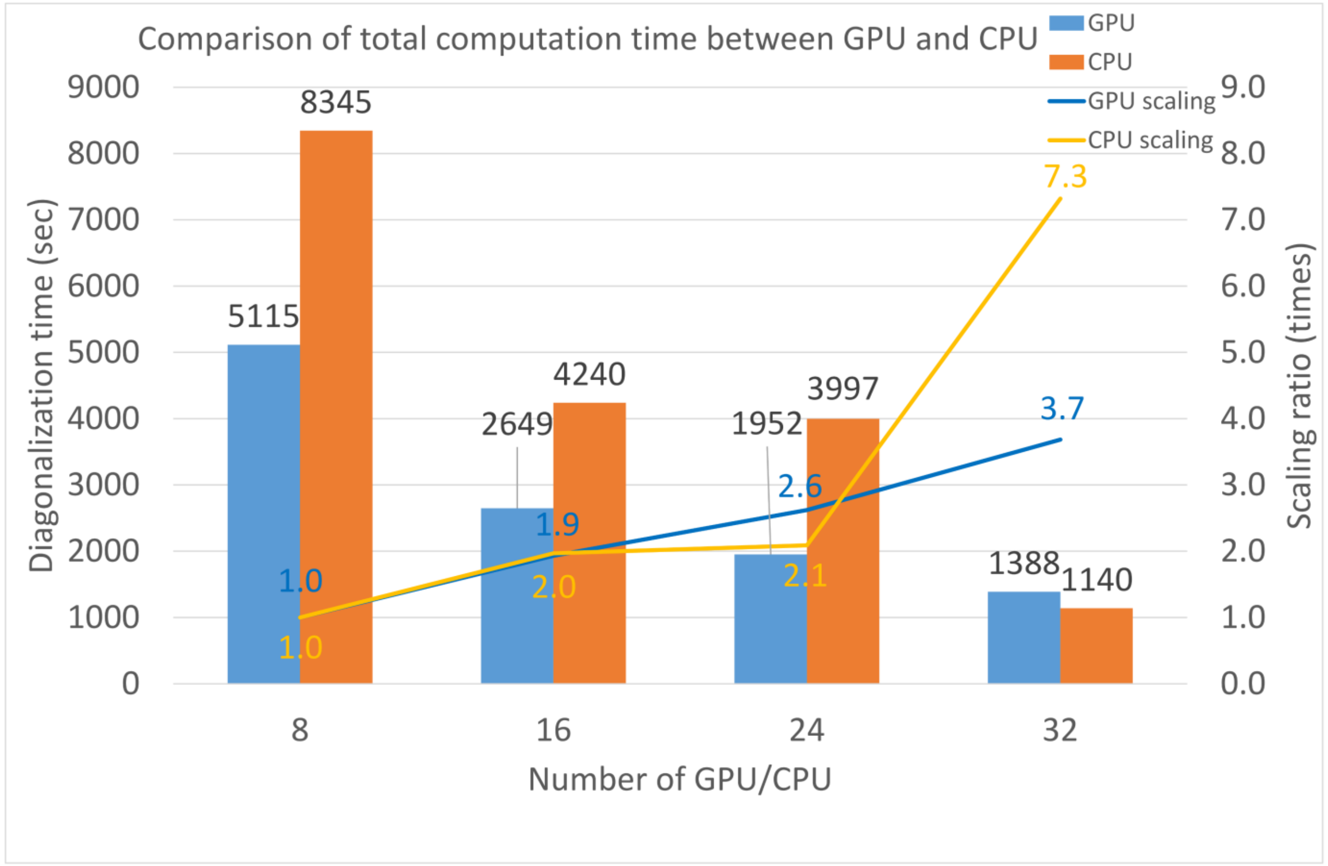
More about GPU acceleration

- For band calculations, the CPU versions of ELPA[4] and ScaLAPACK were accelerated on the GPU by replacing the GPU version of ELPA and ScaLAPACK (cuScaLAPACK[5]), respectively.
- For the DC-LNO method, the real matrix product (DGEMM) and real-symmetric eigenvalue problem solver (dsyevd) of BLAS and LAPACK were replaced by the corresponding routines of cuBLAS and cuSOLVER, respectively, for GPU acceleration.
- The DC method was accelerated on the GPU by offloading some for loops to the GPU with OpenACC and by replacing the LAPACK real-symmetric eigenvalue problem solver (dsyevx) with the corresponding cuSOLVER routines.
- The same procedure was used for the NEGF calculations, which were also performed by GPU.
- In all cases, no modifications to the source code were made at the algorithm level.

- [4] https://github.com/marekandreas/elpa
- [5] https://github.com/smorita/cuscalapack

Comparison of computation time between GPU and CPU for band calculations (collinear)



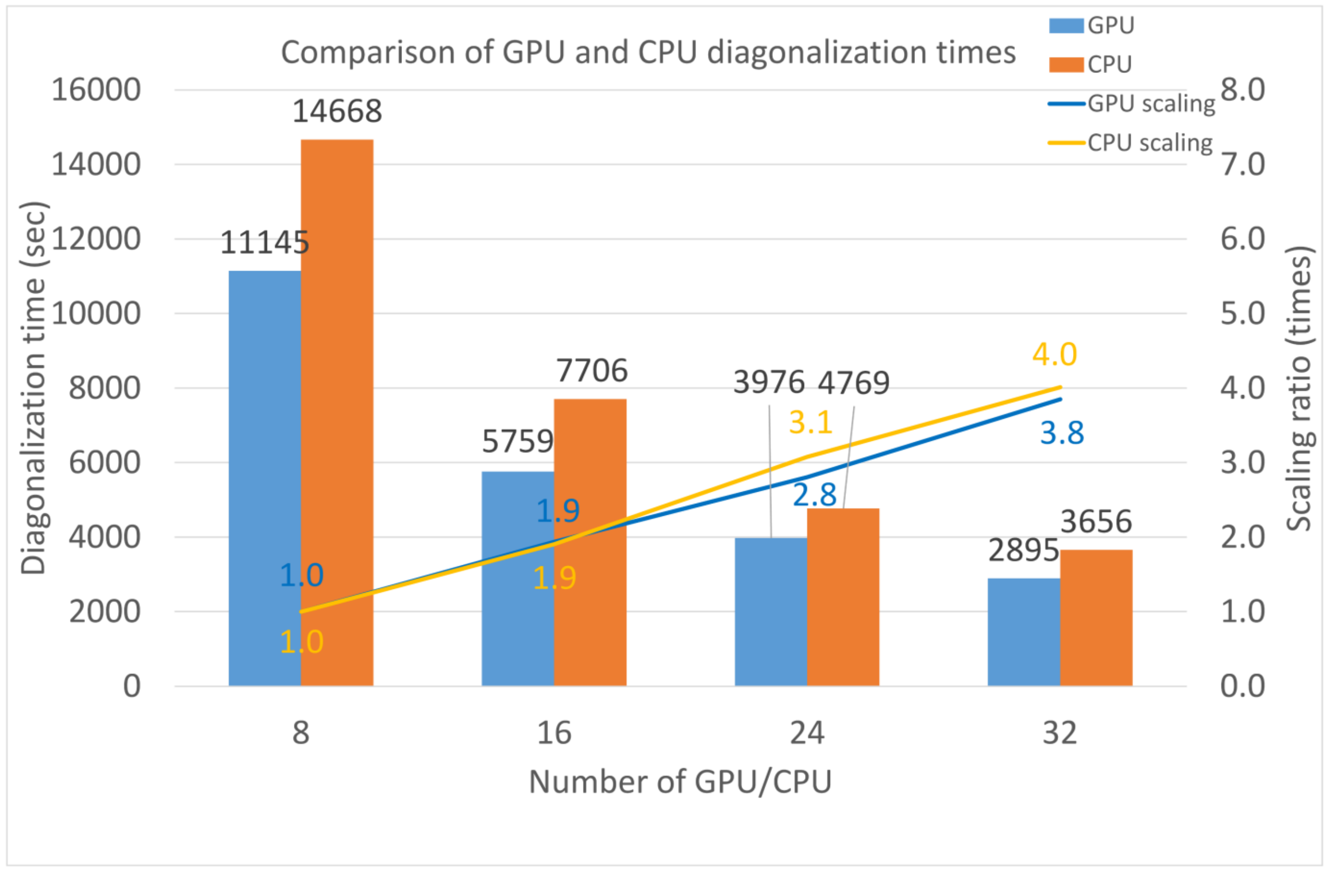


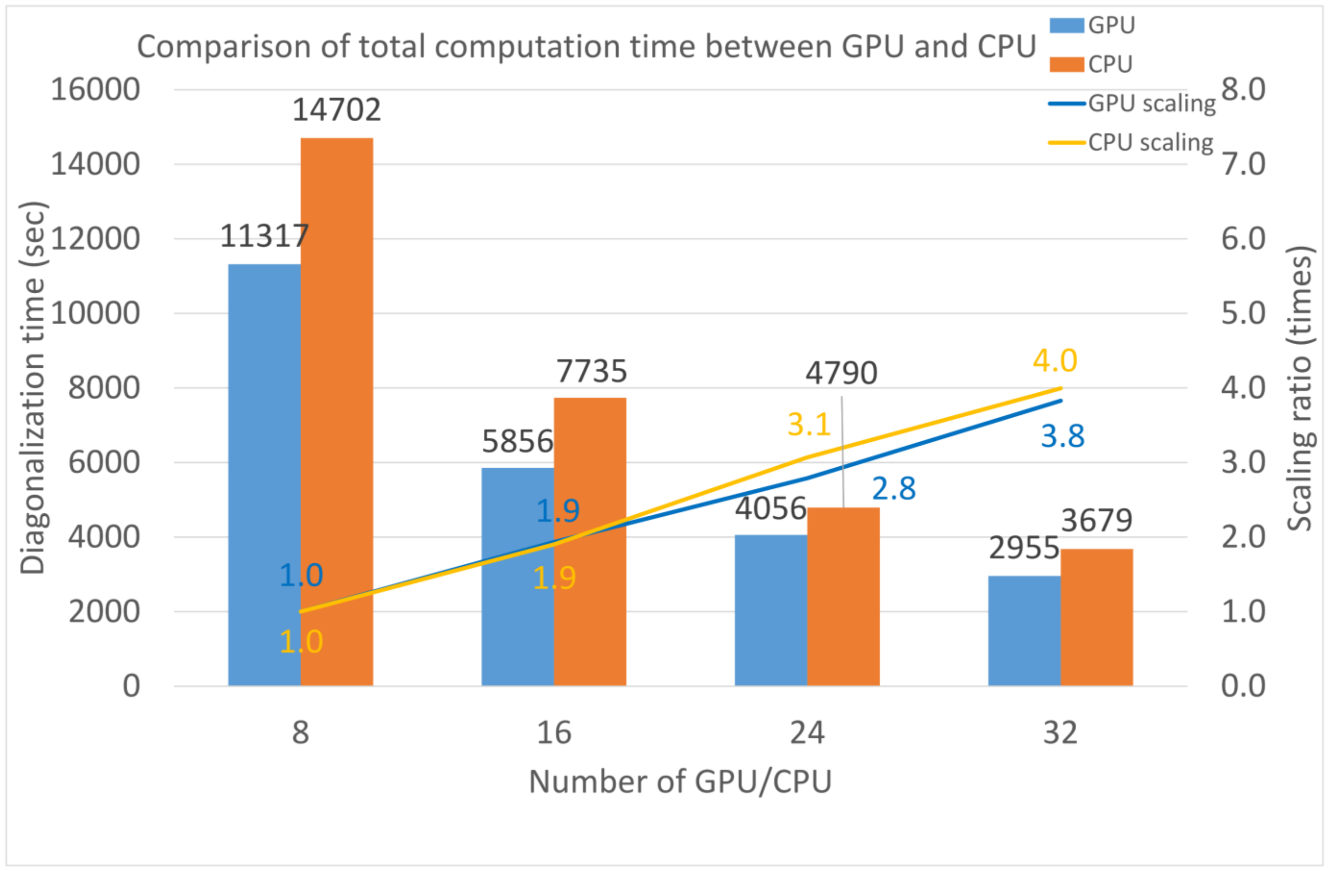
The diagonalization time and total computation time, except for the case with 32 GPU/CPU numbers, consistently show the GPUs to be just under 2 times faster than the CPU. The scaling of the GPUs is better than that of the CPUs, and it scales nearly linearly. This represents an ideal scaling.

Measured by calculation of 384 atoms of silicon supercell (total basis number is 9984), 8×8×8 k-mesh, 1SCF only.

GPU: NVIDIA A100 (8 GPUs per node in Wisteria/BDEC-01 Aquarius) , 4 MPI processes are allocated per GPU.
CPU: AMD EPYC 7763 (2 CPUs per node in System B ohtaka), 16 MPI process allocated per CPU.

Comparison of computation time between GPU and CPU for band calculations (non-collinear)



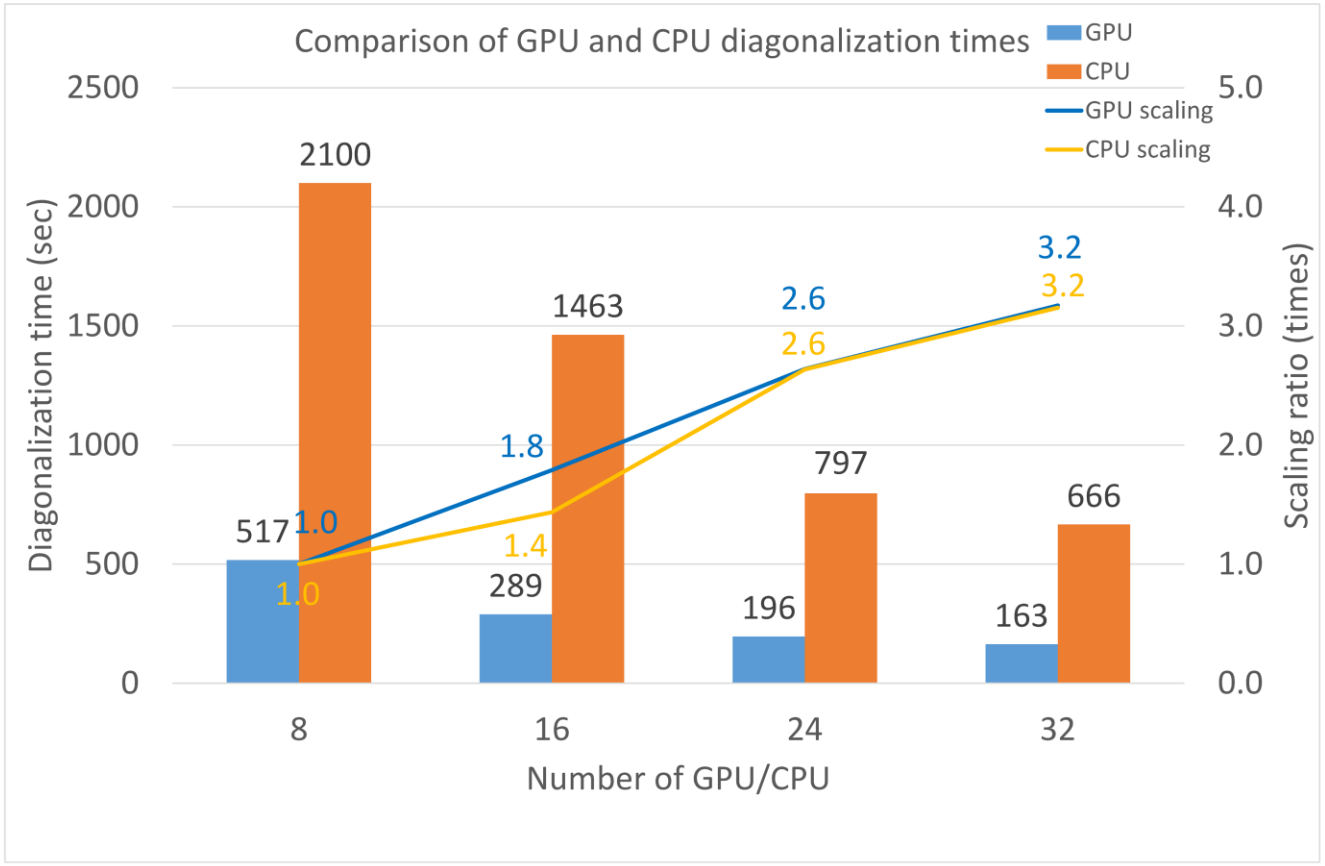


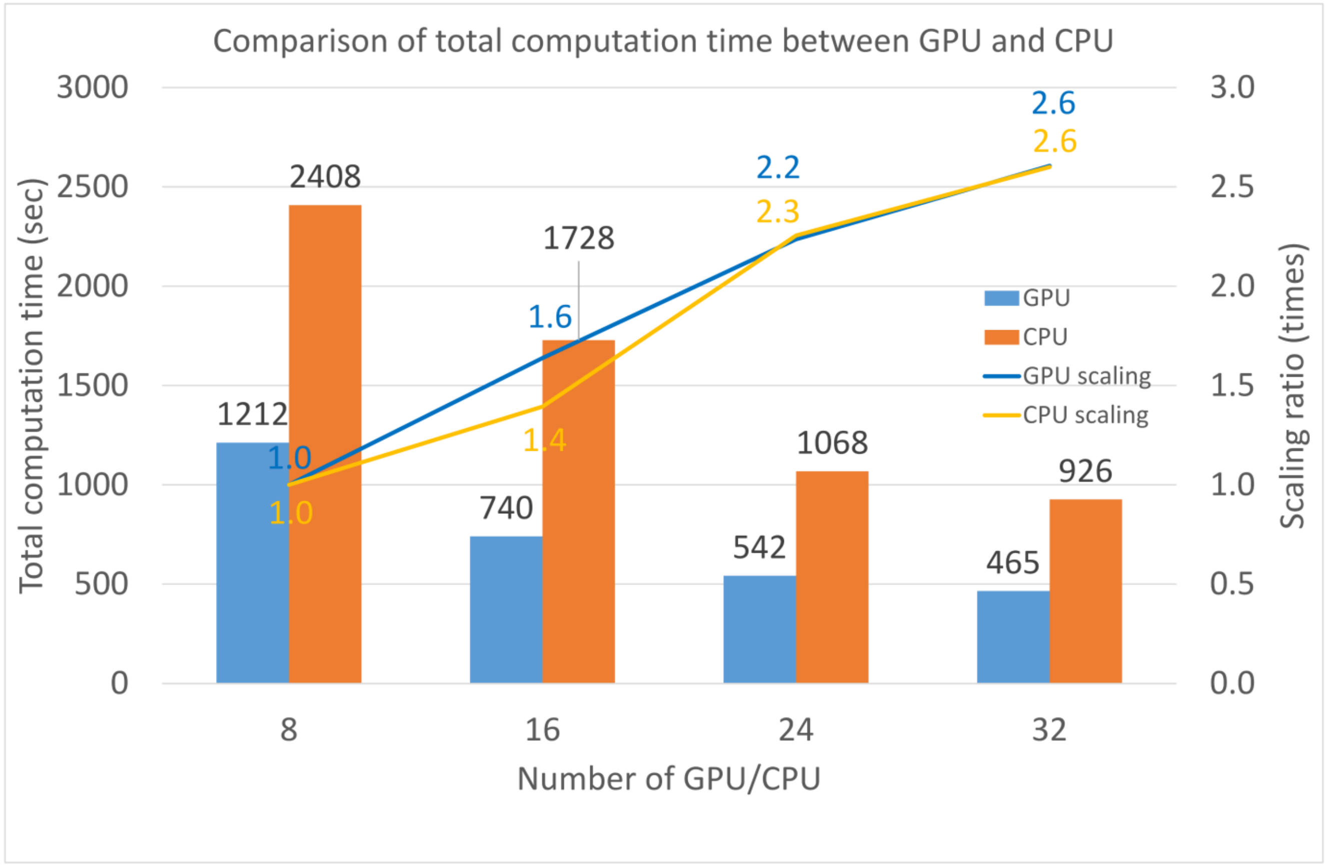
The diagonalization time and total computation time are about 20% to 30% faster on GPUs than on CPUs, regardless of the number of GPUs/CPUs, but scaling on GPUs is worse than on CPUs.

Measured by calculation of 192 atoms of silicon supercell (total basis number is 4992), 9×9×9 k-mesh, 1SCF only.

GPU: NVIDIA A100 (8 GPUs per node in Wisteria/BDEC-01 Aquarius) , 3 MPI processes are allocated per GPU.
CPU: AMD EPYC 7763 (2 CPUs per node in System B ohtaka), 8 MPI process allocated per CPU.

Comparison of computation time between GPU and CPU for DC-LNO method





In terms of diagonalization time, GPUs are 4 to 5 times faster than CPUs for both GPU/CPU counts, and GPU scaling is about the same as that of CPUs. In terms of total computation time, GPUs are about 2 times faster than CPUs for all GPU/CPU numbers, and GPU scaling is about the same as that of CPUs.

Measured by calculation of 650 atoms of DNA (total basis number is 12980)

GPU: NVIDIA A100 (8 GPUs per node in Wisteria/BDEC-01 Aquarius) , 9 MPI processes are allocated per GPU.
CPU: AMD EPYC 7763 (2 CPUs per node in System B ohtaka), 64 MPI process allocated per CPU.

Summary

- The typical $O(N^3)$ band calculations (collinear, non-collinear), DC method, DC-LNO method, and NEGF calculations in OpenMX 3.9 were computed on GPUs.
- These GPU calculations were somewhat faster, except for the NEGF calculation.
- In particular, for the DC-LNO method, the GPU calculations achieved a 4 to 5 times speedup in the comparison of diagonalization times compared to CPU calculations.

Future works

- Increase the processes offloaded to the GPU.
  - Currently, only a portion of the matrix diagonalization calculations are done on the GPU, but the goal is to perform all matrix diagonalization calculations on the GPU.
  - In the future, efforts will be made to utilize the GPU for tasks beyond matrix diagonalization as much as possible.
- Although related to the previous section, rewrite the code so that the OpenMP parallelization part is performed by the GPU as much as possible.
  - Only processes that the GPU is not good at should be done on the CPU.