Performance Characteristics of the Miyabi-G System with NVIDIA GH200 Grace-Hopper Superchip

Toshihiro Hanawa

JCAHPC /
Information Technology Center, The University of Tokyo

Norihisa Fujita Taisuke Boku

JCAHPC /
Center for Computational Sciences, University of Tsukuba

JCAHPC

http://jcahpc.jp/eng/index.html

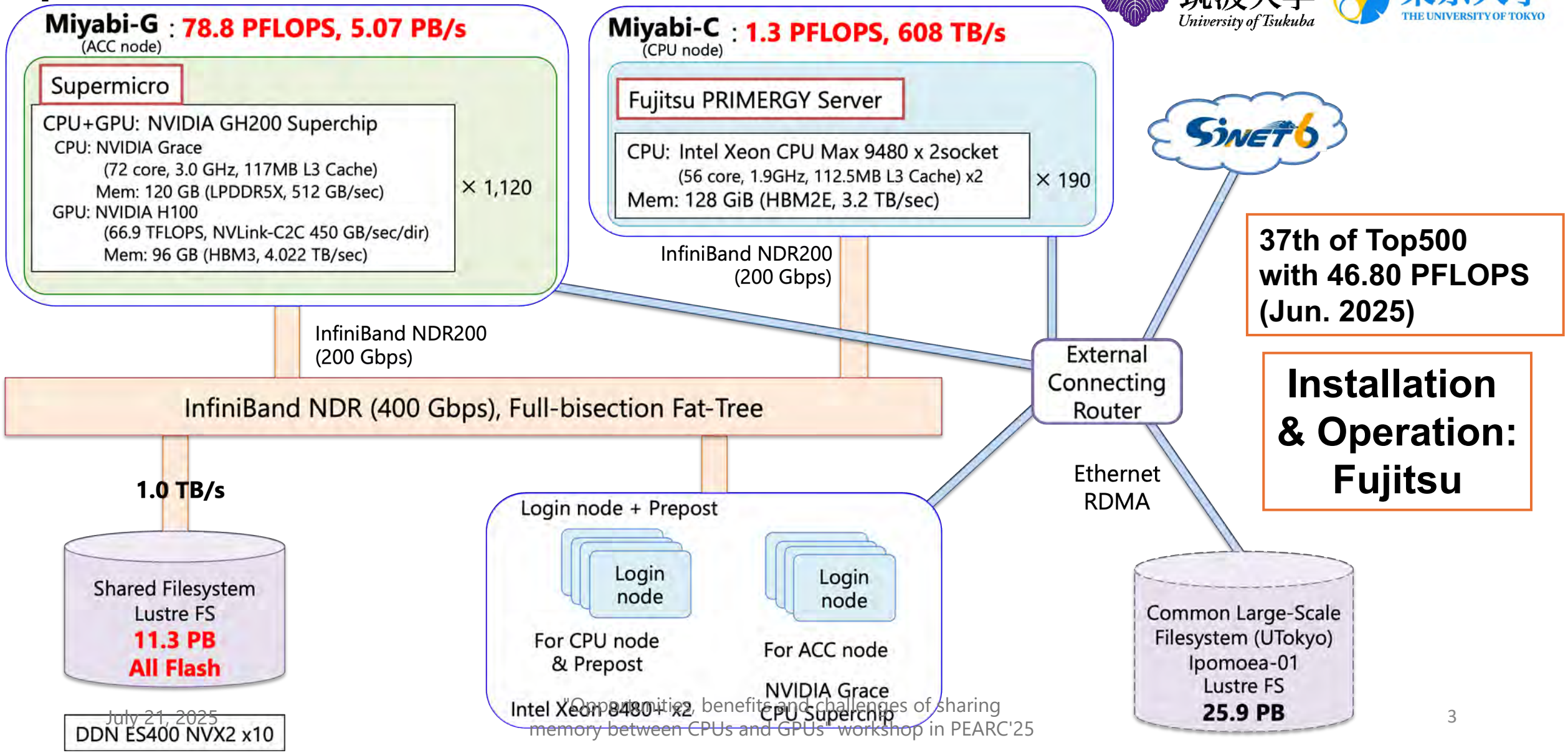
筑波大学 University of Tsukuba
東京大学 THE UNIVERSITY OF TOKYO
JCAHPC

- Joint Center for Advanced High Performance Computing, since 2013
  - University of Tsukuba & University of Tokyo
    - Budgets of 2 Centers are combined
  - Promotion on Computational Science, Design/Procurement/Operation of Large-scale Systems
- Oakforest-PACS (OFP), 1st System of JCAHPC
  - 8,208 Intel Xeon Phi, 25 PF, Fujitsu
  - Top 500 (#6 (Nov.2016), #1 in Japan)
    - Retired in the end of March 2022 (#39 (Nov.2021))
- National Flagship System (in fact) in FY.2019/2020
  - Between K and Fugaku

Miyabi (1/3)

Operation will start from Jan. 2025

JCAHPC
筑波大学 University of Tsukuba
東京大学 THE UNIVERSITY OF TOKYO



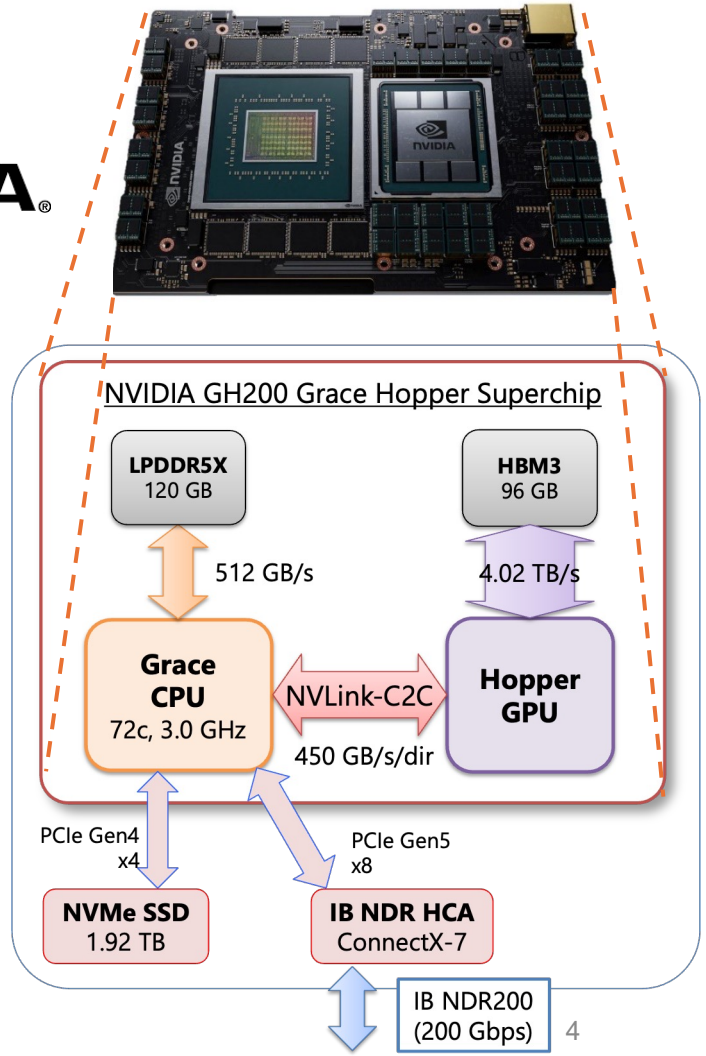
37th of Top500 with 46.80 PFLOPS (Jun. 2025)

Installation & Operation: Fujitsu

Miyabi (2/3)

Operation starts in January 2025

- **Miyabi-G: CPU+GPU: NVIDIA GH200**
  - Node: NVIDIA GH200 Grace-Hopper Superchip
    - Grace: 72c, 3.456 TF, 120 GB, 512 GB/sec (LPDDR5X)
    - H100: 66.9 TF DP-Tensor Core, 96 GB, 4,022 GB/sec (HBM3)
      - Cache Coherent between CPU-GPU
    - NVMe SSD for each GPU: 1.9TB, 8.0GB/sec, GPUDirect Storage
  - **Total (Aggregated Performance: CPU+GPU)**
    - **1,120 nodes, 78.8 PF, 5.07 PB/sec, IB-NDR 200**
- **Miyabi-C: CPU Only: Intel Xeon Max 9480 (SPR)**
  - Node: Intel Xeon Max 9480 (1.9 GHz, 56c) x 2
    - 6.8 TF, 128 GiB, 3,200 GB/sec (HBM2e only)
  - **Total**
    - **190 nodes, 1.3 PF, IB-NDR 200**
    - **372 TB/sec for STREAM Triad (Peak: 608 TB/sec)**



Miyabi (3/3)

Operation starts in **January 2025**

- **File System: DDN EXA Scalar, Lustre FS**
  - 11.3 PB (NVMe SSD) 1.0TB/sec, “Ipomoea-01” with 26 PB is also available
- **All nodes are connected with Full Bisection Bandwidth**
  - **(400Gbps/8)×(32×20＋16×1) = 32.8 TB/sec**
- **Operation starts in January 2025, h3-Open-SYS/WaitoIO will be adopted for communication between Acc-Group and CPU-Group**

IB-NDR（400Gbps）

IB-NDR200（200）

IB-HDR（200）

Miyabi-G
NVIDIA GH200 1,120
78.2 PF, 5.07 PB/sec

Miyabi-C
Intel Xeon Max
(HBM2e) 2 x 190
1.3 PF, 608 TB/sec

File System
DDN EXA Scaler
11.3 PB, 1.0TB/sec

Ipomoea-01
Common Shared Storage
26 PB

Miyabi

Performance Evaluation of System-Allocated Memory on NVIDIA GH200

Objective

- This research aims for memory and communication performance on GH200
- GH200 introduces a new unified memory (UM)
  - However, its performance has not been well studied yet
  - New UM has lower-overhead of CPU-GPU memory access than that of previous implementations
- We also evaluate performance of inter-node communication (MPI)
  - InfiniBand communication performance when data is located on the new UM
  - Especially, communication performance on UM if data is located on GPU-memory

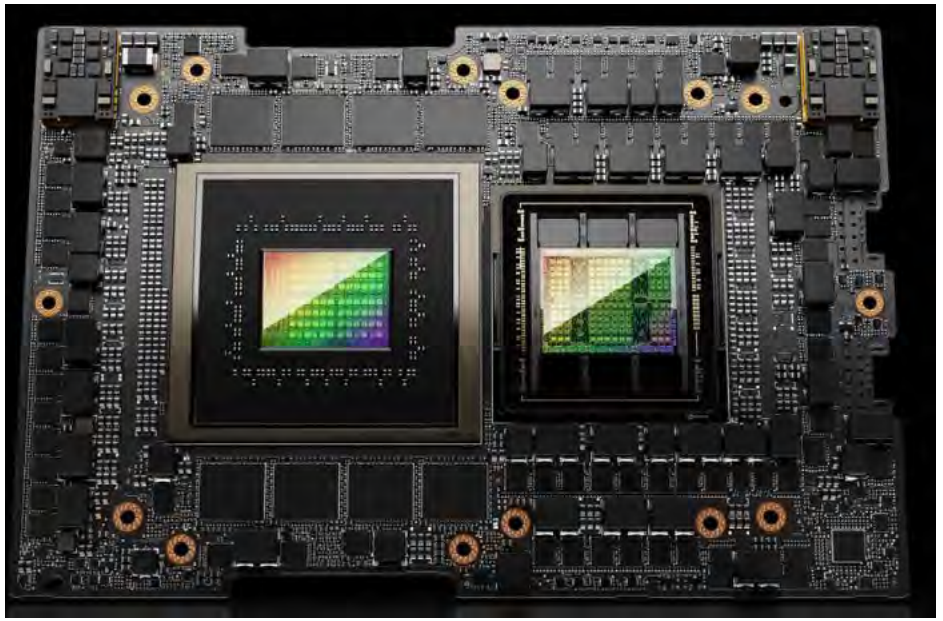
Related Work

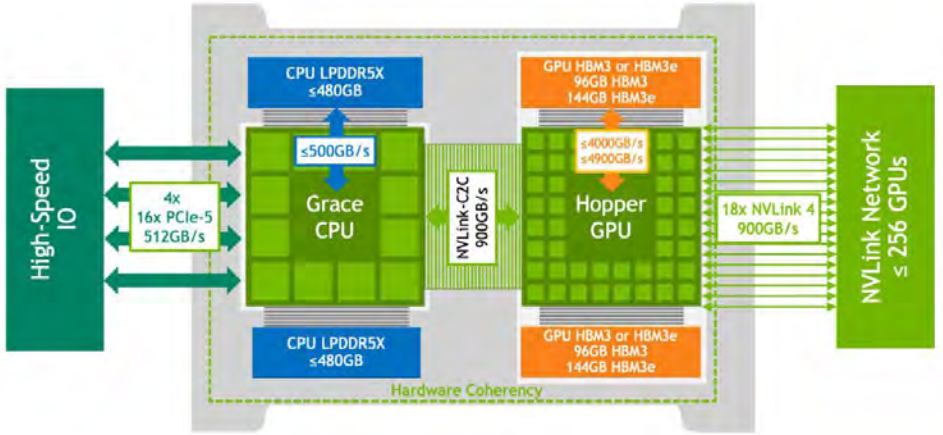
- Schieffer et. al [1]
  - Performance evaluation on GH200 memory system
  - Rodinia Benchmark and QC Simulation were used in the evaluation
  - The new Unified Memory on GH200 is faster than traditional Managed Memory
  - Easy to port applications to GH200
- Originality of this research
  - Performance evaluation on new system Miyabi-G
  - Communication performance evaluation using multiple GH200 nodes

[1] Gabin Schieffer, et. al., Harnessing Integrated CPU-GPU System Memory for HPC: a first look into Grace Hopper. In Proceedings of the 53rd International Conference on Parallel Processing (ICPP '24). Association for Computing Machinery, New York, NY, USA, 199–209. https://doi.org/10.1145/3673038.3673110

GH200

- GH200 is a module which tightly couples Grace CPU and Hopper GPU
  - However, CPU-memory and GPU-memory are separated
    - CPU: LPDDR5X 120GB, GPU: HBM3 96GB (Miyabi-G)
    - Differs from AMD's APU (MI300A)
      (In MI300A, CPU and GPU share the same HBM)
  - Cache-Coherent is maintained between CPU and GPU
- Proprietary bus (NVLink-C2C) connects CPU and GPU
  - Very high-bandwidth: 450GB/s (each direction)

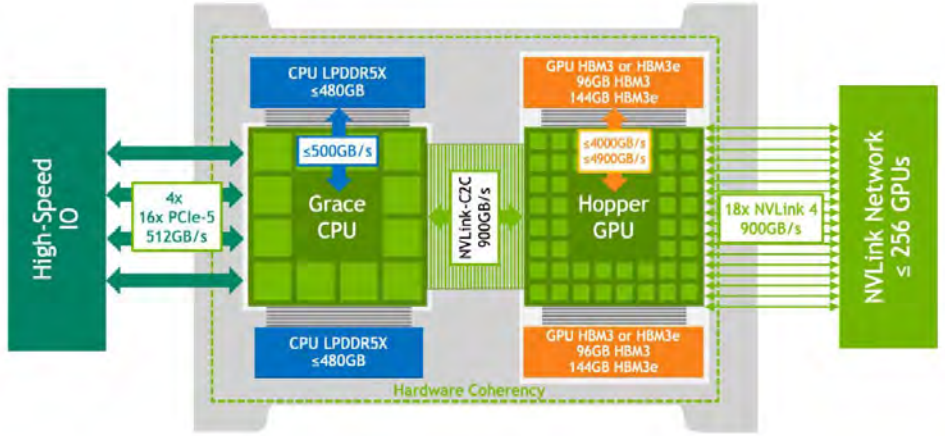




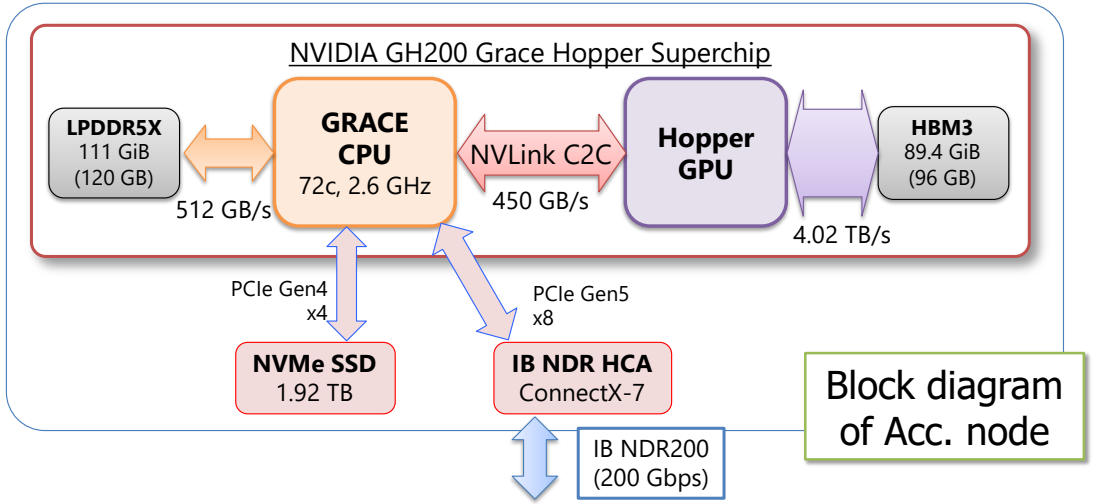
From Data Sheet:
https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip

GH200 Architecture

- Grace CPU
  - CPU architecture is ARM. Be careful about x86-dependent proprietary software
  - 128bit SVE SIMD, 3.4TFLOPS@3GHz
    - Applications designed for "Fugaku" A64FX can be ran on Grace as-is
    - The width of SVE is 128bit, which is different from A64FX. Performance may differ.
  - LPDDR5X Memory 120GB, 512GB/s
- Hopper GPU (H100)
  - 67 TFLOPS (DP, Tensor)
  - HBM3 Memory 4.02TB/s
  - Same CUDA core with SXM and PCIe H100
    - You can run CUDA or OpenACC applications without any change
  - NVLink-C2C is the same as PCIe bus in terms of application's view
    - NVLink-C2C is like a fast PCIe bus and cudaMemcpy() will be faster
    - NVLink-C2C@450GB/s is 7x faster than PCIe Gen.5 x16@64GB/s



From Data Sheet:
https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip



JCAHPC,
https://www.itc.u-tokyo.ac.jp/OFP-II/poster_OFP2-JCAHPC.pdf

GH200 and System Allocated Memory

- CPU and GPU share the same address space and cache-coherent
- They are treated as NUMA nodes in kernel
  - CPU is Node 0 and GPU is Node 1
  - APIs for CPU NUMA are also applicable
  - Memory characteristics is completely different in each NUMA domain
- System Allocated Memory (SAM)
  - Normally (Regularly) allocated memory
  - Both CPU and GPU can access and cache-coherent
  - Memory pages are allocated by first-tough policy (same as CPU NUMA)
  - Memory migration based on memory access from GPU
    - One way CPU memory -> GPU memory. Opposite direction is not observed
- You can use CUDA APIs as same as traditional environment

System Allocated Memory on GH200

- System Allocated Memory(SAM)
  - Normally (Regularly) allocated memory
    - malloc(), mmap(), new (C++), allocate (Fortran), etc...
- Page Size
  - Regular Page 64KB, Huge Page 512MB
    (4KB page is possible but NVIDIA recommends 64KB page)
  - Like false-sharing in caches, mixing CPU and GPU region in one page will have unexpected performance degradation
- Traditional methods (CUDA API) are also supported
  - Behavior is same as traditional environment
  - Page allocation is in fixed location and no migration will happen

Allocation API	Location	Migration	CPU Access	GPU Access
malloc(), etc.	First Touch	○	○	○
cudaMalloc()	GPU	×	×	○
cudaMallocHost()	CPU	×	○	○

SAM Page Migration

- When GPU accesses a page on CPU, the page may be migrated to GPU (Page Migration)
  - Each page has counter, and migration will happen in certain condition
  - Single access is not enough. Multiple accesses are required.
  - System automatically move data. Application does not require any action.
  - GPU⇔LPDDR5X@450GB/s is improved to GPU⇔HBM3@4TB/s. Next access will be faster.
- Opposite direction (GPU to CPU) migration is not supported
  - We don’t observe migration from GPU to CPU trigged by memory access
  - No detailed information about page migration in official documents
  - More investigation is future work

Memory Copy Benchmark

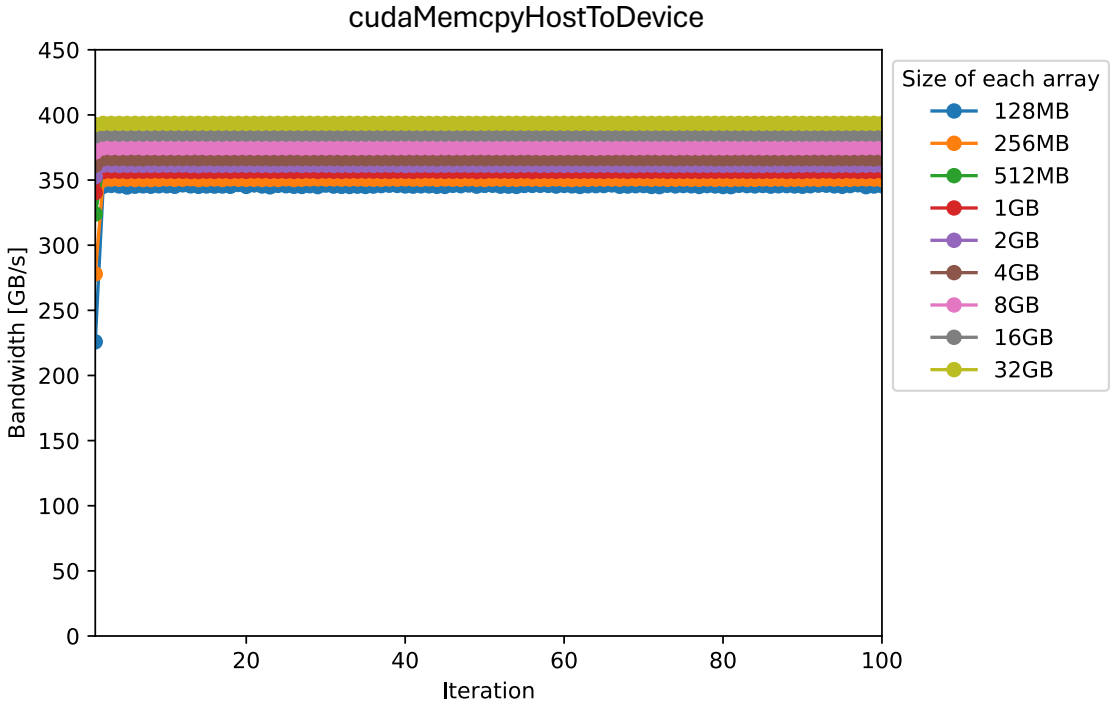
- Memory copy benchmark is used to evaluate memory system performance of GH200 (A[i]=B[i])
  - Combination of parameters: Memory allocation, page location, copy processor and copy method

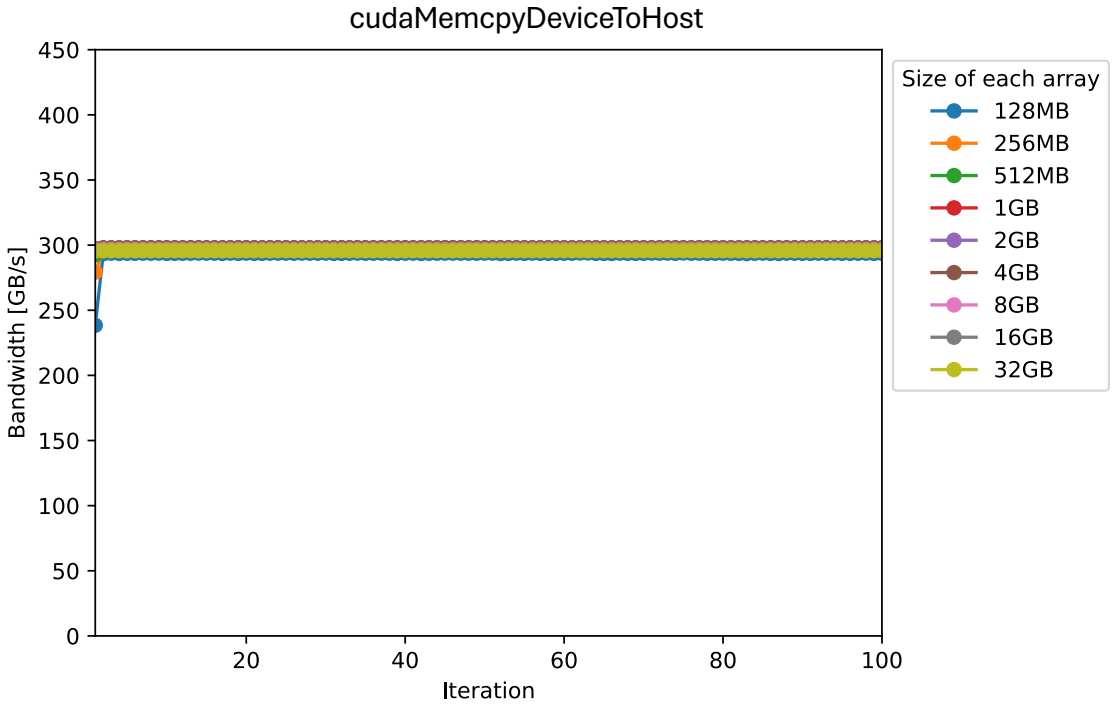
Memory Allocation Method	SAM, cudaMalloc(), cudaMallocHost()
Memory Page Location	LPDDR5X, HBM3
Copy Processor	CPU, GPU
Copy Method (API)	OpenMP CPU, CUDA Kernel, CUDA API

- For SAM, first-touch is performed before benchmark to ensure the entire array is located either LPDDR5X or HBM3
- Copy all elements = 1 iteration. 100 iterations are measured in total
  - To check performance fluctuations and changes by migration
- Hereafter, we call fixed memory allocation by like cudaMalloc() as “Traditional”

NVLink-C2C Performance

- NVLink-C2C performance evaluation using CUDA API
  - Performance with traditional ways (no SAM) (cudaMalloc+cudaMemcpy)
  - HostToDevice is up to 400GB/s, DeviceToHost is up to 300GB/s
  - No performance fluctuations and stable for each size
  - Using 1 CUDA stream (serial execution)
    - If they are multiplexed, performance would be better
    - → Up to bandwidth of NVLink-C2C
  - Much faster than PCIe bus

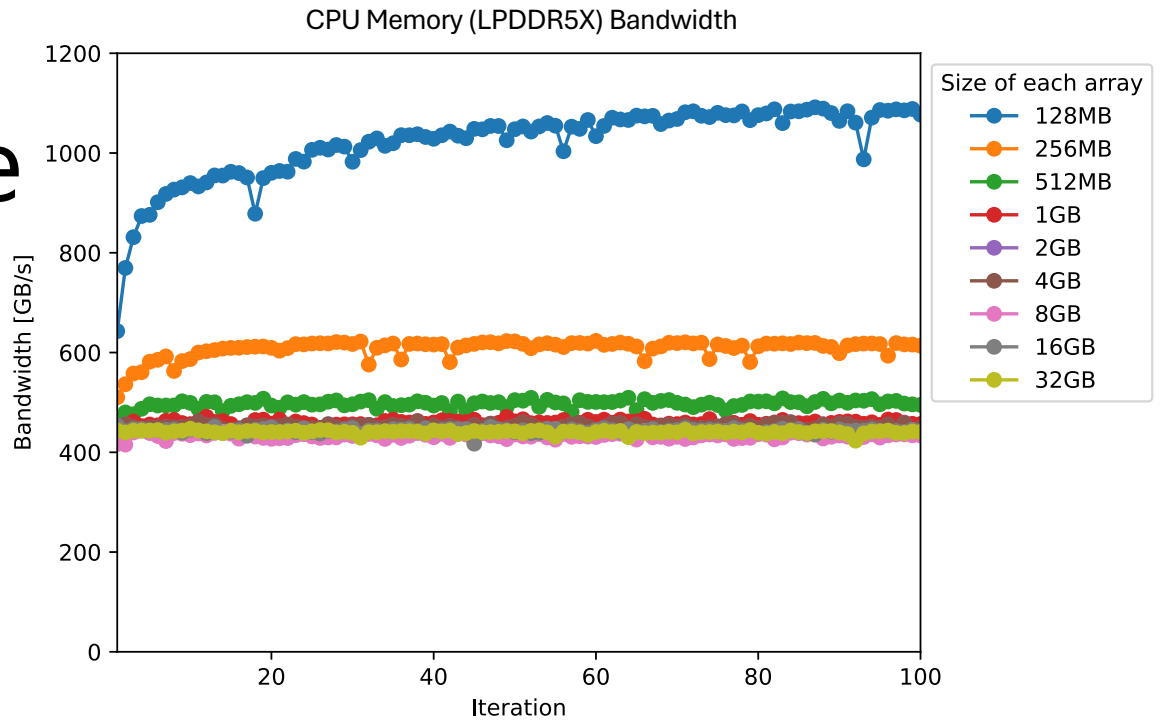


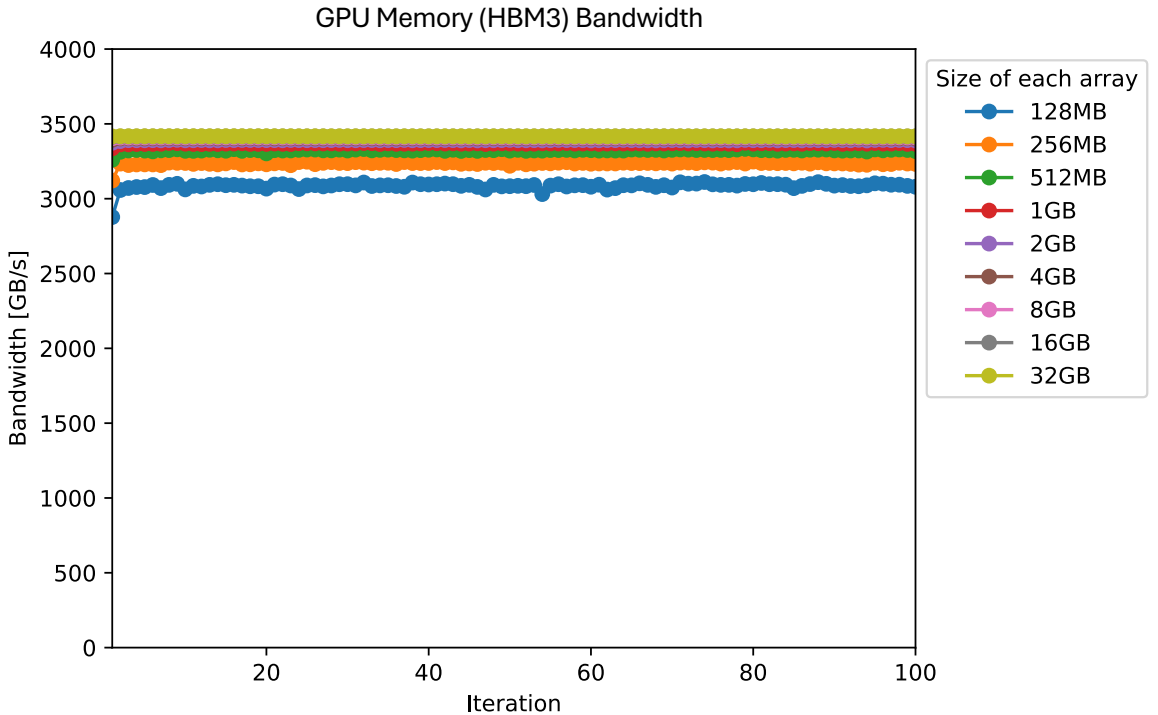


"Opportunities, benefits and ch
memory between CPUs and GPUs"

GH200 Memory Performance

- CPU Memory Bandwidth
  - About 400GB/s~420GB/s
  - Faster performance on small cases may be affected by the cache
- GPU Memory Bandwidth
  - Larger array higher performance, up to 3.4TB/s
  - No performance fluctuations and stable for each size
- NVLink-C2C is 400GB/s of bandwidth
  - LPDDR5X and HBM3 have same order performance from CPU perspective

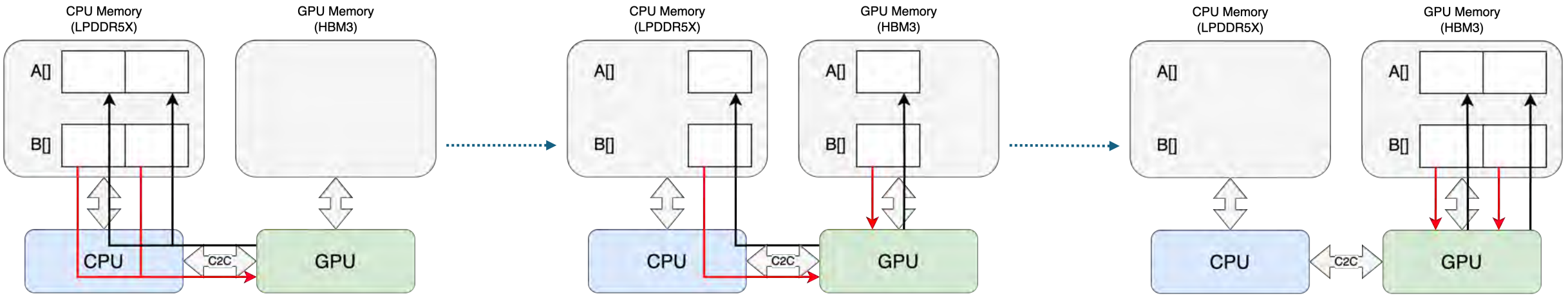




"Opportunities, benefits and cha
memory between CPUs and GPUs"

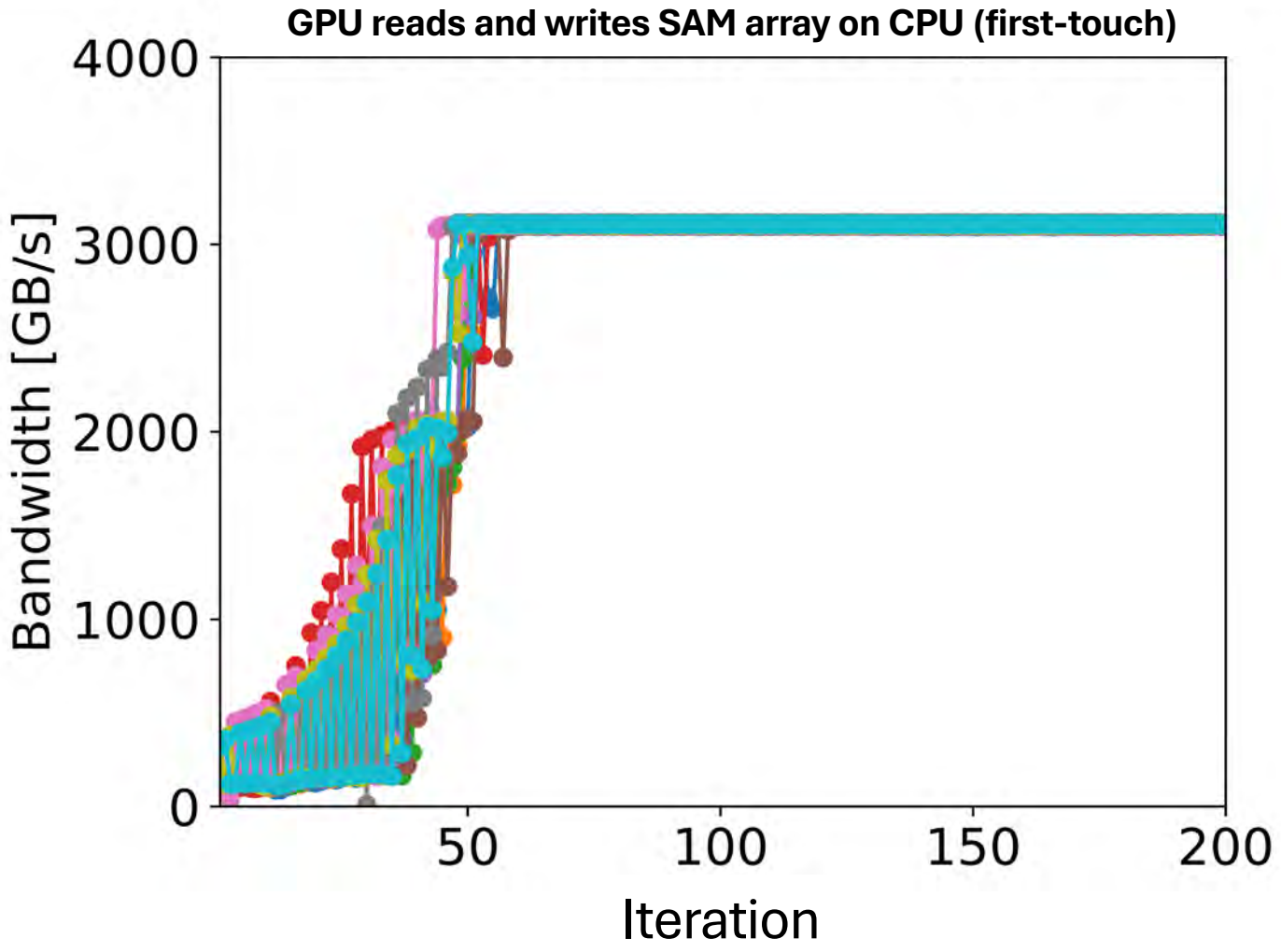
Migration Performance

- GPU copies between arrays allocated on CPU memory
  - Initial: all elements are LPDDR5-to-LPDDR5 copy by GPU → lower performance
  - Partially migrated: some elements are HBM3-to-HBM3 → gradually improving performance
  - All migrated: all elements are HBM3-to-HBM3 copy by GPU → best performance



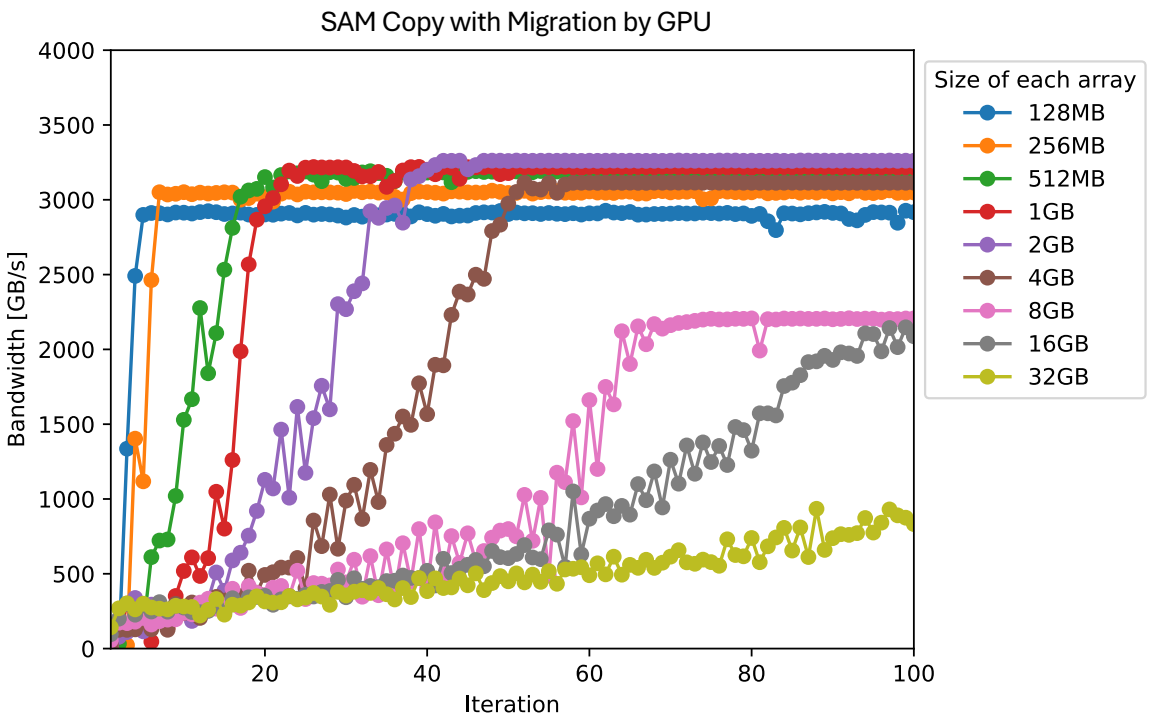
Migration Result (4GB data size)

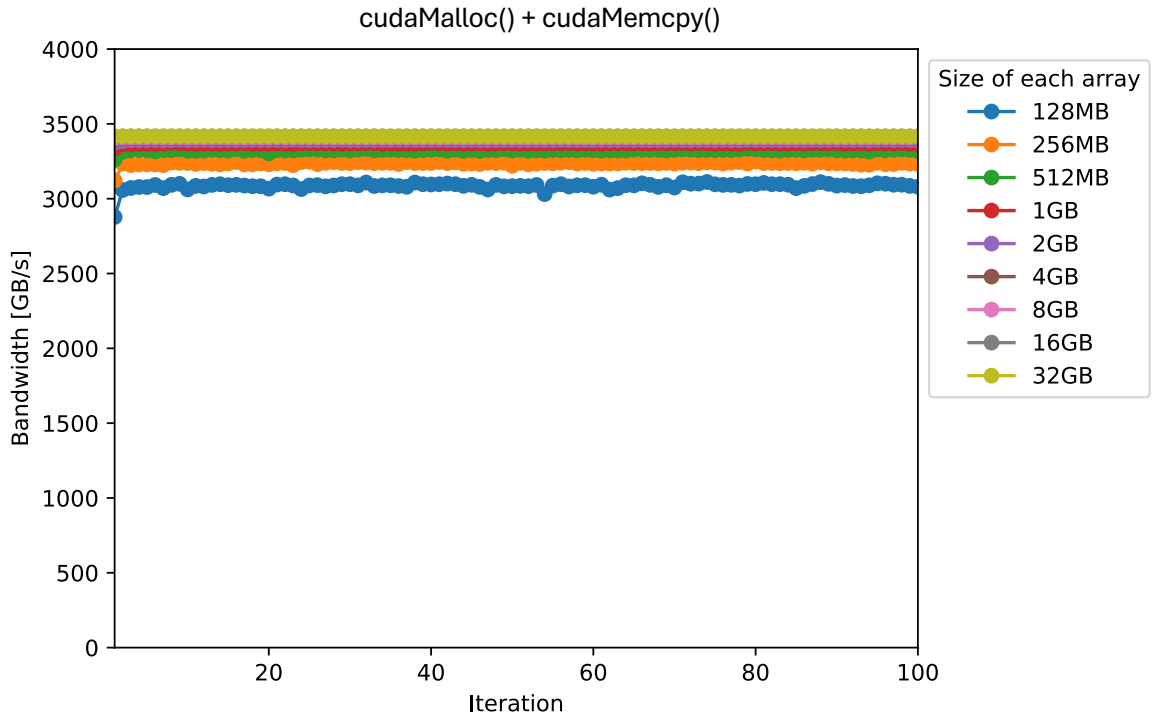
- Run benchmark 200 iterations and measure performance for each
  - 10 executions from start to show variance
  - malloc() + first touch, reside on CPU initially
  - GPU reads and writes arrays
  - Performance improves over iterations and close to cudaMalloc() on HBM3
  - → Pages are migrated gradually
- Performance after migration has completed is worse than native HBM3 access



Migration Result (various data size)

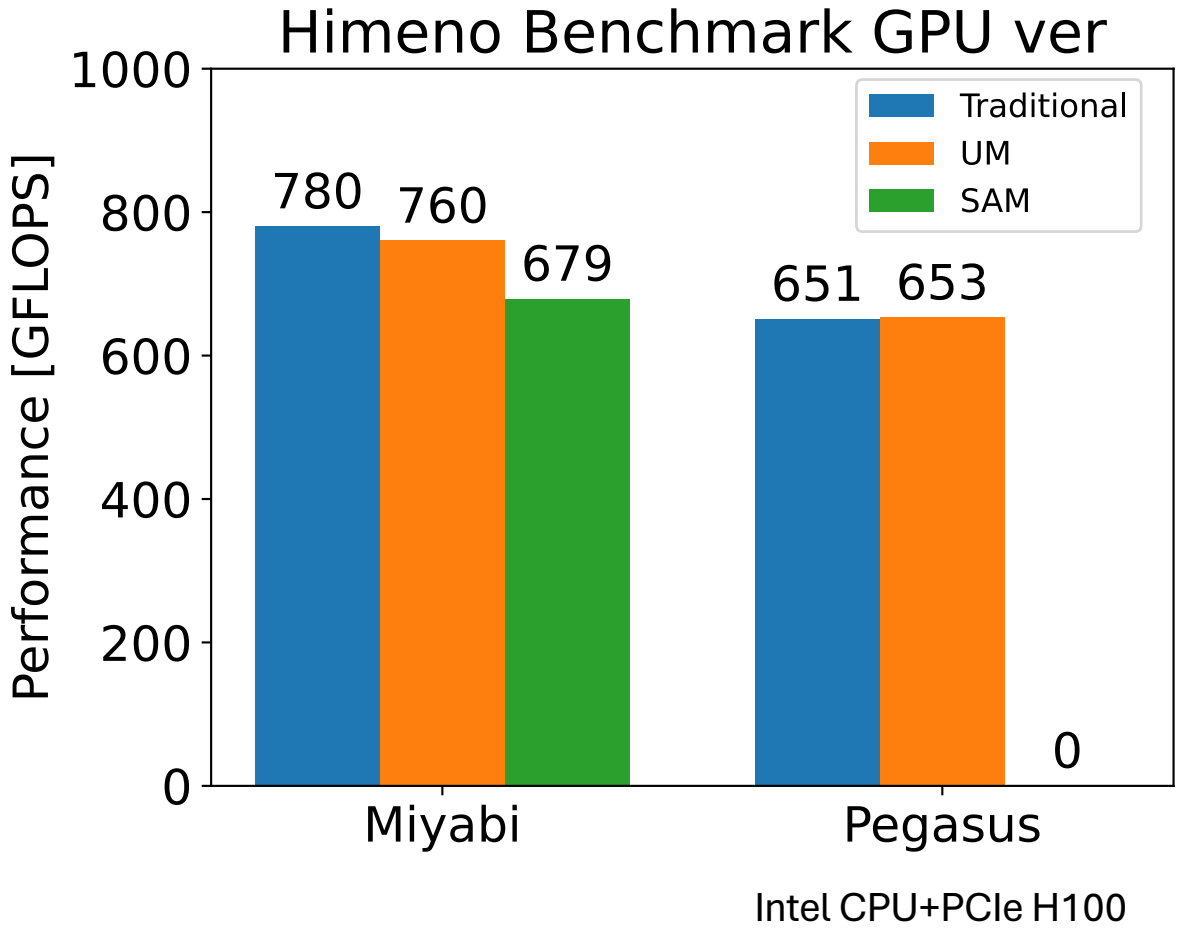
- 100 iterations of copy, each performance is plotted (128MB to 32GB)
  - Time to migrate is roughly proportional to array size
- Especially, in large array (e.g., 8GB=Pink), performance after migration is worse than small cases
  - GPU-side TLB misses may degrade performance
  - Performance using HugePage is future work





Himeno Benchmark

- Himeno Benchmark
  - Solves Poisson equation using Jacobi iterative method
  - Memory bandwidth bound
- SAM is applied to benchmark
  - Problem Size XL (512x512x1024)
  - 3000 Iteration (fixed)
    - We consider SAM migration is completed
- SAM performance underperforms Traditional
  - Lower memory bandwidth of SAM may affect the performance, same as the copy benchmark
- SAM advantage is that no memory management is required
  - CUDA program can take advantages
  - If we use SAM and OpenACC, we can program GPUs with the same complexity as OpenMP CPU



SAM and MPI Communication

- NVIDIA provides GPUDirect (GDR) for direct communication between GPUs
  - Many MPI implementations support GDR for direct data transfer
  - GDR has been developed for PCIe-connected GPU environment
- GDR assumes GPU (isolated) memory over PCIe
  - We observed behavior on GH200 and found that GDR is not applicable for SAM
  - Even if GDR is disabled, SAM on GPU memory is still communicable
- Both SAM on CPU and on GPU can be transferred without GDR
  - Communication library must support GH200 architecture
    - If communication library does not consider SAM,
      We find out that communication behavior is not optimal for GH200
    - → All pages are migrated to CPU from GPU while communication

SAM and UCX

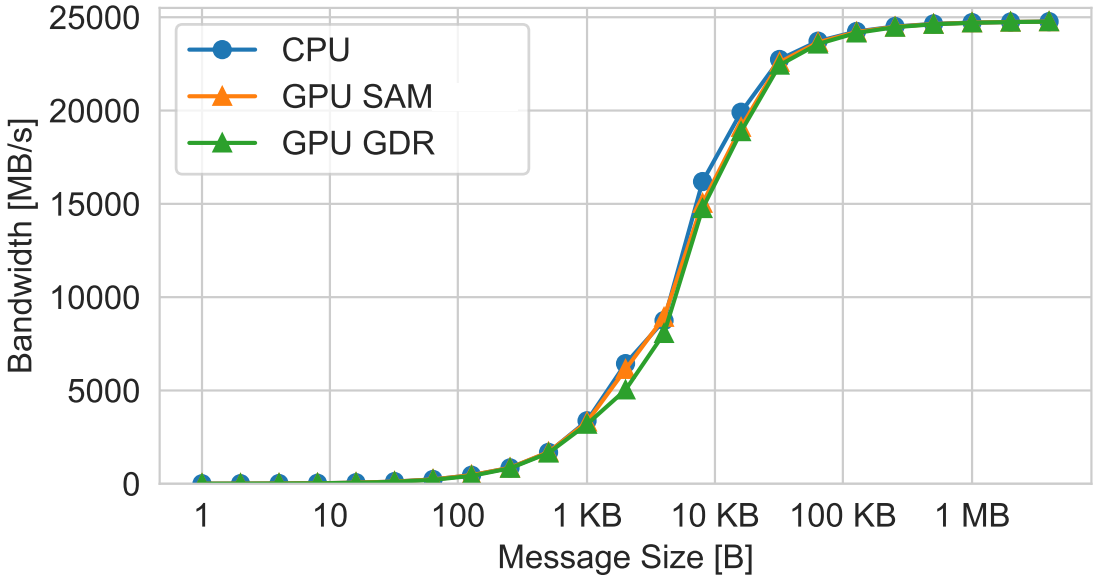
- OpenMPI and its derivatives uses Unified Communication X(UCX)
  - Communication capability and characteristics are from UCX
  - “UCX can handle SAM” ≡ “MPI can handle SAM”
- NVIDIA InfiniBand supports SAM
  - InfiniBand On-Demand Paging (ODP) is key feature on GH200
  - ODP enables to keep pages on GPU while communication
    - Without ODP, pages move to CPU while communication, which is not desirable on GH200
  - UCX configuration to enable ODP on SAM (host) memory
    - Environment Variable `UCX_REG_NONBLOCK_MEM_TYPES`=“`host`”
    - Using ucx.conf bundled with UCX 1.17 or later applies this configuration
    - ucx_info command to show the value

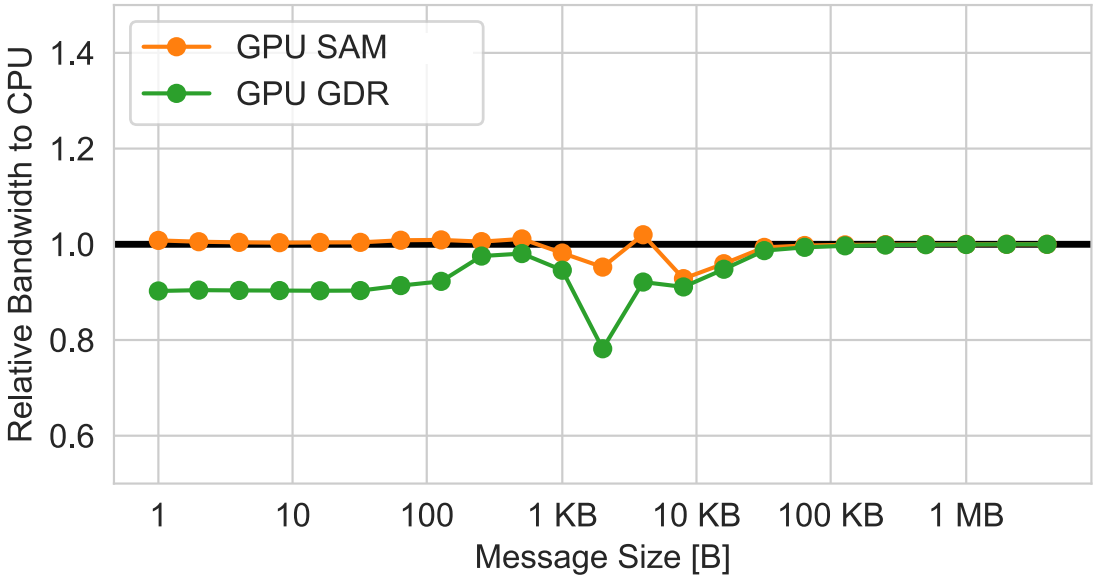
osu_bw Evaluation

- Performance evaluation using `osu_bw` from OSU Micro Benchmark 7.5
  - It supports GDR, but does not support SAM
  - We modified CPU measurement code to support SAM on GPU memory
- MPC-X MPI bundled with NVIDIA HPC SDK 24.9
  - OpenMPI-based MPI, UCX backend
  - ODP support is enabled in UCX
- 3 evaluation patterns
  - CPU ODP: CPU-to-CPU, ODP
  - GPU SAM-ODP: GPU-to-GPU, SAM, ODP
  - GPU GDR: GPU-to-GPU, cudaMalloc(), GPUDirect
- Tuning
  - `Set UCX_RNDV_THRESH="intra:auto,inter:4096"`
  - Improve bandwidth between 4-16KB on GH200

osu_bw Result

- No difference for >= 128KB, and GPU memory is fast as CPU memory
  - In most cases, we don't need to take care of memory allocation method and memory page location
- GPU GDR is 10% slower than GPU SAM
  - It seems that overhead comes from management cost of GDR region
- GPU SAM is comparable performance to CPU
  - Better than GDR on small cases
  - Eager communication on GPU SAM memory has lower overhead than GDR memory
- Note: not all MPIs use UCX
  - I tested with NVIDIA MPC-X MPI (UCX backend)
    - Other MPIs and communication libraries needs to be verified
    - MVAPICH2, GASNet, ...
  - NVIDIA InfiniBand supports SAM via Verbs API
    - Other network interfaces like Slingshot needs to be verified





Summary and Future Work for SAM analysis

- We evaluated SAM performance on GH200
  - Fast communication over NVLink-C2C, cache-coherent between CPU and GPU, and page migration contribute to efficient data sharing between CPU and GPU
  - However, SAM migration has performance issue in some cases
    - It appears TLB misses degrade performance, but further analysis is future work
  - Inter-node communication has same or higher performance than traditional method
- Future Work
  - Detailed analysis of page migration
  - Evaluation of various communication patterns
  - Non UCX-based MPIs
  - Performance evaluation of SAM on practical applications
    - with OpenACC/OpenMP (without data directives)
    - It's difficult to estimate SAM performance right now
  - Performance comparison with AMD MI300A APU

Improvement of power efficiency in GH200

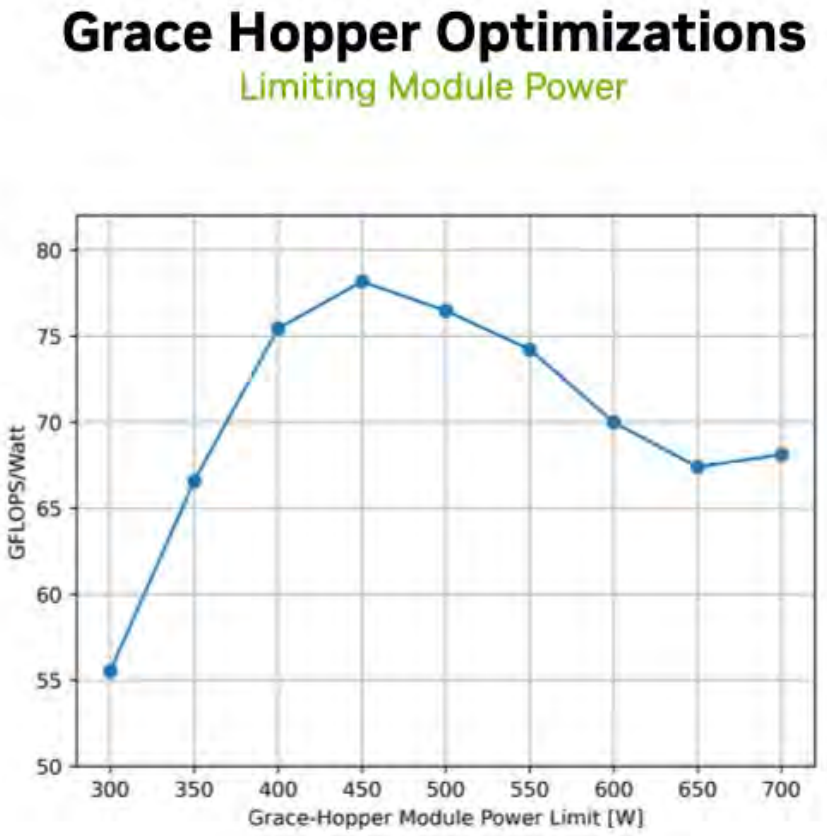
- So far, Miyabi-G operates without power capping for GH200.
  - Default: 900W for GH200 package, 300W for Grace, 700W for Hopper
- By NVIDIA, the best power efficiency is around 450W (!!); However, we do not want to increase the execution time of user jobs.

➔ Is there any way to make it work both ways?

- CPU Frequency Governor could be used to adjust the balance between CPU and GPU power budget.

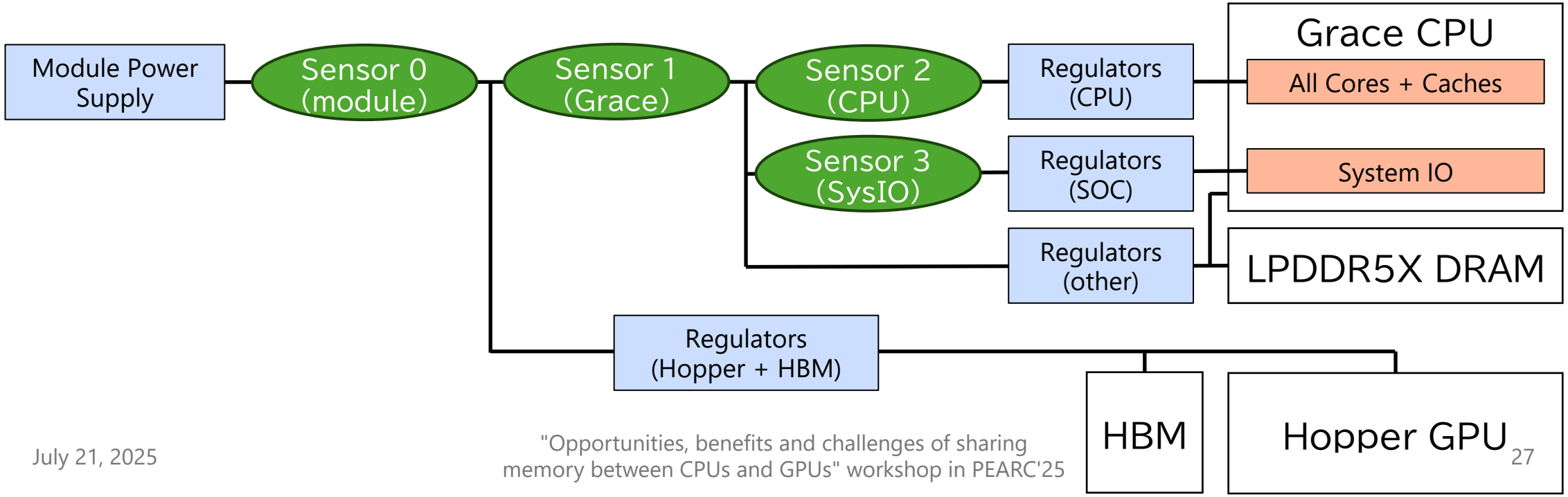
- Related work
  - Julita Corbalan, “EAR: Energy management framework for HPC”
    - Barcelona Supercomputer Center, LRZ SuperMUC
  - Anna Yue et al., “EVeREST: An Effective and Versatile Runtime Energy Saving Tool for GPUs,” PPoPP’25

  ➔ Will try in the future



Power measurement points in GH200

- Read by sensor driver in Arm core
- Module Power : `/sys/class/hwmon/hwmon1/device/power1_average`
- Grace Power: `/sys/class/hwmon/hwmon2/device/power1_average`
  - `Similarly, hwmon3 ➔ Sensor2, hwmon4 ➔ Sensor3`

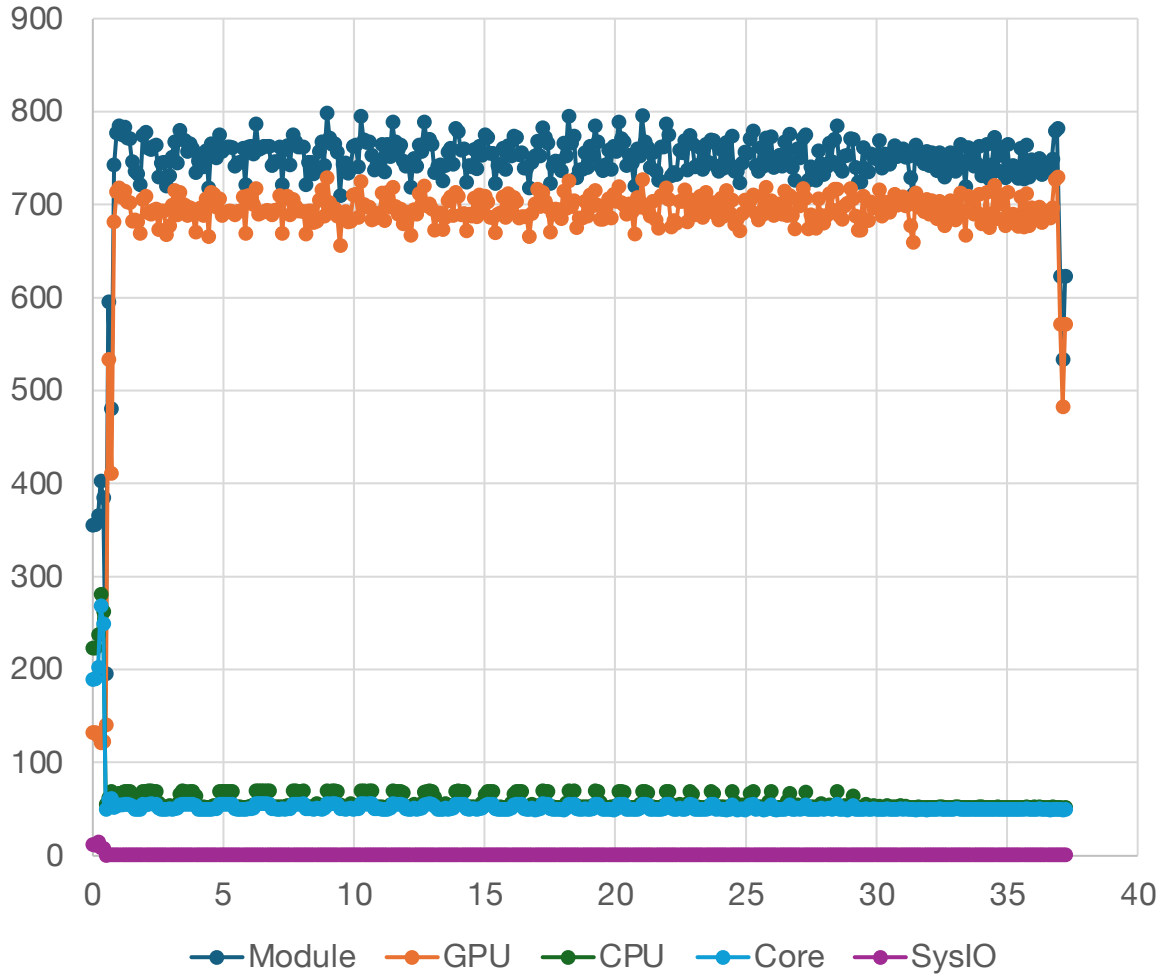


HPL @ GH200, CPU governor: performance (900W-700W)

- `HPL_OOC_MODE=1, N=141312`
- `50.58 TFLOPS`
- `27.73 kJ`

➔ avg power 745.1 W

➔ 67.89 GF/W

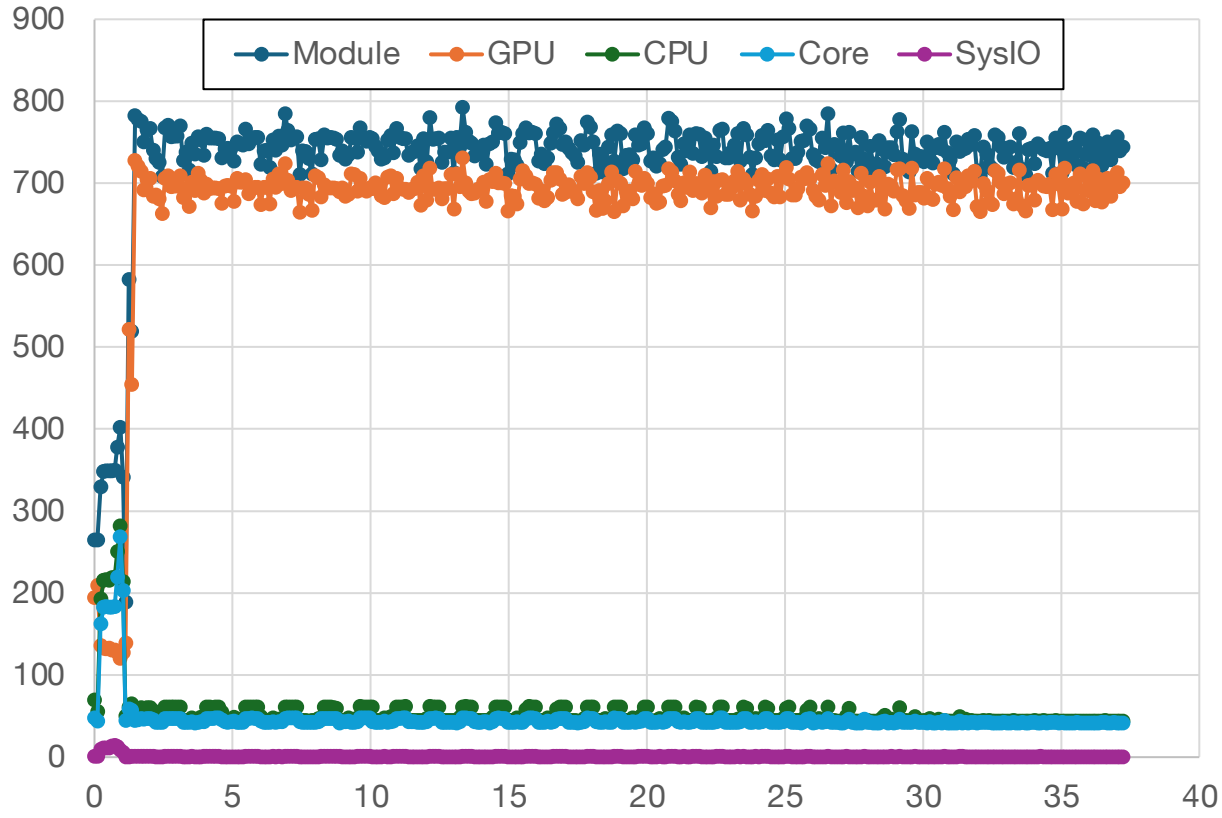


HPL @ GH200, CPU governor: ondemand

- `HPL_OOC_MODE=1, N=141312`
- `50.57 TFLOPS`
- `27.21 kJ`

➔ avg power 731.1 W

➔ 69.17 GF/W

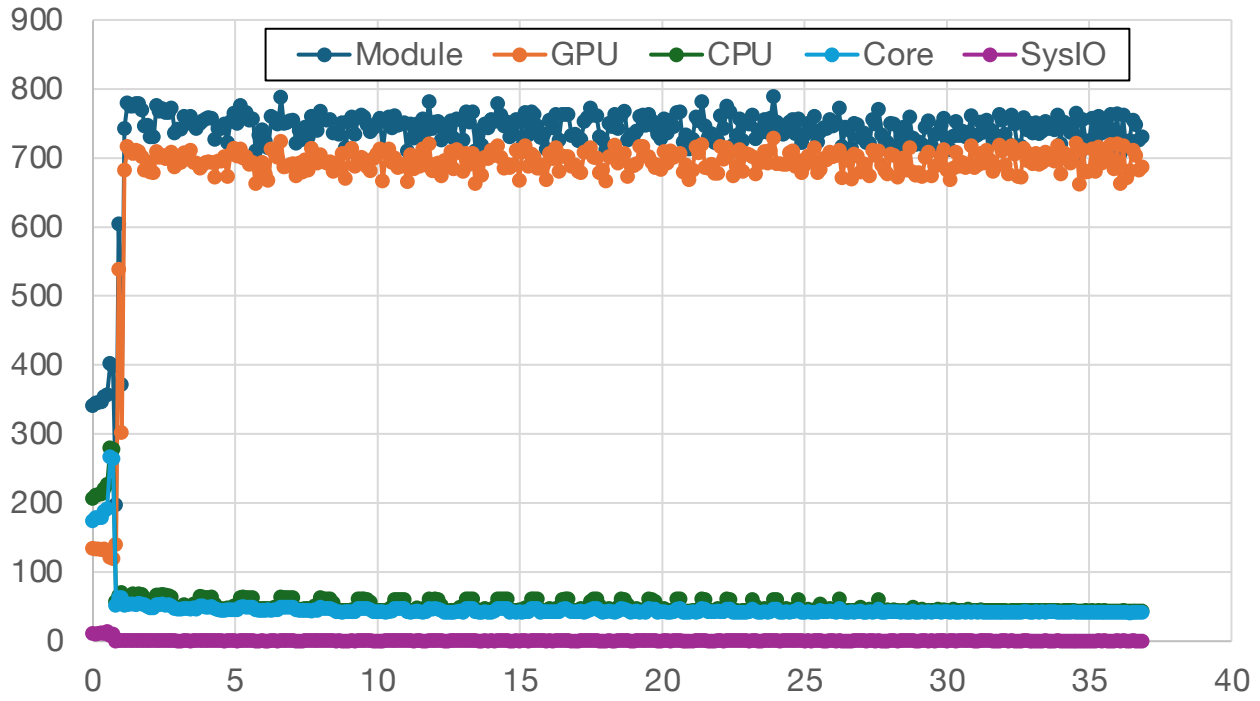


Ondemand with optimizing parameters

- /sys/devices/system/cpu/cpufreq/ondemand/up_threshold : 95➔80
  - Raise the frequency earlier.
- /sys/devices/system/cpu/cpufreq/ondemand/sampling_down_factor: 1➔100
  - Less likely to drop frequency
- HPL_OOC_MODE=1, N=141312
- 50.63 TFLOPS
- 27.29 kJ

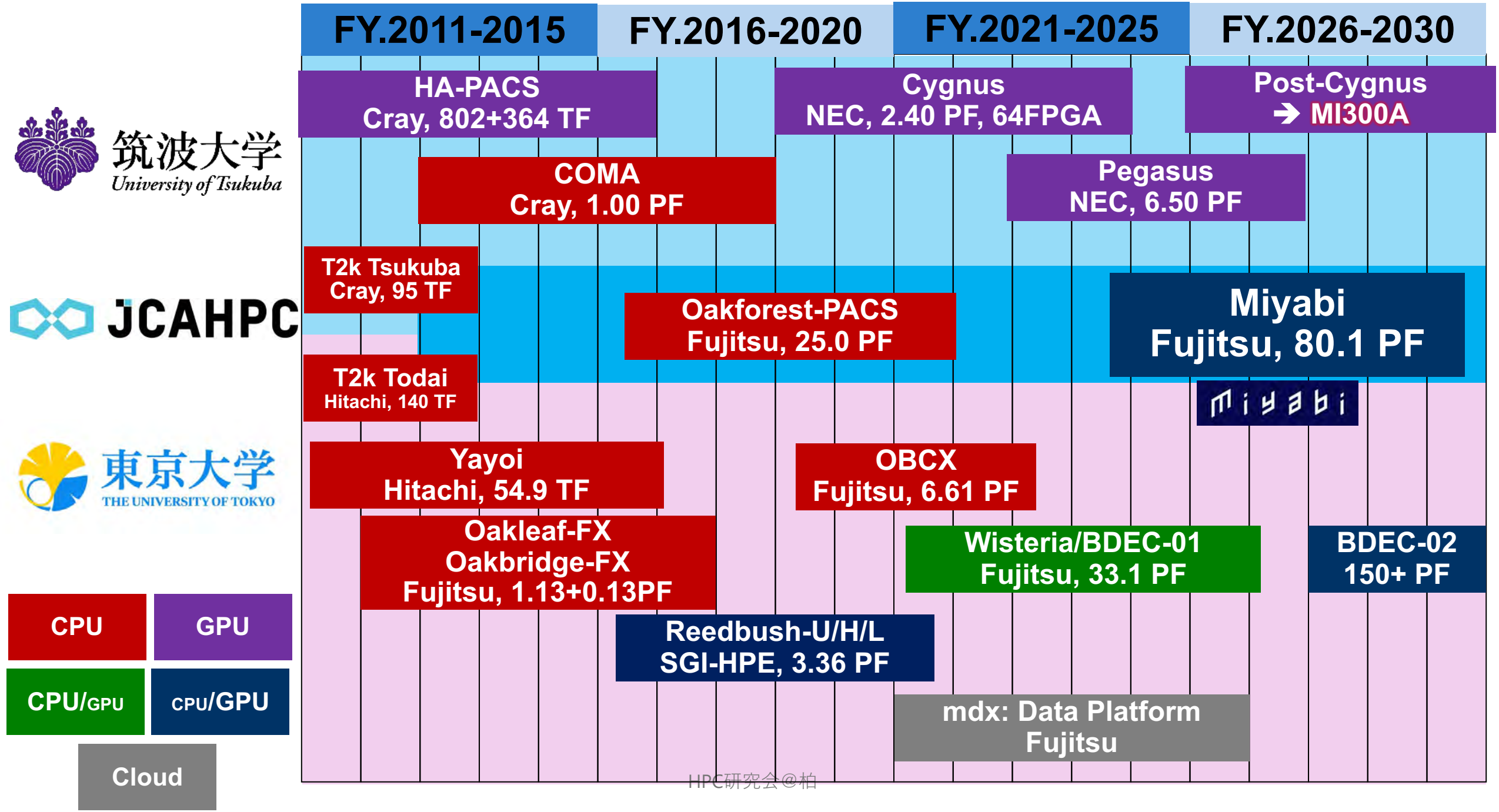
➔ avg power 733.8 W

➔ 68.99 GF/W



Summary & Future work

- Performance evaluation for System Allocated Memory: SAM
  - U Tsukuba will also introduce **MI300A system** March 2026 (?)
- Power optimization: Checking for optimal settings unique to CG1 (1 socket/node)
  - Control by CPU Frequency governor, ondemand + parameter change looks good
  - Better performance for Green500
    - To obtain good performance without setting special conditions
  - Evaluation using practical applications
- Efficient use of local storage by NVMe SSD on each compute node
  - Hopper GPU can access thru Grace CPU to SSD easily
    - Once a file is mmaped by Grace CPU, and pointer can be used from Hopper GPU directly


FY.2011-2015
FY.2016-2020
FY.2021-2025
FY.2026-2030
筑波大学
University of Tsukuba
JCAHPC
東京大学
THE UNIVERSITY OF TOKYO
HA-PACS
Cray, 802+364 TF
Cygnus
NEC, 2.40 PF, 64FPGA
Post-Cygnus
➔ MI300A
COMA
Cray, 1.00 PF
Pegasus
NEC, 6.50 PF
T2k Tsukuba
Cray, 95 TF
Oakforest-PACS
Fujitsu, 25.0 PF
Miyabi
Fujitsu, 80.1 PF
T2k Todai
Hitachi, 140 TF
Yayoi
Hitachi, 54.9 TF
OBCX
Fujitsu, 6.61 PF
Oakleaf-FX
Oakbridge-FX
Fujitsu, 1.13+0.13PF
Wisteria/BDEC-01
Fujitsu, 33.1 PF
BDEC-02
150+ PF
Reedbush-U/H/L
SGI-HPE, 3.36 PF
mdx: Data Platform
Fujitsu
CPU
GPU
CPU/GPU
CPU/GPU
Cloud

Unification of OpenACC/OpenMP target

- Directive-based implementation (for easier GPU porting)
  - OpenACC: virtually for NVIDIA GPU (except for HPE Cray compiler)
  - OpenMP target: for NVIDIA/AMD/Intel GPUs, fewer function/docs
- Is unification of the interfaces possible?
  - Yes, we can unify the interfaces by using preprocessor macros
  - Miki & Hanawa (2024, IEEE Access)
    - https://github.com/ymiki-repo/solomon
  - Basis: `_Pragma()`-style directive
  - Supported backends:
    - OpenACC, OpenMP target, OpenMP
- Which style do users prefer?
  - Normal/messy implementation ➡️
  - ⬇️ Simplified one

```
OFFLOAD(AS_INDEPENDENT, NUM_THREADS(NTHREADS))
for (int32_t i = 0; i < N; i++) {
```

```
#ifdef OFFLOAD_BY_OPENACC
#pragma acc kernels vector_length(NTHREADS)
#pragma acc loop independent
#endif  // OFFLOAD_BY_OPENACC
#ifdef OFFLOAD_BY_OPENMP_TARGET
#ifdef OFFLOAD_BY_OPENMP_TARGET_LOOP
#pragma omp target teams loop
thread_limit(NTHREADS)
#else  // OFFLOAD_BY_OPENMP_TARGET_LOOP
#pragma omp target teams distribute parallel
for simd thread_limit(NTHREADS)
#endif  // OFFLOAD_BY_OPENMP_TARGET_LOOP
#endif  // OFFLOAD_BY_OPENMP_TARGET
  for (int32_t i = 0; i < N; i++) {
```