Miki & Hanawa (2024, *IEEE Access*, 12, 181644) https://github.com/ymiki-repo/solomon

Solomon: unified schemes for directive-based GPU offloading

Yohei Miki
(Information Technology Center, The University of Tokyo)

CExA Kokkos Tea Time, 2025/04/16

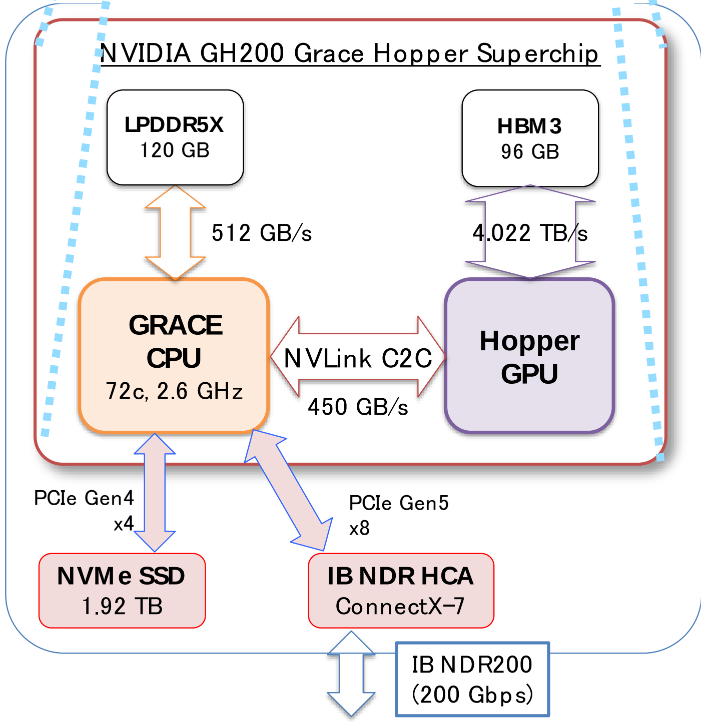
GPU-accelerated supercomputers

- GPU is the dominant accelerator on recent supercomputers
  - GPU is a parallel computer equipped with many cores
- Majority of GPU supercomputers are equipped with NVIDIA GPUs; however
  - Rise of AMD GPUs (El Capitan & Frontier: #1 & #2 in TOP500)
  - Intel GPU-powered systems (Aurora: #3 in TOP500)
- Increased competition among GPU vendors
  - Drives performance improvement of NVIDIA GPUs:
    - $\sim\sqrt{2}$x in each generation between P100, V100, and A100
    - ~ 3x in H100
  - Announcement of new GPU sometimes includes the release of additional features (e.g., new function)
  - We need to find out how to optimize for each GPU

Miyabi (1/2)

Operation starts in **January 2025**

- **<u>Miyabi-G: CPU+GPU: NVIDIA GH200</u>**
  - 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**
- **<u>Miyabi-C: CPU Only: Intel Xeon Max 9480 (SPR)</u>**
  - Node: Intel Xeon Max 9480 (1.9 GHz, 56c) x 2
    - 6.8 TF, 128 GiB, 3,200 GB/sec (<u>HBM2e only</u>)
  - **Total**
    - **190 nodes, 1.3 PF, IB-NDR 200**
    - **372 TB/sec for STREAM Triad (Peak: 608 TB/sec)**



Miyabi (2/2)

Operation starts in **January 2025**

- **<u>File System: DDN EXA Scalar, Lustre FS</u>**
  - 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
All Flash Storage

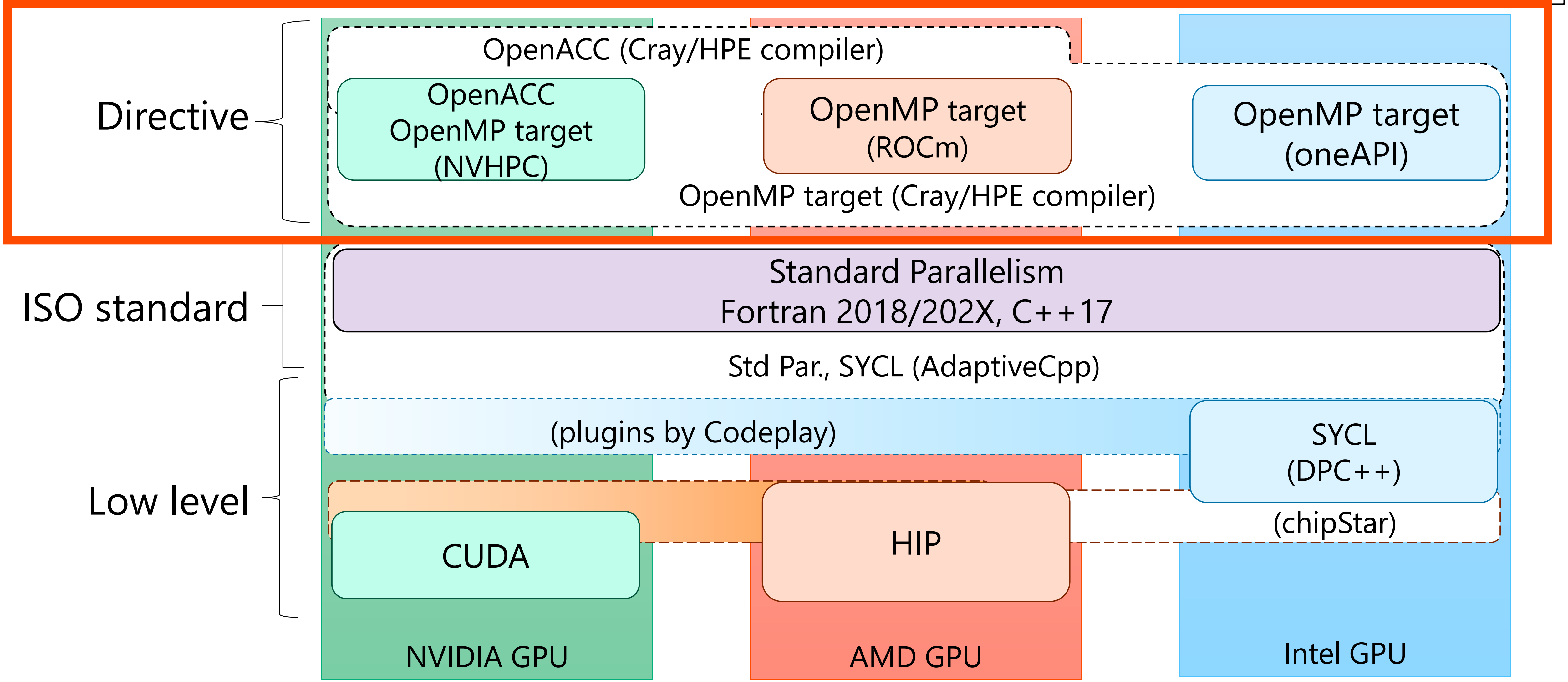
Ipomoea-01
Common Shared Storage
26 PB

What should we do for our users?

- Majority of our users: MPI+OpenMP hybrid parallelization
- Shortest path to GPU computing (for them) is directives
  - Top priority: most users can use GPU (faster than CPU, at least)
  - (Performance maximization is the next step)
- Toward vendor-neutral GPU computing
  - Currently, all GPU-equipped supercomputers in Japan adopt NVIDIA GPUs ➔ OpenACC is the majority in Japan
    - OpenACC is virtually for NVIDIA GPUs (= not vendor-neutral)
  - QST/NIFS will shortly introduce AMD MI300A accelerated system
  - Post Fugaku (aka FugakuNEXT) will equip accelerator devices
- What we should do for our users:
  - Provide an easy way to enable vendor-neutral GPU computing for directive (mostly OpenACC) users
    - Learning cost should be minimized: adoption of OpenACC-like notation?

Programming environments for GPU

Taken from
https://www.pccluster.org/ja/event/data/240205_pccc_wsAI-HPC-OSS_06_hanawa-miki.pdf



Directive-based GPU-offloading methods

- OpenACC
  - First choice for directives (better function and rich documents)
  - Virtually for NVIDIA GPUs owing to the acquisition of PGI by NVIDIA
  - AMD and Intel would not support OpenACC
    - Exception: HPE Cray compiler supports OpenACC for AMD GPU
    - Intel has developed a source-to-source translator (OpenACC to OpenMP target)
- OpenMP target
  - OpenMP 4.0+ supports offloading on accelerator devices
  - OpenMP 5.0+ supports loop directives (OpenACC-like implementation)
  - Supported by all GPU vendors (NVIDIA, AMD, and Intel)
  - (Currently) lacks a part of the functions provided in OpenACC
    - e.g., sophisticated control of multiple (asynchronous) queues
- Developers must select OpenACC or OpenMP target for their code implementation

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
  - 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++) {
```

Overview/benefits of Solomon

- We have developed a header-only library named “Solomon” (Simple Off-LOading Macros Orchestrating multiple Notations)
    - Miki & Hanawa (2024, *IEEE Access*, 12, 181644)
    - Available at https://github.com/ymiki-repo/solomon
- Inputs are converted to OpenACC or OpenMP target directives
    - Users can select backend (OpenACC or OpenMP target) by specifying the compilation flags
    - Users can use OpenACC on NVIDIA GPU and OpenMP target on AMD/Intel GPUs
    - Easy performance comparison of OpenACC and OpenMP target
- Users can use compilers by GPU-manufacturing vendors as is
    - Users will benefit from the improvement of the vendors’ compilers
    - Solomon is like a directive-version HIP
- Modification by user side is also easy
    - Solomon is aggregate of preprocessor macros, so that you can edit

Representative directives in Solomon

- Intuitive notation, OpenACC/OpenMP-like notation

input	output	backend
OFFLOAD(...)		
PRAGMA_ACC_KERNELS_LOOP(...)	_Pragma("acc kernels __VA_ARGS__") _Pragma("acc loop __VA_ARGS__")	OpenACC (kernels)
PRAGMA_ACC_PARALLEL_LOOP(...)	_Pragma("acc parallel __VA_ARGS__") _Pragma("acc loop __VA_ARGS__")	OpenACC (parallel)
PRAGMA_OMP_TARGET_TEAMS_LOOP(...)	_Pragma("omp target teams loop __VA_ARGS__")	OpenMP (loop)
PRAGMA_OMP_TARGET_TEAMS_DISTRIBUTE_PARALLEL_FOR(...)	_Pragma("omp target teams distribute parallel for __VA_ARGS__")	OpenMP (distribute)
MALLOC_ON_DEVICE(...)		
PRAGMA_ACC_ENTER_DATA_CREATE(...)	_Pragma("acc enter data create(__VA_ARGS__)")	OpenACC
PRAGMA_OMP_TARGET_ENTER_DATA_MAP_ALLOC(...)	_Pragma("omp target enter data map(alloc: __VA_ARGS__)")	OpenMP
FREE_FROM_DEVICE(...)		
PRAGMA_ACC_EXIT_DATA_DELETE(...)	_Pragma("acc exit data delete(__VA_ARGS__)")	OpenACC
PRAGMA_OMP_TARGET_EXIT_DATA_MAP_DELETE(...)	_Pragma("omp target exit data map(delete: __VA_ARGS__)")	OpenMP
MEMCPY_D2H(...)		
PRAGMA_ACC_UPDATE_HOST(...)	_Pragma("acc update host(__VA_ARGS__)")	OpenACC
PRAGMA_OMP_TARGET_UPDATE_FROM(...)	_Pragma("omp target update from(__VA_ARGS__)")	OpenMP
MEMCPY_H2D(...)		
PRAGMA_ACC_UPDATE_DEVICE(...)	_Pragma("acc update device(__VA_ARGS__)")	OpenACC
PRAGMA_OMP_TARGET_UPDATE_TO(...)	_Pragma("omp target update to(__VA_ARGS__)")	OpenMP
DATA_ACCESS_BY_DEVICE(...)		
PRAGMA_ACC_DATA(...)	_Pragma("acc data __VA_ARGS__")	OpenACC
PRAGMA_OMP_TARGET_DATA(...)	_Pragma("omp target data __VA_ARGS__")	OpenMP

Representative clauses in Solomon

- Intuitive notation, OpenACC/OpenMP-like notation
  - Users can combine three types of notations for clauses even in a single directive

input intuitive notation	 OpenACC/OpenMP-like notation	output	backend
AS_INDEPENDENT	ACC_CLAUSE_INDEPENDENT	independent	OpenACC
	OMP_TARGET_CLAUSE_SIMD	simd	OpenMP
NUM_THREADS(n)	ACC_CLAUSE_VECTOR_LENGTH(n)	vector_length(n)	OpenACC
	OMP_TARGET_CLAUSE_THREAD_LIMIT(n)	thread_limit(n)	OpenMP
COLLAPSE(n)	ACC_CLAUES_COLLAPSE(n)	collapse(n)	OpenACC
	OMP_TARGET_CLAUSE_COLLAPSE(n)		OpenMP
REDUCTION(...)	ACC_CLAUSE_REDUCTION(...)	reduction(__VA_ARGS__)	OpenACC
	OMP_TARGET_CLAUSE_REDUCTION(...)		OpenMP
AS_ASYNC(...)	ACC_CLAUSE_ASYNC(...)	async(__VA_ARGS__)	OpenACC
	OMP_TARGET_CLAUSE_NOWAIT	nowait	OpenMP

How to switch backends

- Compilation flags to switch backend directives

compilation flag	backend
-DOFFLOAD_BY_OPENACC	OpenACC (kernels)
-DOFFLOAD_BY_OPENACC -DOFFLOAD_BY_OPENACC_PARALLEL	OpenACC (parallel)
-DOFFLOAD_BY_OPENMP_TARGET	OpenMP target (loop)
-DOFFLOAD_BY_OPENMP_TARGET -DOFFLOAD_BY_OPENMP_TARGET_DISTRIBUTE	OpenMP target (distribute)

- Flags to activate OpenACC or OpenMP target are also required
  - `-DOFFLOAD_BY_OPENACC` is automatically disabled when OpenACC is disabled
- If both of `-DOFFLOAD_BY_OPENACC` and `-DOFFLOAD_BY_OPENMP_TARGET` are specified
  - Solomon exploits OpenACC for GPU offloading
- If neither `-DOFFLOAD_BY_OPENACC` nor `-DOFFLOAD_BY_OPENMP_TARGET` is specified
  - Solomon sets OpenMP backend and performs thread-parallelization for multicore CPUs (e.g., `_Pragma("omp parallel for")`)

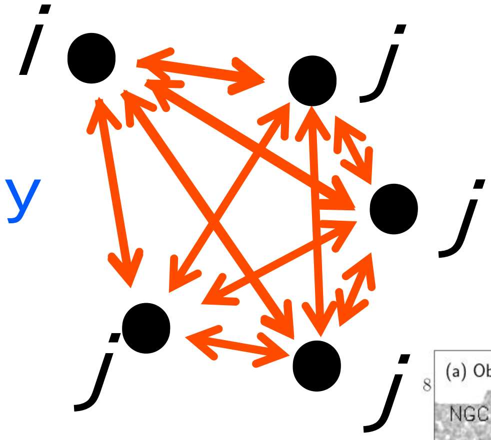
Precautions and recommendations

- Indication of optional clauses: comma-separated notation
  - Solomon automatically drops invalid inputs for each directive
  - To realize the below transformation (OpenMP-like to OpenACC-like)
    - `_Pragma("omp target teams loop collapse(3) thread_limit(128)")`
      ➔ `_Pragma("acc kernels vector_length(128)") _Pragma("acc loop collapse(3)")`
    - Solomon adequately assigns the optional clauses for each directive
- Recommended style: `OFFLOAD(…)` or `PRAGMA_ACC_[KERNELS PARALLEL]_LOOP(…)` or `PRAGMA_OMP_TARGET_...`
  - Adequate transformation to OpenMP target is difficult for separated style (`PRAGMA_ACC_[KERNELS PARALLEL](…)` + `PRAGMA_ACC_LOOP(…)`)
- `AS_INDEPENDENT` must be the head of all optional inputs
  - Corresponding output in OpenMP (`simd`) is not a clause and is a part of the combined construct
  - Insertion of another clause ahead of `simd` causes an error
- Use of `PRAGMA_OMP_TARGET_DATA(…)` is deprecated
  - `data`, `host_data` in OpenACC ⇄ `data` in OpenMP target
  - Difficulty in transforming to OpenACC
  - Recommend: `PRAGMA_ACC_[DATA HOST_DATA](…)`, `DATA_ACCESS_BY_[DEVICE HOST](…)`

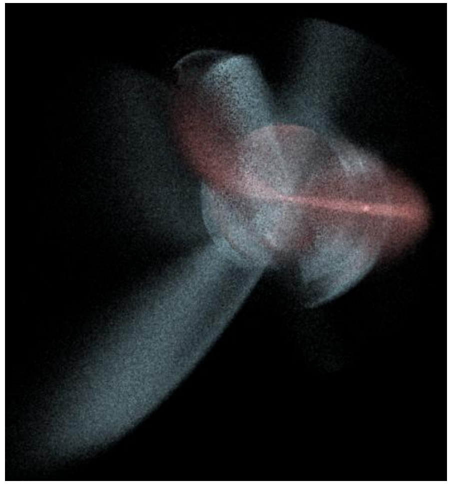
Target application: N-body simulation

- Calculating time evolution of the gravitational many-body system by solving Newton's equation of motion
  - Data: $\mathcal{O}(N)$
  - Force calculation: $\mathcal{O}(N^2)$
  - Time integration: $\mathcal{O}(N)$

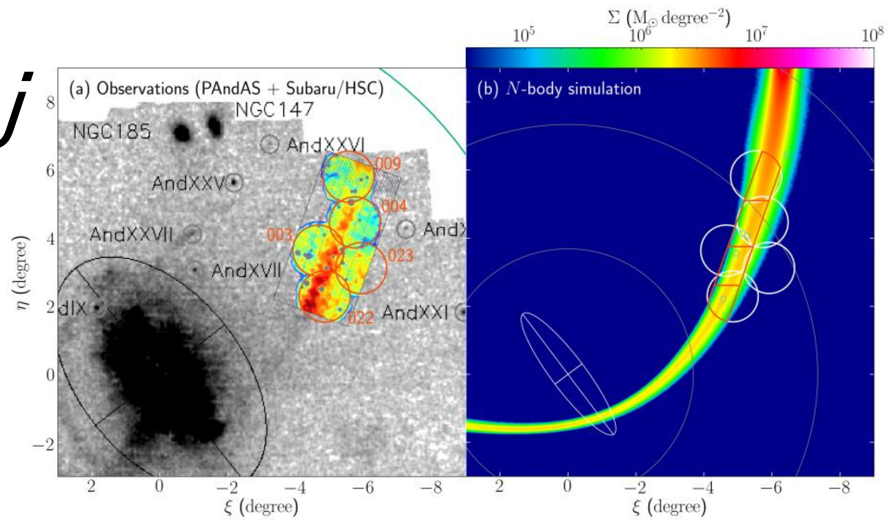
$$\boldsymbol{a}_i = \sum_{\substack{j=0 \\ j \neq i}}^{N-1} \frac{G m_j \left(\boldsymbol{x}_j - \boldsymbol{x}_i\right)}{\left(\left|\boldsymbol{x}_j - \boldsymbol{x}_i\right|^2 + \epsilon^2\right)^{3/2}}$$



- Terminology
  - i-particle: particle feels gravity
  - j-particle: particle causes gravity



- Direct solver is
  - a backend of tree method
  - employed in collisional N-body simulations
- Note: we used FP32 arithmetics
  - Sufficient for collisionless systems



Example in intuitive notation

- Generate initial condition on CPU, then compute on GPU

```
set_uniform_sphere(num, pos, vel, Mtot, rad, virial, newton);
MEMCPY_H2D(pos [0:num], vel [0:num])
calc_acc(num, pos, acc, num, pos, eps);
```

  - `MEMCPY_H2D()`: data transfer from CPU to GPU

- Outline of gravity solver

```
void calc_acc(⋯) {
  OFFLOAD(AS_INDEPENDENT, NUM_THREADS(NTHREADS))
  for (std::remove_const_t<decltype(Ni)> i = 0; i < Ni; i++) {
    // initialization
    PRAGMA_ACC_LOOP(ACC_CLAUSE_SEQ)
    for (std::remove_const_t<decltype(Nj)> j = 0; j < Nj; j++) {
      // main computation
    }
    iacc[i] = ai;
  }
}
```

- Parallelize i-loop (outer loop)
  - `NTHREADS` threads
- Serialize j-loop (inner loop) for performance (remove additional atomic operations)

Example in OpenACC-like notation

- Generate initial condition on CPU, then compute on GPU

```
set_uniform_sphere(num, pos, vel, Mtot, rad, virial, newton);
PRAGMA_ACC_UPDATE_DEVICE(pos [0:num], vel [0:num])
calc_acc(num, pos, acc, num, pos, eps);
```

  - `PRAGMA_ACC_UPDATE_DEVICE()` : data transfer from CPU to GPU

- Outline of gravity solver

```
void calc_acc(⋯) {
  PRAGMA_ACC_KERNELS_LOOP(ACC_CLAUSE_INDEPENDENT, ACC_CLAUSE_VECTOR_LENGTH(NTHREADS))
  for (std::remove_const_t<decltype(Ni)> i = 0; i < Ni; i++) {
    // initialization
    PRAGMA_ACC_LOOP(ACC_CLAUSE_SEQ)
    for (std::remove_const_t<decltype(Nj)> j = 0; j < Nj; j++) {
      // main computation
    }
    iacc[i] = ai;
  }
}
```

Example in OpenMP-like notation

- Generate initial condition on CPU, then compute on GPU

```
set_uniform_sphere(num, pos, vel, Mtot, rad, virial, newton);
PRAGMA_OMP_TARGET_UPDATE_TO(pos [0:num], vel [0:num])
calc_acc(num, pos, acc, num, pos, eps);
```

  - `PRAGMA_OMP_TARGET_UPDATE_TO()` : data transfer from CPU to GPU

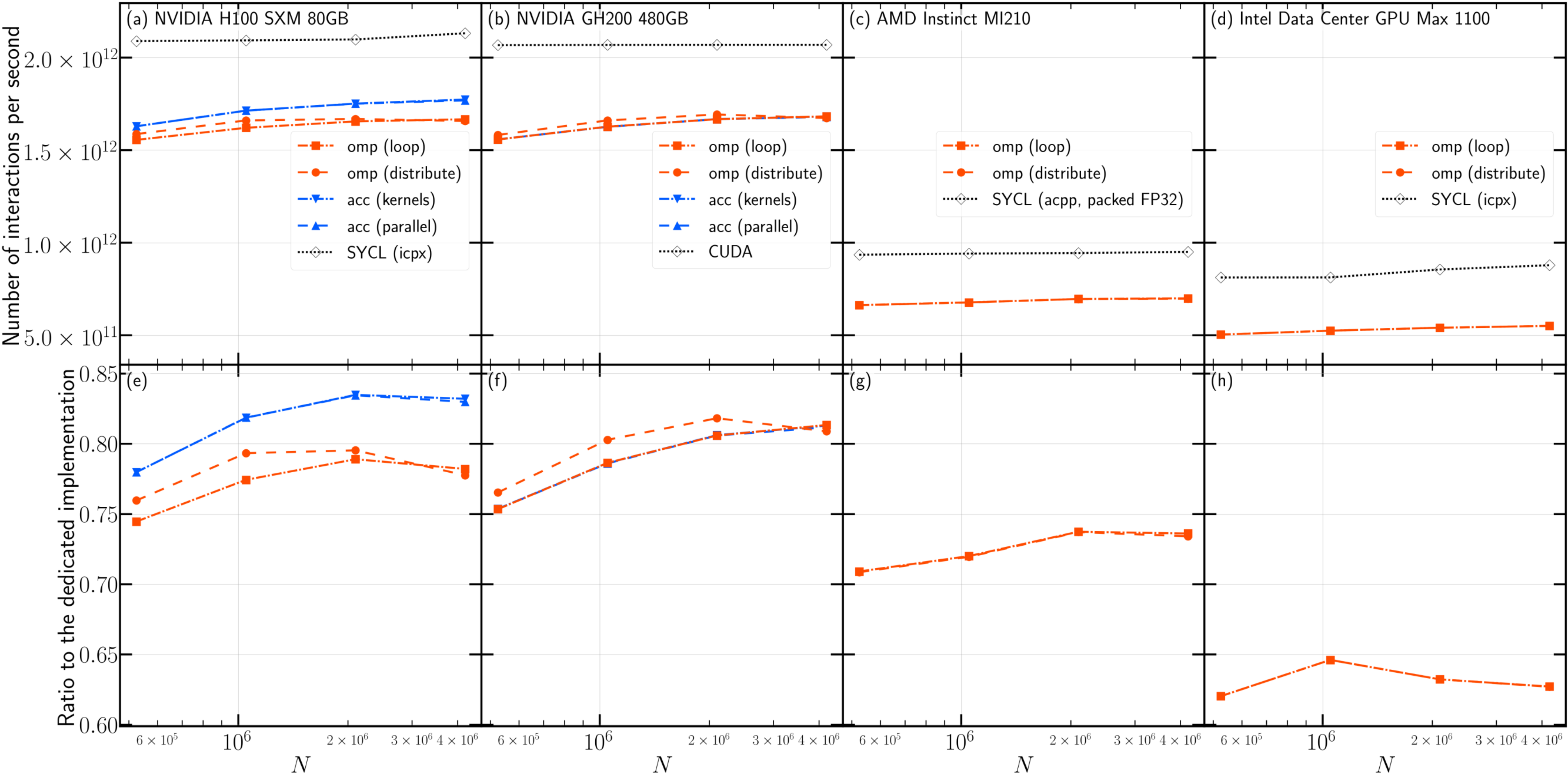
- Outline of gravity solver

```
void calc_acc(…) {
  PRAGMA_OMP_TARGET_TEAMS_LOOP(OMP_TARGET_CLAUSE_SIMD, OMP_TARGET_CLAUSE_THREAD_LIMIT(NTHREADS))
  for (std::remove_const_t<decltype(Ni)> i = 0; i < Ni; i++) {
    // initialization
    PRAGMA_ACC_LOOP(ACC_CLAUSE_SEQ)
    for (std::remove_const_t<decltype(Nj)> j = 0; j < Nj; j++) {
      // main computation
    }
    iacc[i] = ai;
  }
}
```

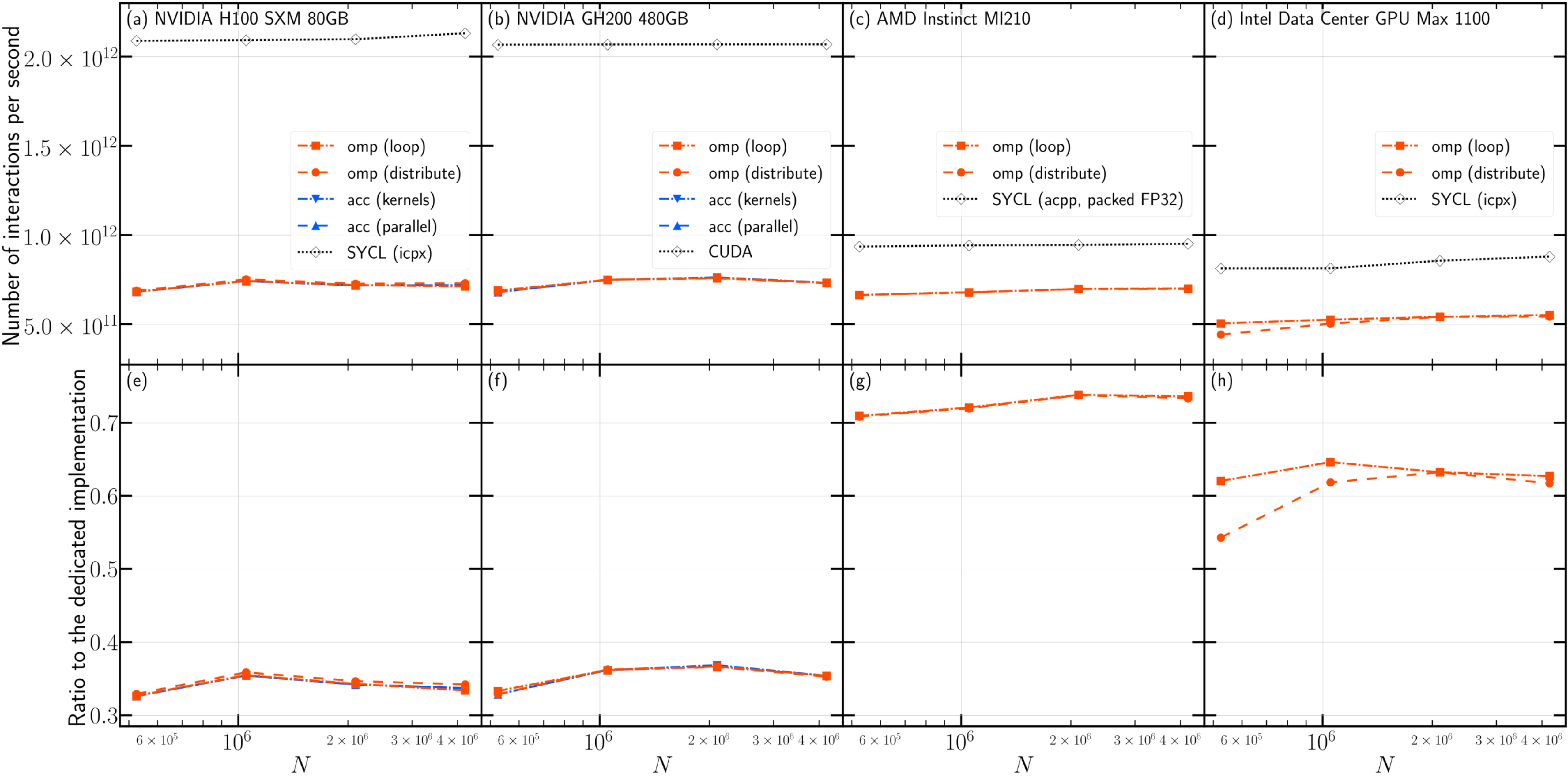
HW/SW environments

GPU	NVIDIA H100 SXM 80GB	NVIDIA GH200 480GB	AMD Instinct MI210	Intel Data Center GPU Max 1100
FP32 peak	66.9 TFlop/s	66.9 TFlop/s	22.6 TFlop/s	22.2 TFlop/s
# of units	132 SMs	132 SMs	104 CUs	448 EUs
FP32 parallelism	16896	16896	6656	7168
Clock frequency	1980 MHz	1980 MHz	1700 MHz	1550 MHz
Memory	HBM3 80GB	HBM3 96GB	HBM2e 64GB	HBM2e 48GB
Bandwidth	3.36 TB/s	4.02 TB/s	1.64 TB/s	1.23 TB/s
TDP	700 W	1000 W (total)	300 W	300 W
Host CPU	Intel Xeon Platinum 8468	NVIDIA Grace	AMD EPYC 7713	Intel Xeon Platinum 8468
	48 cores × 2 sockets	72 cores	64 cores × 2 sockets	48 cores × 2 sockets
	2.1 GHz	3.1 GHz	2.0 GHz	2.1 GHz
	CUDA 12.3	CUDA 12.4	ROCm 6.0.2	
Intel oneAPI	2024.1.0		2024.1.0	2024.1.0
NVIDIA HPC SDK	24.3-0	24.5-1		
AdaptiveCpp			24.02.0	

Measured N-body performance



If the compiler option is suboptimal (e.g., `-Ofast -gpu=cc90`)



Example: 3D diffusion equation

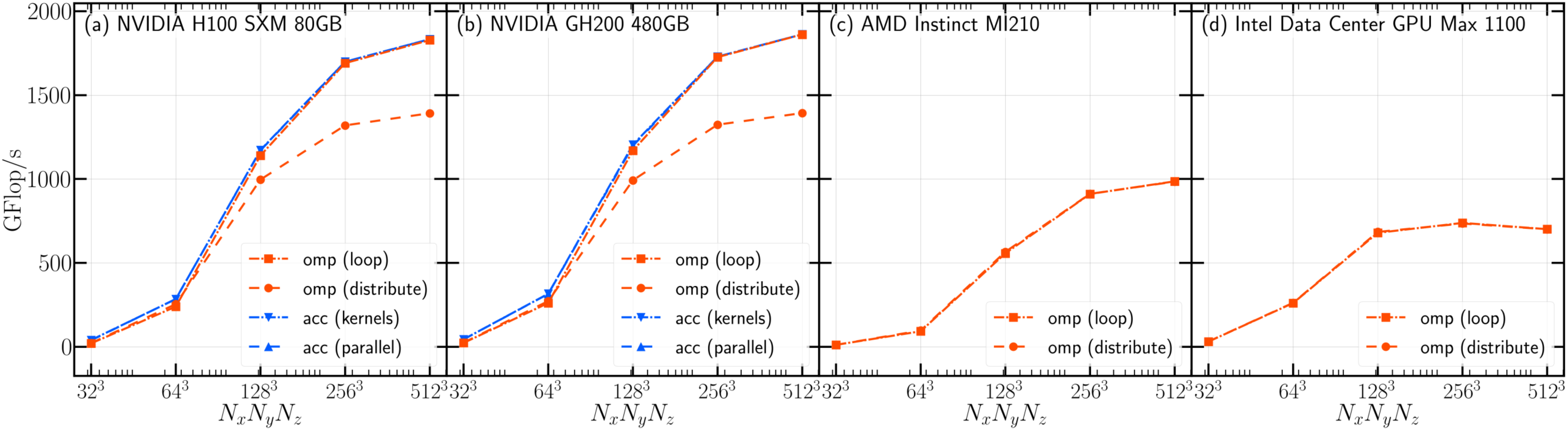
- “Solomonized” OpenACC code
  - Original OpenACC code was provided by Tetsuya Hoshino (Nagoya Univ.)

```
init(nx, ny, nz, dx, dy, dz, f);
PRAGMA_ACC_DATA(ACC_CLAUSE_COPY(f [0:n]), ACC_CLAUSE_CREATE(fn [0:n])) {
  for (; icnt < nt && time + 0.5 * dt < 0.1; icnt++) {
    flop += diffusion3d(nx, ny, nz, dx, dy, dz, dt, kappa, f, fn);
    swap(&f, &fn);
    time += dt;
  }
}
```

```
OFFLOAD(AS_INDEPENDENT, COLLAPSE(3), ACC_CLAUSE_PRESENT(f, fn))
for (int i = 0; i < nx; i++) {
  for (int j = 0; j < ny; j++) {
    for (int k = 0; k < nz; k++) {
      const int ix = INDEX(nx, ny, nz, i, j, k);
      const int ip = INDEX(nx, ny, nz, IMIN(i + 1, nx - 1), j, k);
      const int im = INDEX(nx, ny, nz, IMAX(i - 1, 0), j, k);
      const int jp = INDEX(nx, ny, nz, i, IMIN(j + 1, ny - 1), k);
      const int jm = INDEX(nx, ny, nz, i, IMAX(j - 1, 0), k);
      const int kp = INDEX(nx, ny, nz, i, j, IMIN(k + 1, nz - 1));
      const int km = INDEX(nx, ny, nz, i, j, IMAX(k - 1, 0));
      fn[ix] = cc * f[ix] + ce * f[ip] + cw * f[im] + cn * f[jp] + cs * f[jm] + ct * f[kp] + cb * f[km];
    }}}
```

Performance: 3D diffusion equation

- Low performance of OpenMP (distribute) on NVIDIA GPUs
- No difference in loop and distribute on AMD/Intel GPUs
- Memory-intensive application (B/F=2.5): limiter is cache
  - NVIDIA: 4.51 TB/s (H100), 4.58 TB/s (GH200)
  - AMD MI210: 2.43 TB/s
  - Intel Data Center GPU Max 1100: 1.82 TB/s



Summary

- Directive-based GPU offloading has two options:
  - OpenACC: virtually for NVIDIA GPU, better function/docs
  - OpenMP target: for NVIDIA/AMD/Intel GPUs, fewer function/docs
- We have developed the head-only library “Solomon”
  - Miki & Hanawa (2024, *IEEE Access*, 12, 181644)
  - Simple Off-LOading Macros Orchestrating multiple Notations
  - Aggregate of preprocessor macros for directives and clauses
  - Users can exploit OpenACC on NVIDIA GPU and OpenMP target on NVIDIA/AMD/Intel GPUs
  - Three types of notations to reduce learning costs: intuitive notation and OpenACC/OpenMP-like notation
  - Users can use compilers by GPU-manufacturing vendors as is
  - Easy performance comparison of OpenACC and OpenMP target
- Available at https://github.com/ymiki-repo/solomon
- Kokkos and Solomon are complementary
  - Kokkos will provide higher performance
  - Solomon is more simple and easier (especially for beginners)