PCクラスタワークショップ

AMD Instinct™ MI300Aが実現するAPUプログラミング・モデルについて

日本AMD株式会社
コマーシャル営業本部
シニア・ソリューション・アーキテクト

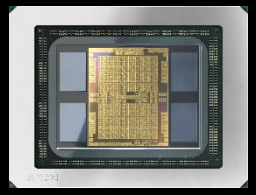
大原　久樹

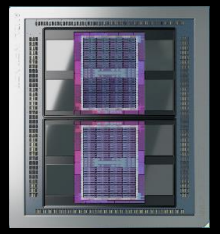
AMD
together we advance_

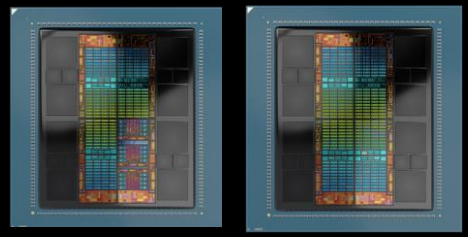
AMD Instinct™ アクセラレーターの軌跡

Multiple generations of architecture focused advancing HPC & AI compute

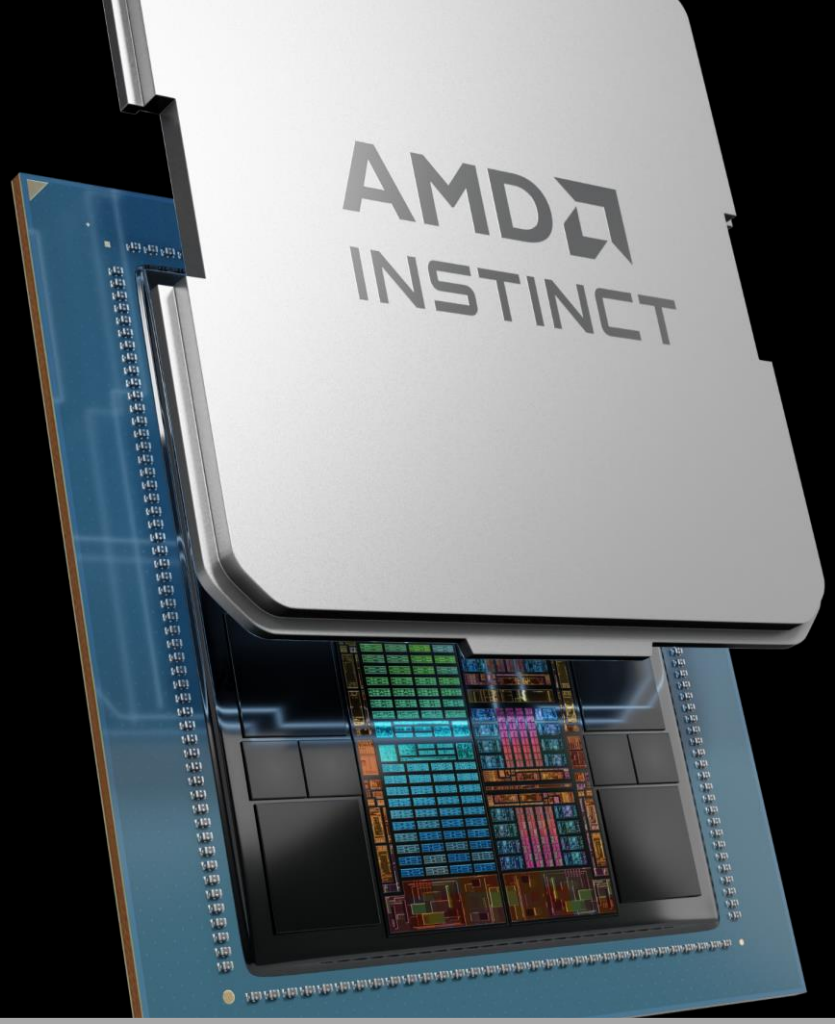
MI100	MI200	MI300
AMD CDNA™	AMD CDNA™ 2	AMD CDNA™ 3
ECOSYSTEM GROWTH	DRIVING HPC AND AI TO A NEW FRONTIER	DATA CENTER APU & DISCRETE GPU
FP64とFP32のHPCワークロードを加速するために設計された初のGPUアーキテクチャ	メモリ容量・帯域に優れた高密度計算アーキテクチャ	ユニファイド・メモリ、AI性能、ノード内ネットワークを大幅に改善したアーキテクチャ







2020 — 2023



AMD Instinct™ MI300A

世界初のAI・HPC向けデータセンターAPUアクセラレーター

AMD CDNA 3

ZEN 4

128 GB
HBM3

5nm and **6**nm
Process Technology

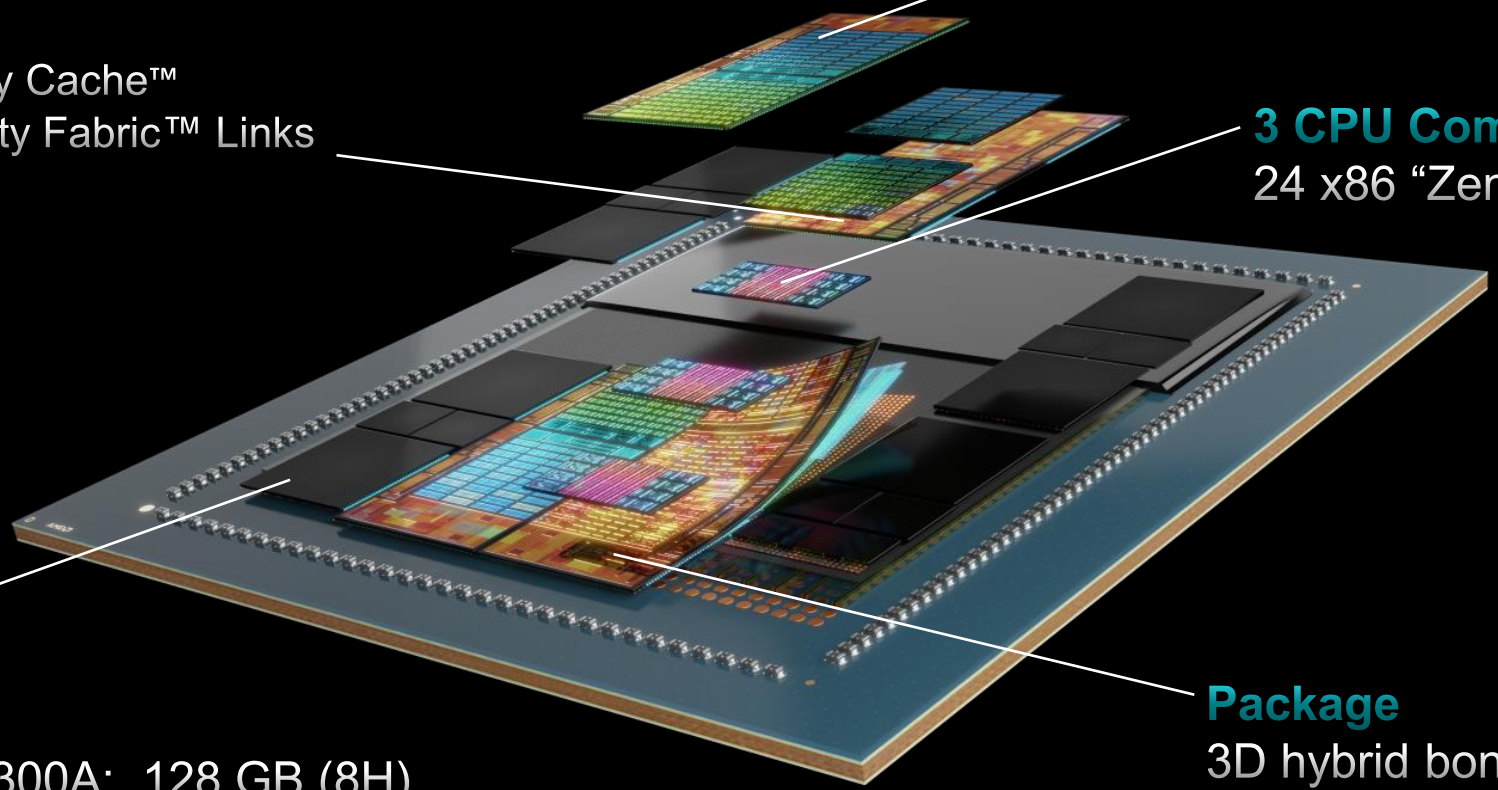
Shared Memory
CPU + GPU

AMD Instinct™ MI300A

Accelerator Complex Die (XCD)
228 AMD CDNA™ 3 Compute Units

I/O Die (IOD)
256MB AMD Infinity Cache™
4 x16 4th Gen Infinity Fabric™ Links
4 x16 PCIe® 5

3 CPU Complex Die (CCD)
24 x86 “Zen 4” Cores



HBM3
8 physical stacks
AMD Instinct™ MI300A: 128 GB (8H)
~5.3 TB/s Bandwidth

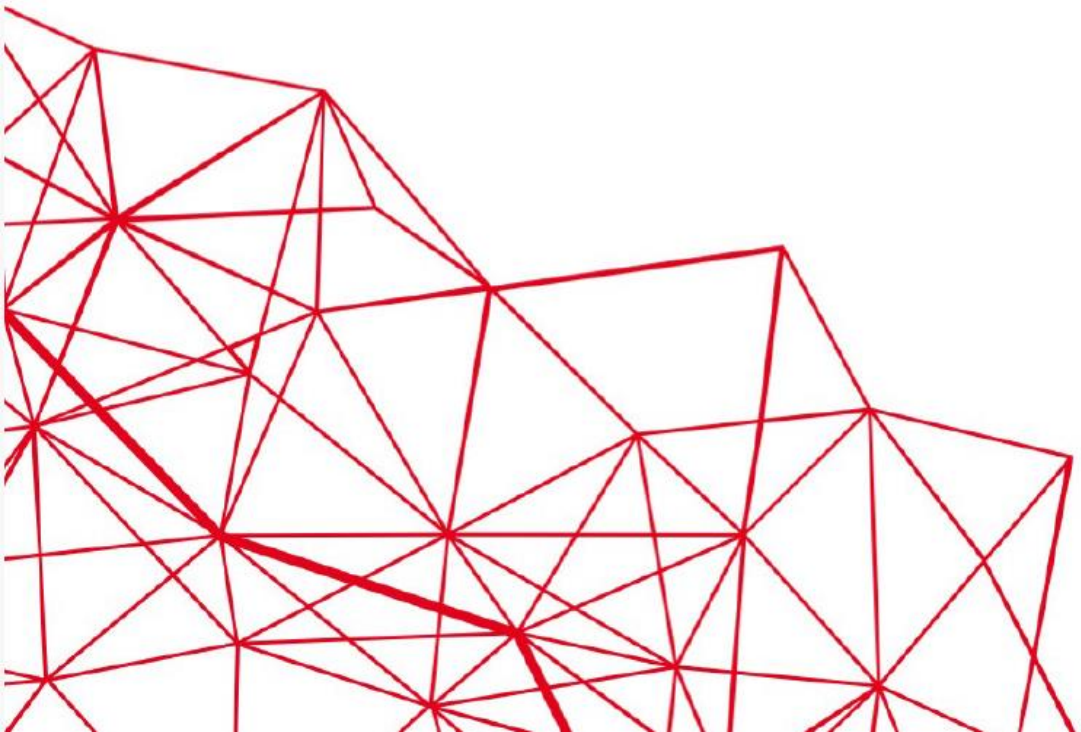
Package
3D hybrid bonded
2.5D silicon interposer

See endnotes: MI300-13, MI300-06

APUの優位性

Unlocking new performance capabilities

ユニファイド・メモリ	共有キャッシュ AMD Infinity Cache™	CPUとGPUの 電力共有	プログラミング の容易性



ISC High Performance

REINVENTING HPC

MAY 12 – 16, 2024 | HAMBURG, GERMANY

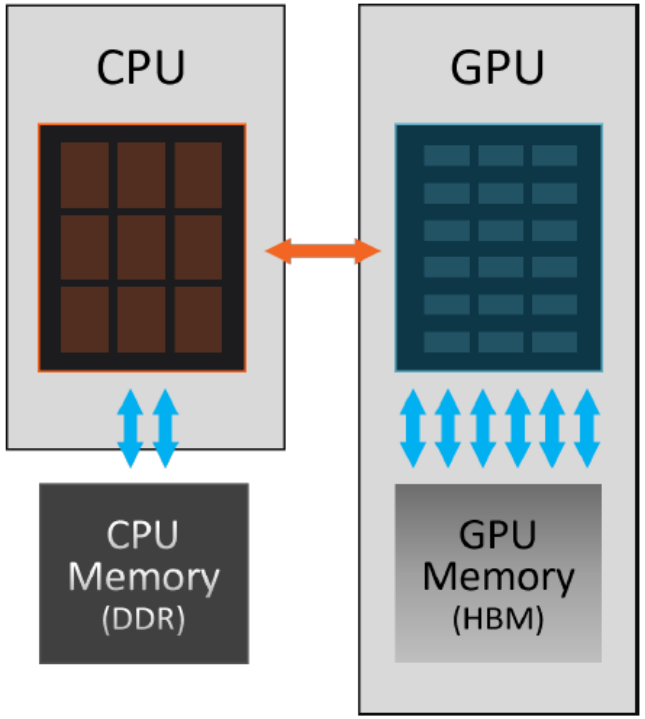
Porting HPC Applications to AMD Instinct™ MI300A Using Unified Memory and OpenMP®

Suyash Tandon, Leopold Grinberg, Gheorghe-Teodor Bercea, Carlo Bertolli, Mark Olesen, Simone Bna, and Nicholas Malaya

MI300A – The APU

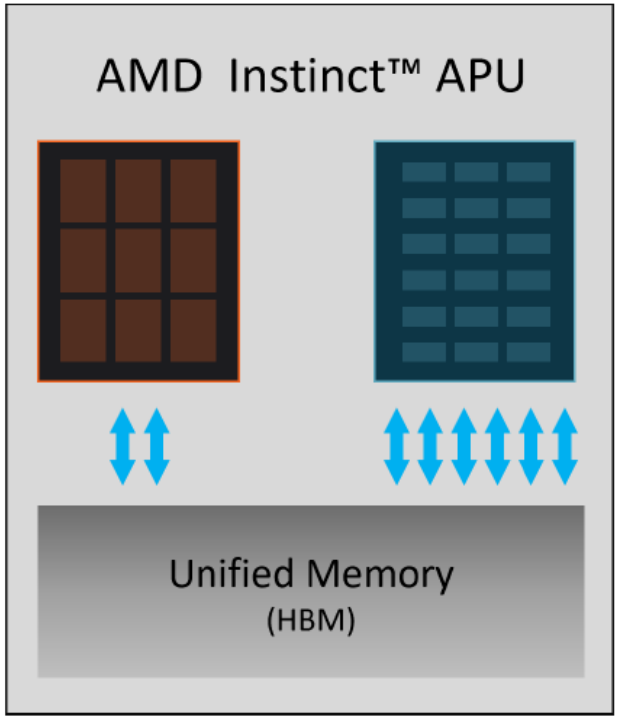
AMD CDNA™ 2 Coherent Memory Architecture

AMD CDNA™ 3 Unified Memory APU Architecture



Schematic representation of a socket with discrete CPU and GPU

- Eliminate Redundant Memory Copies
- Does not need a programming distinction between host and device memory spaces
- High performance, fine-grained sharing between CPU and GPU processing elements



Schematic representation of a socket with APU

Porting applications with HIP or OpenMP®

CPU CODE

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);



for (int i=0; i<M; i++) // initialize
   in_h[i] = …;

cpu_func(in_h, out_h, M);



for (int i=0; i<M; i++) // CPU-process
  … = out_h[i];
```

GPU CODE - HIP

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);
hipMalloc(&in_d, Msize);
hipMalloc(&out_d, Msize);



for (int i=0; i<M; i++) // initialize
   in_h[i] = …;
hipMemcpy(in_d,in_h,Msize);
gpu_func<< >>(in_d, out_d, M);
hipDeviceSynchronize();
hipMemcpy(out_h,out_d,Msize);

for (int i=0; i<M; i++) // CPU-process
  … = out_h[i];
```

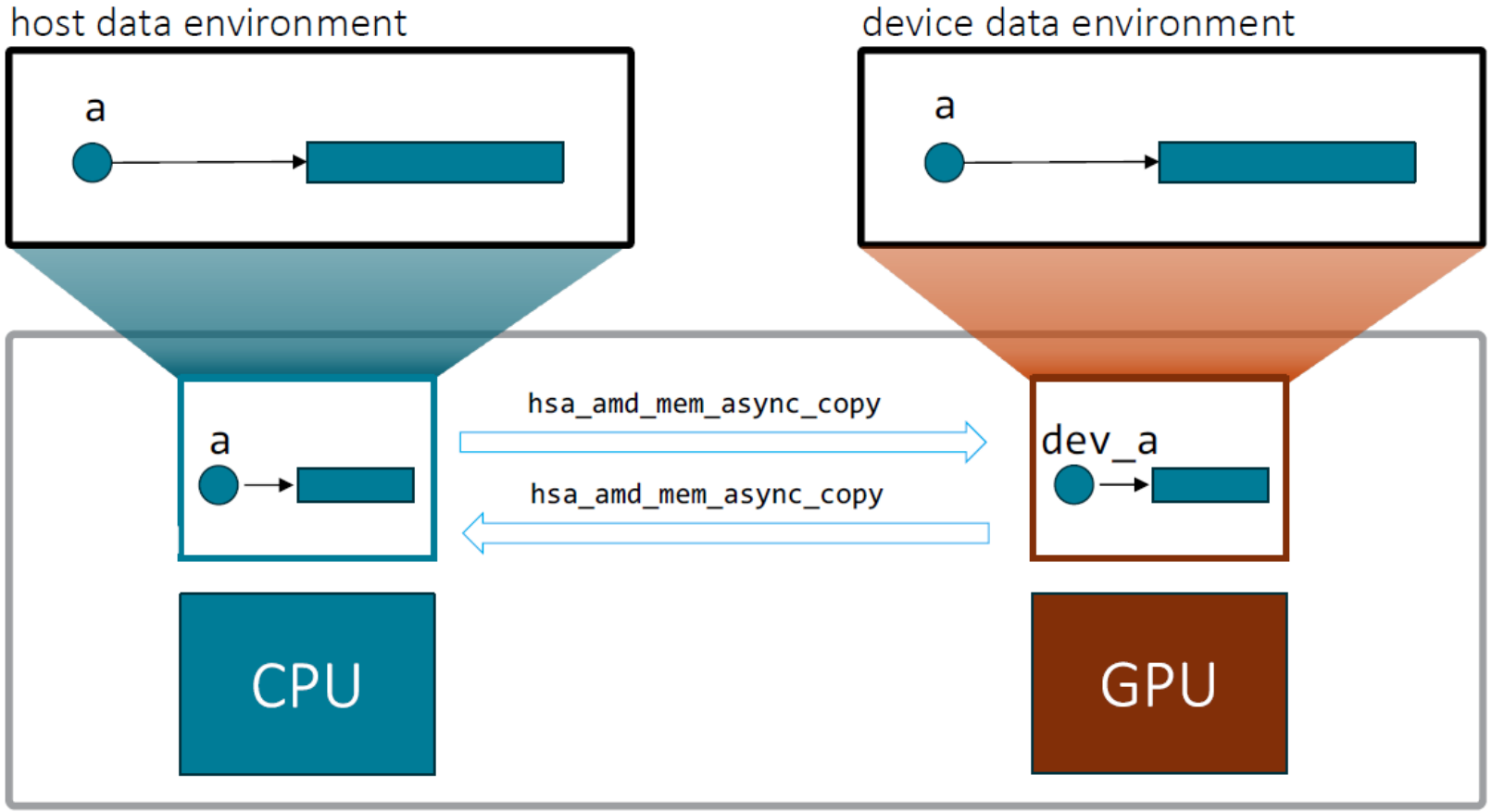
GPU CODE - OPENMP

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);



for (int i=0; i<M; i++) // initialize
   in_h[i] = …;

#pragma omp target teams
map(to:in_h[0:M]) map(tofrom:out_h[0:M])
{ … }

for (int i=0; i<M; i++) // CPU-process
  … = out_h[i];
```

- GPU memory allocation on Device
- Explicit memory management between CPU & GPU
- Synchronization Barrier. Implicit in case of OpenMP

OpenMP Data Environments: dGPU Implementation


```
int *a = new int[N];
#pragma omp target teams loop map(tofrom:a[0:N])
```



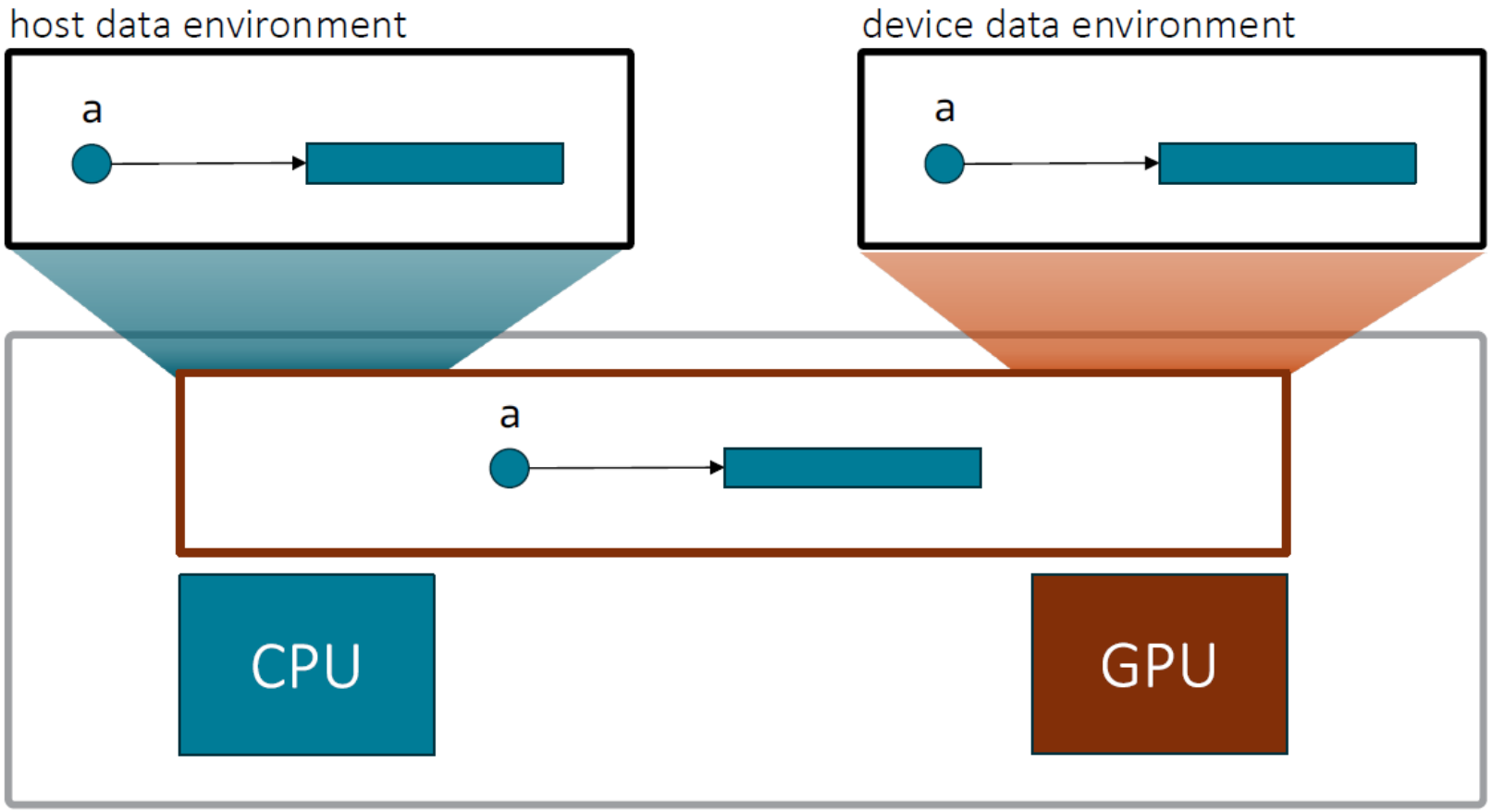
Map Clause

Specifies how an original list item is mapped from the host data environment to a corresponding list item in the device data environment

`map-type` – describe the type of mapping
`from or tofrom:` the value of the list item is copied from the device environment to original list

OpenMP Data Environments: APU Implementation


```
int *a = new int[N];
#pragma omp target teams loop map(tofrom:a[0:N])
```



Zero Copies

IMPLICIT OR AUTOMATIC

The runtime detects the node is an MI300A and turns on zero-copy if XNACK*-Enabled

UNIFIED SHARED MEMORY (USM)

Compiler and runtime know this application will require unified memory support on any target GPU. Map memory by users is not needed.

*Technical term used by ROCm developers XNACK (address-not-translated), to indicate that memory address could not be translated due to missing page table entry in cache

The APU programming model

GPU CODE - OPENMP

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);



for (int i=0; i<M; i++) // initialize
   in_h[i] = …;

#pragma omp target teams
map(to:in_h[0:M]) map(tofrom:out_h[0:M])
{ … }

for (int i=0; i<M; i++) // CPU-process
  … = out_h[i];
```

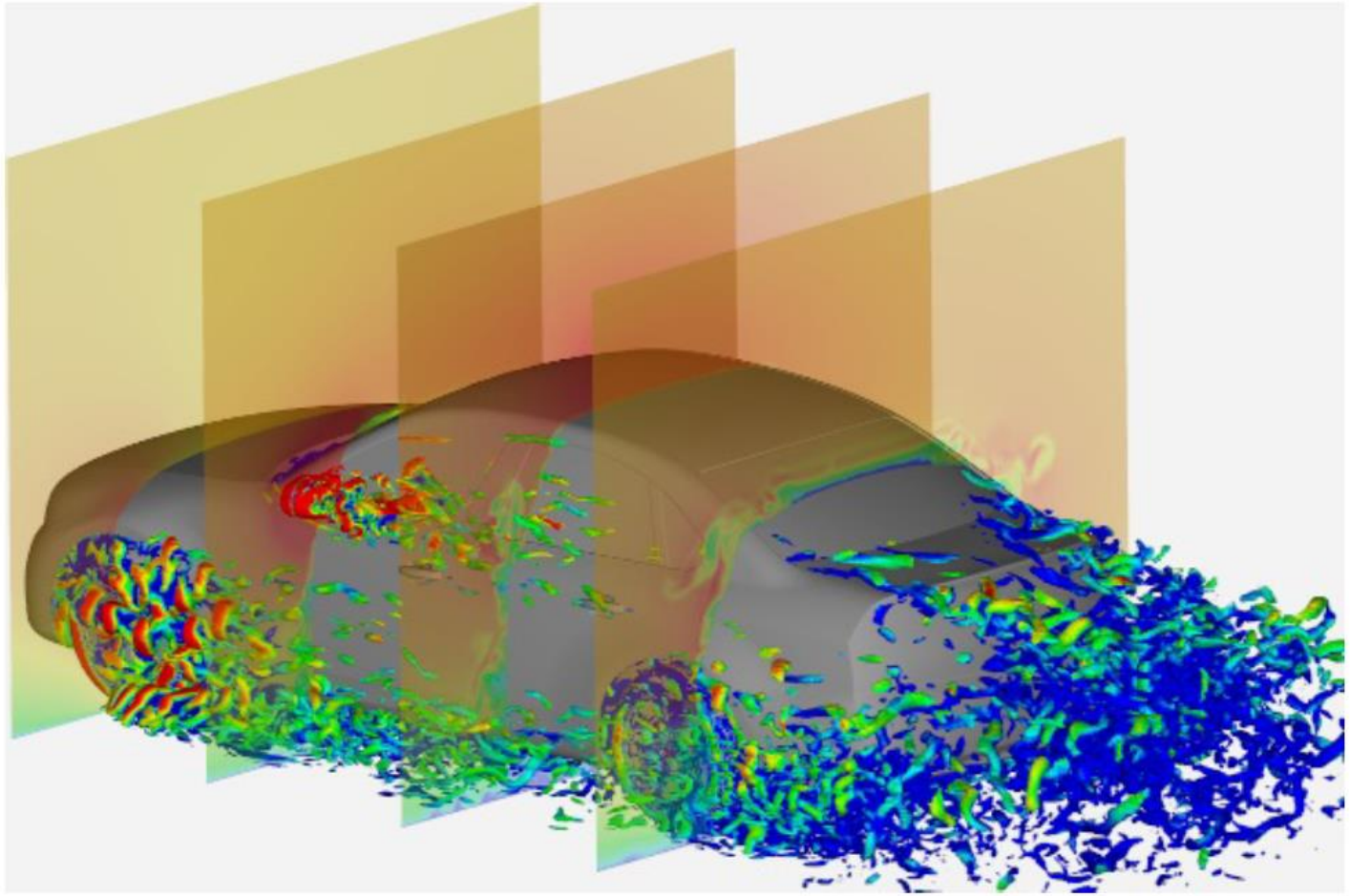
OpenMP runtime knows it can omit copies and map clauses when unified_shared_memory is used.

APU CODE - OPENMP

```
#pragma omp requires unified_shared_memory
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);



for (int i=0; i<M; i++) // initialize
   in_h[i] = …;

#pragma omp target
{ … }

for (int i=0; i<M; i++) // CPU-process
  … = out_h[i];
```

- ~~GPU memory allocation on Device~~
- ~~Explicit memory management between CPU & GPU~~
- Synchronization Barrier. Implicit in case of OpenMP

OpenFOAM®



www.openfoam.com

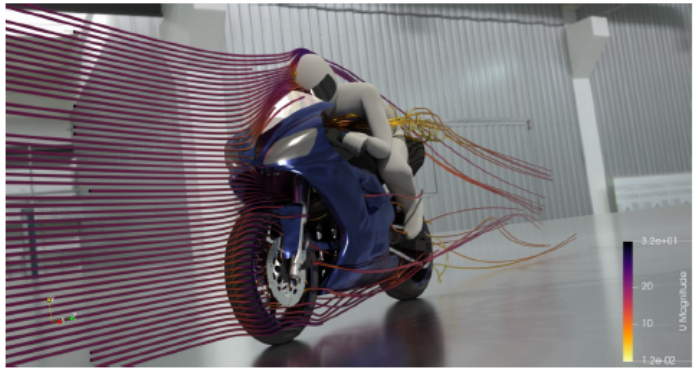
OpenFOAM® is an open source C++ library for computational fluid dynamics (CFD) and primarily developed by OpenCFD, Ltd. since 2004.

- $O(10^6)$ lines of C++ code
- **Solvers,** large number of smoothers, preconditioners & linear algebra functionality
- **CPU-Optimized** code with efficient scalability on large distributed platforms using MPI
- **Interface** with external third-party libraries, such as PETSc, AMGx, Scotch, Adios, etc.

Benchmark – HPC Motorbike Large (34.4 Million cells)

Default setup:

Parameters	Configuration
Start Time [s]	0
End Time [s]	20
Delta T [s]	1
#Iterations (time steps)	20
Momentum Solve	PBiCGStab with DILU preconditioner
Pressure Solve	PCG with DIC preconditioner
Turbulence	PBiCGStab with DILU preconditioner
Application	**SimpleFoam**
Figure of Merit (FoM)	Total time or time per time-step (all in sec.)
FoM – Lower is Better?	Yes

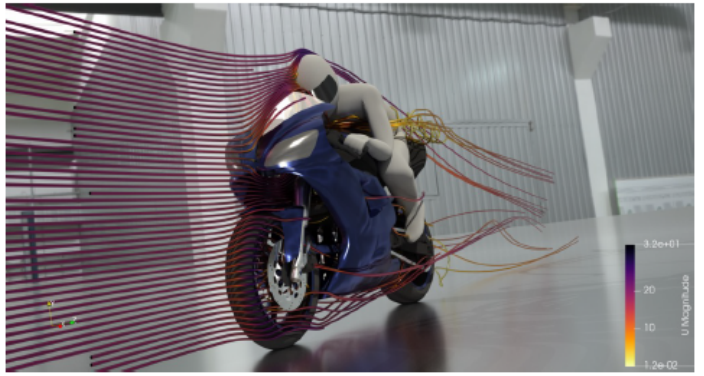


HPC Motorbike, rendered in ParaView (www.kitware.com).

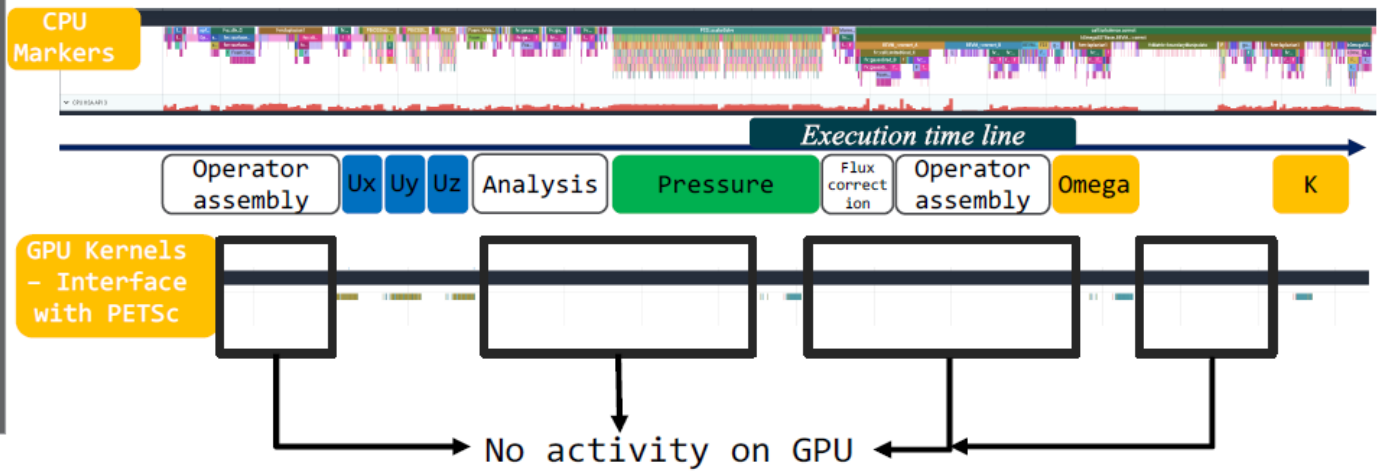
SimpleFoam – inspecting trace of a time step


```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Info<< "\nStarting time loop\n" << endl;

while (simple.loop())
{
Info<< "Time = " << runTime.timeName() << nl << endl;

// --- Pressure-velocity SIMPLE corrector
{
#include "UEqn.H"
#include "pEqn.H"
}

laminarTransport.correct();
turbulence->correct();

runTime.write();

runTime.printExecutionTime(Info);
}

Info<< "End\n" << endl;

return 0;
}
```

1. Solve momentum – for `Ux, Uy, Uz`
2. Solve pressure equation
3. Evaluate turbulence



HPC Motorbike, rendered in ParaView (www.kitware.com).



PETSc is accelerating only the solves on the GPU.

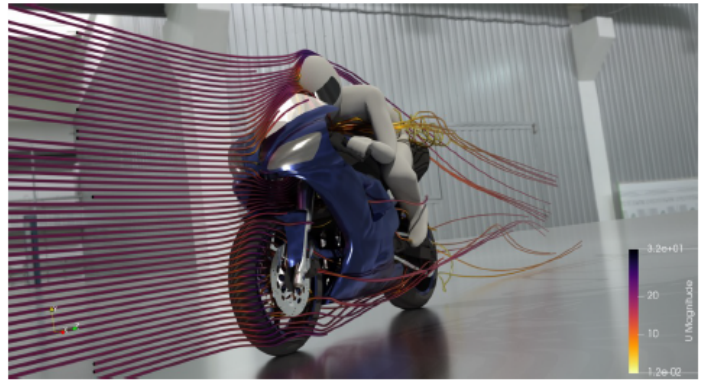
Major chunk of the workflow still resides on the CPU!

SimpleFoam – offloading parallelizable loops


```
#include "PBiCGStab.H"
#include "PrecisionAdaptor.H"

#ifdef USE_OMP
#include <omp.h>
#ifndef OMP_UNIFIED_MEMORY_REQUIRED
#define OMP_UNIFIED_MEMORY_REQUIRED
#pragma omp requires unified_shared_memory
#endif
#endif
...
// --- Solver iteration
do
{
// --- Store previous rA0rA
const solveScalar rA0rAold = rA0rA;

rA0rA = gSumProd(rA0, rA, matrix().mesh().comm());
...
// --- Update pA
if (solverPerf.nIterations() == 0)
{
#ifdef USE_OMP
#pragma omp target teams distribute parallel for if (target:nCells>20000)
#endif
for (label cell=0; cell<nCells; cell++)
{
pAPtr[cell] = rAPtr[cell];
}
...
// --- Precondition pA
preconPtr_->precondition(yA, pA, cmpt);

// --- Calculate AyA
matrix_.Amul(AyA, yA, interfaceBouCoeffs_, interfaces_, cmpt);
...
```

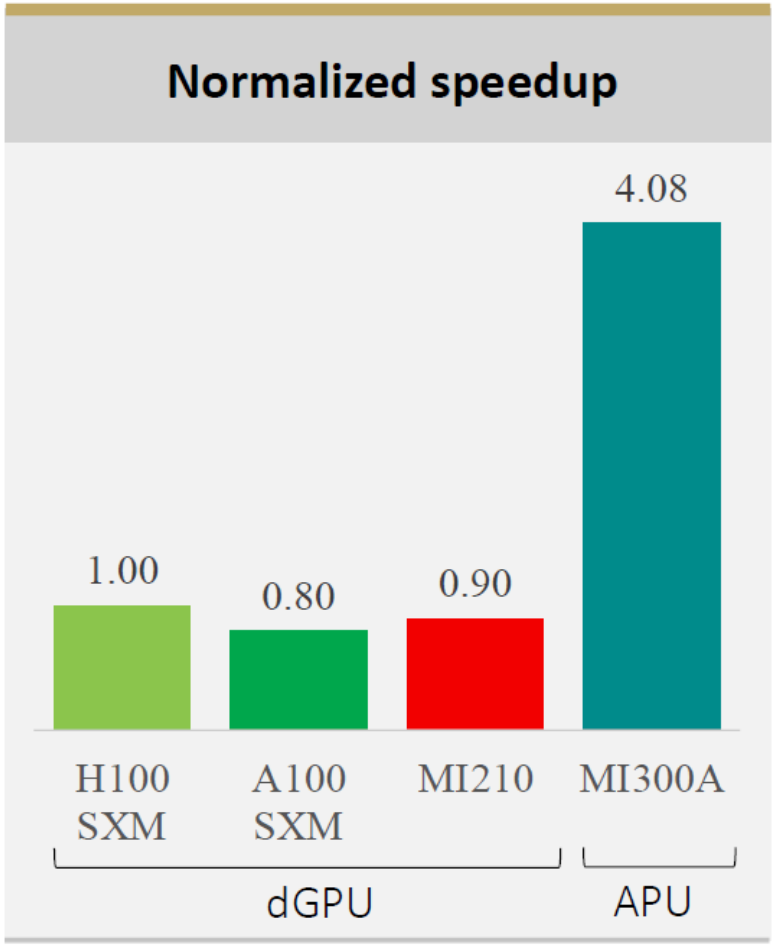
Matrix assembly

```
...
const label nCells = diag().size();
#if 1 //mi300A

const label nFaces = upper().size();


#pragma omp target teams distribute parallel for if(target:nCells>TARGET_CUT_OFF)
for (label cell=0; cell<nCells; cell++)
{
ApsiPtr[cell] = diagPtr[cell]*psiPtr[cell];
}

#pragma omp target teams distribute parallel for thread_limit(64) if(target:nCells>TARGET_CUT_OFF)
for (label face=0; face<nFaces; face++)
{
#pragma omp atomic
ApsiPtr[uPtr[face]] += lowerPtr[face]*psiPtr[lPtr[face]];
#pragma omp atomic
ApsiPtr[lPtr[face]] += upperPtr[face]*psiPtr[uPtr[face]];
}
...
```



re.com).

Performance Evaluation – The APU advantage

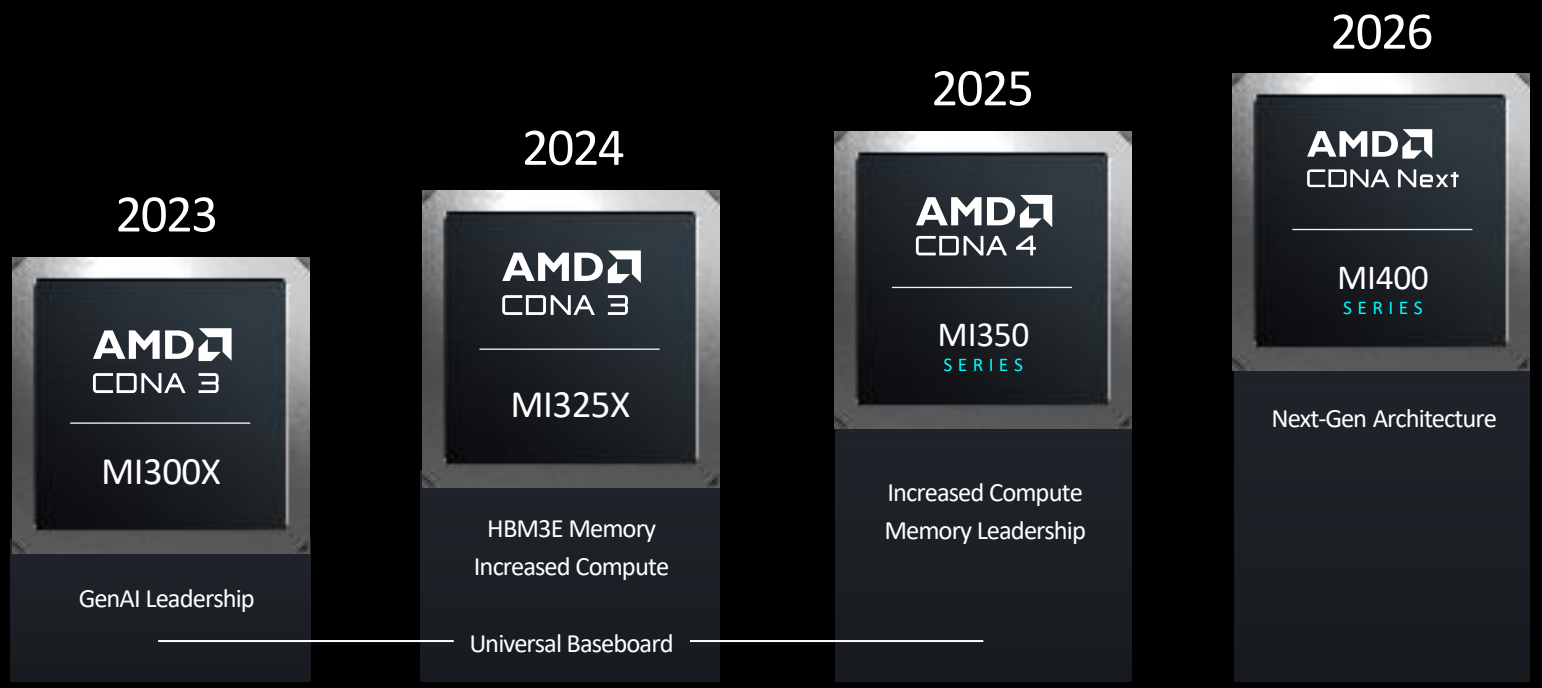


Device	Stack, compiler, NPS	Memory pool
MI300A	ROCm-6.0, clang++ (clang-17.0), NPS1	System memory
x86, MI210, PCIe4.0	ROCm-6.0, clang++ (clang-17.0), NPS1	Managed memory
x86, A100-80GB SXM, PCIe 4.0	CUDA-12.2.2, clang-18.0, NPS 1	Managed memory
X86, H100 SXM, PCIe 5.0	CUDA-12.2.2, clang-18.0, NPS 1	Managed memory

AMD INSTINCT

Leadership roadmap on annual cadence

All roadmaps are subject to change.



Announced last week

Ultra Accelerator Link

Partner group of innovators for scale up AI infrastructure

AMD BROADCOM cisco Google Hewlett Packard Enterprise intel Meta Microsoft

High Performance | Open | Scalable

Ultra Ethernet is the answer for scale out AI infrastructure


AMD