Programming the AMD Instinct™ MI300 APU

Or everything you wanted to know about APU programming but were afraid to ask

Presented at ESPM2 at SC23

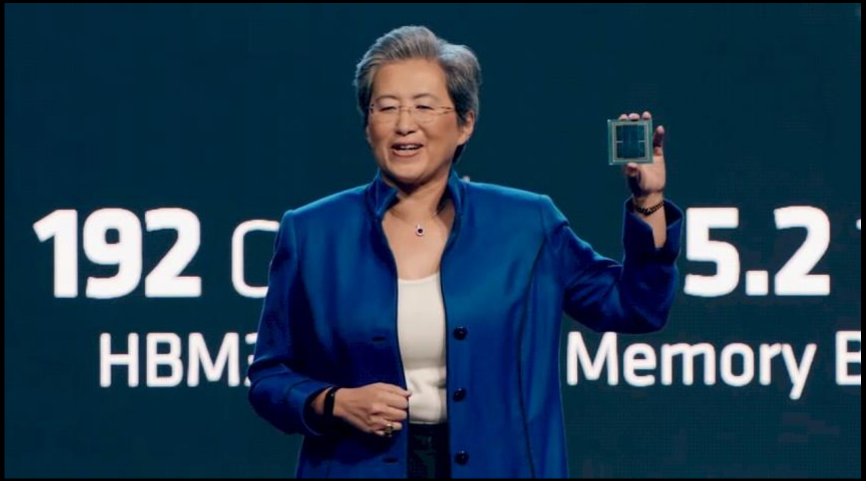
AMD

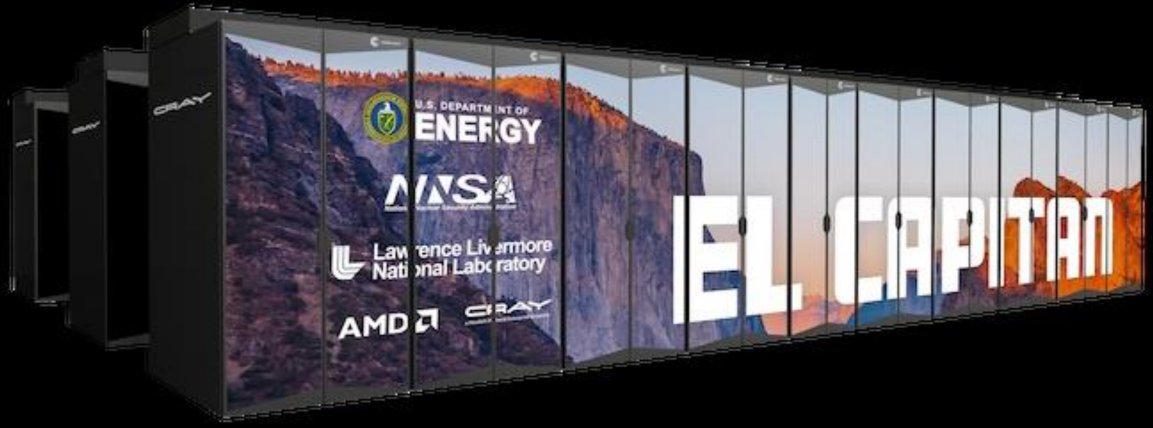
together we advance_

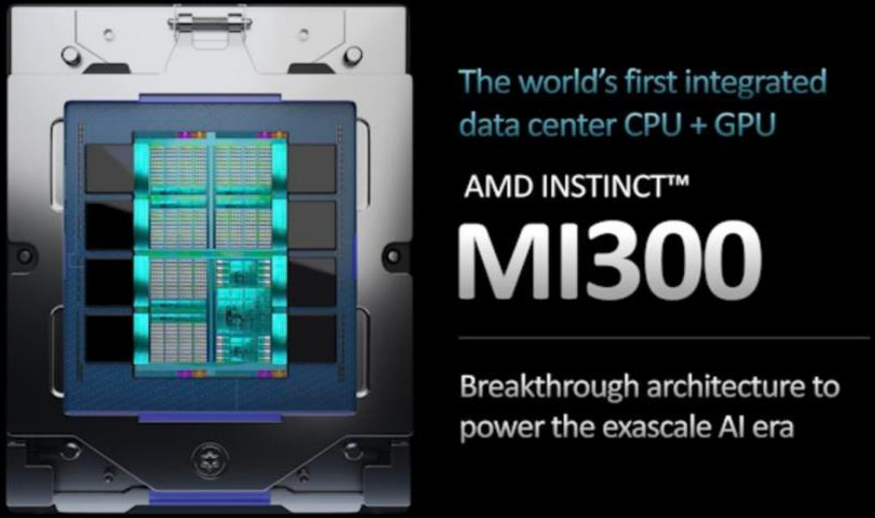
AMD INSTINCT™ MI300 APU

- LLNL’s El Capitan Exascale will be powered by the AMD Instinct™ MI300 APU: “MI300A”
- MI300A is an APU, with AMD CDNA™ 3 GPUs, Zen 4 CPUs, cache memory, and HBM chiplets in a single package
- 24 Zen4 CPU cores
- 128 GiB of HBM3

“It’s much easier to program”




U.S. DEPARTMENT OF ENERGY
NNSA
Lawrence Livermore National Laboratory
AMD
CRAY
EL CAPITAN


The world’s first integrated data center CPU + GPU
AMD INSTINCT™
MI300
Breakthrough architecture to power the exascale AI era

AGENDA

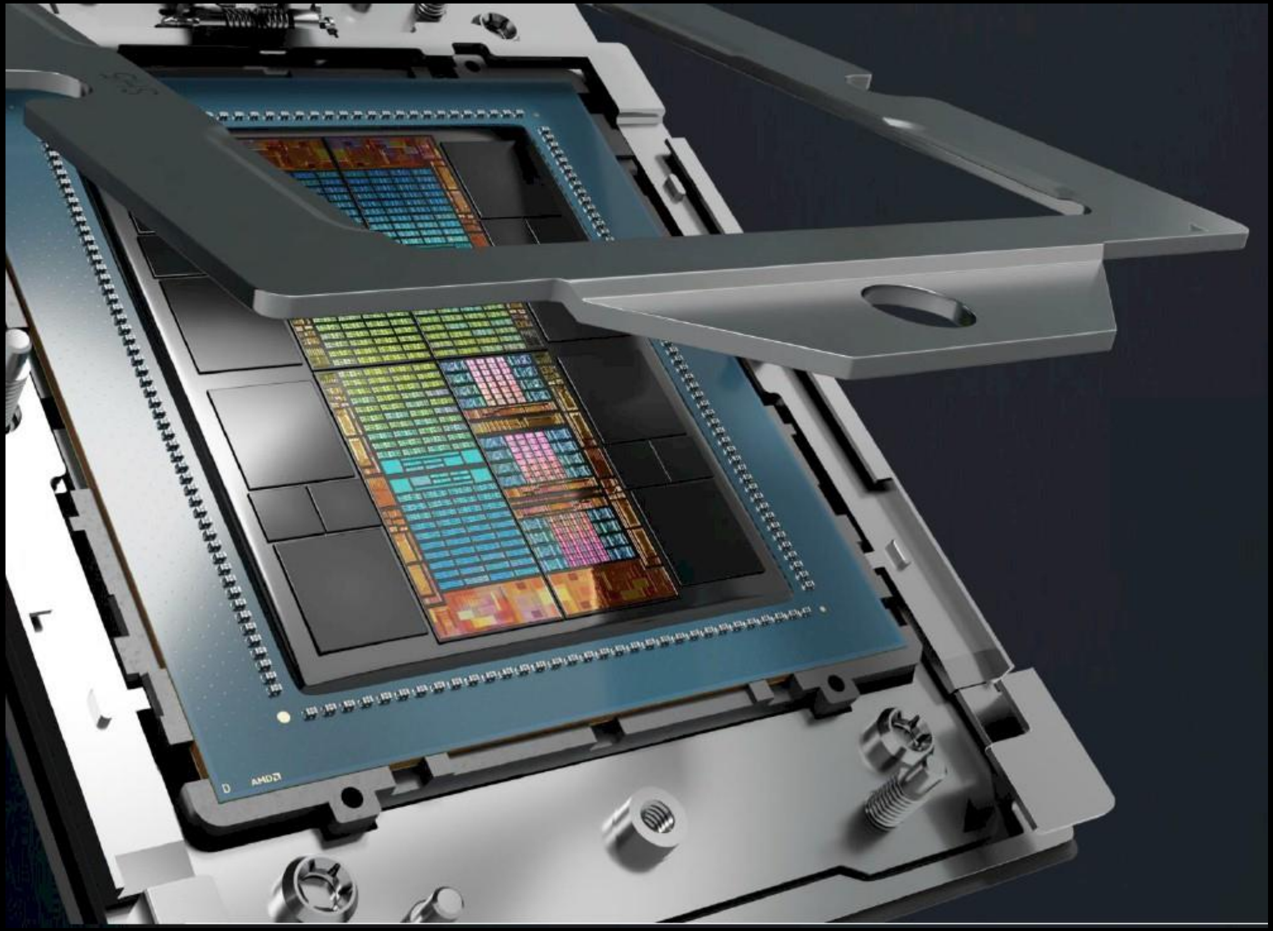
MI300A Hardware: What is an APU?

How can programmers prepare for the APU architecture?

- Simple examples in HIP, RAJA, Kokkos, OpenMP

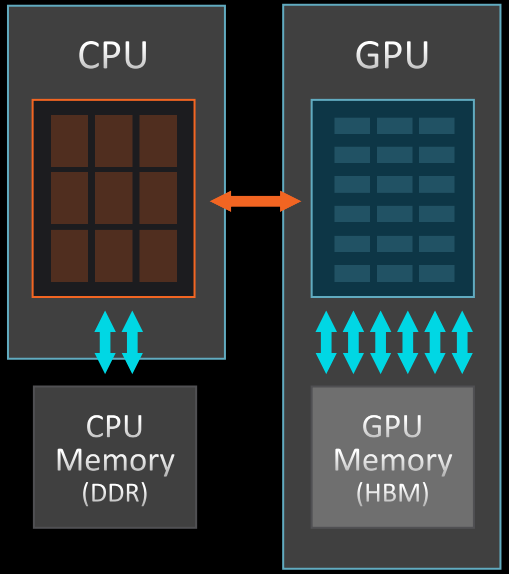
What are the performance opportunities of an APU?

- Removing redundant memory copies
- Fine-grained interleaving between CPU and GPU
- Scheduling concurrent work
- Efficient memory allocations and kernel launches

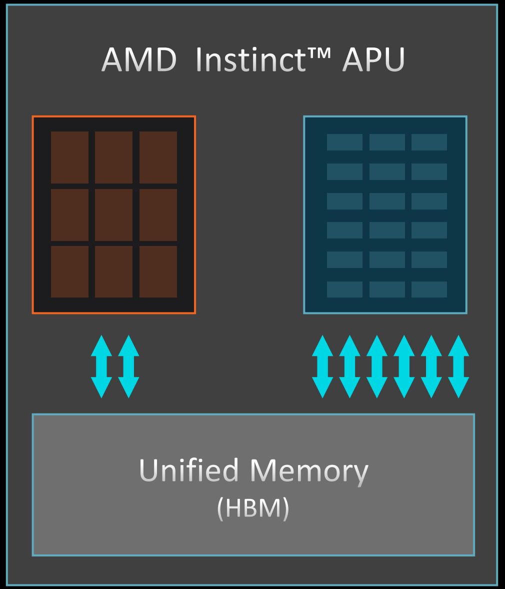


AMD MI300A UNIFIED MEMORY APU ARCHITECTURE BENEFITS

AMD CDNA™ 2 Coherent Memory Architecture → AMD CDNA™ 3 Unified Memory APU Architecture



- Eliminate Redundant Memory Copies
- No programming distinction between host and device memory spaces
- High performance, fine-grained sharing between CPU and GPU processing elements
- Single process can address all memory, compute elements on a socket



APU PROGRAMMING MODEL

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

```
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];
```

APU CODE

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


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

gpu_func<< >>(in_h, out_h, M);
hipDeviceSynchronize();


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

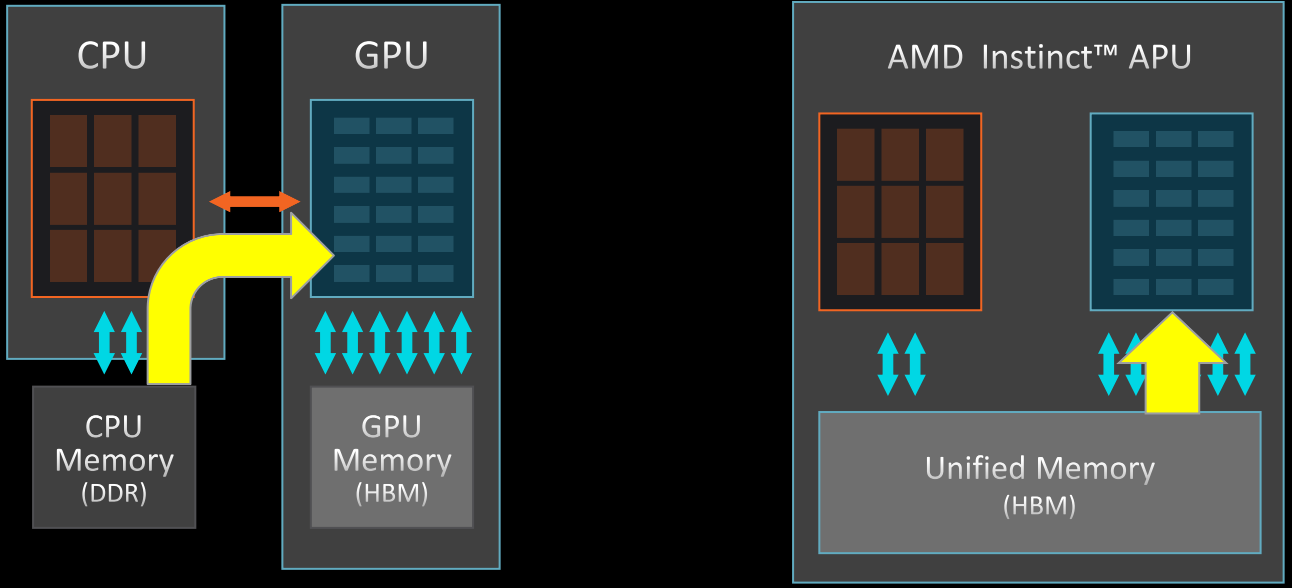
APU PROGRAMMING: PERFORMANCE IMPLICATIONS

GPU CODE

```
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];
```



Operation	MI250X (MCM)	MI300A
H2D Copy	O(10) GB/s	O(TB/s)

- GPU memory allocation on Device
- Explicit memory management between CPU & GPU
- Synchronization Barrier

AMD together we advance_

[Public]

APU PROGRAMMING: PERFORMANCE IMPLICATIONS

APU CODE

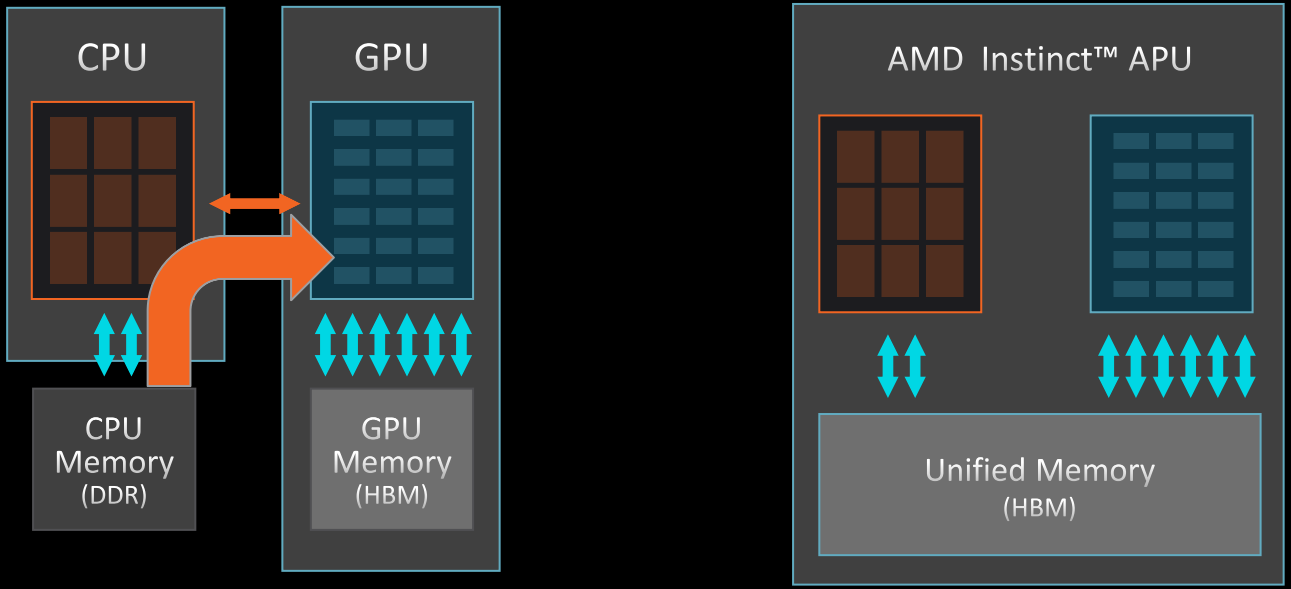
```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);



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

gpu_func<< >>(in_h, out_h, M);
hipDeviceSynchronize();


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



Operation	MI250X (MCM)	MI300A
Coherent Access	O(10) GB/s	N/A

- ~~GPU memory allocation on Device~~
- ~~Explicit memory management between CPU & GPU~~
- Synchronization Barrier

AMD
together we advance_

PROGRAMMING ACROSS FRAMEWORKS/COMPILERS

OpenMP® CODE	RAJA CODE	KOKKOS CODE

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

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

`#pragma omp target`
`{ … }`
	`RAJA::forall< exec_policy >(arange, [=]`
	`(int i) { … } );`
		`Kokkos::parallel_for(M, [=] (const  int`
		`i){ … };`
		`Kokkos::fence();`

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

AMD
together we advance_

APU OPENMP PROGRAMMING

GPU CODE W/ MI300A

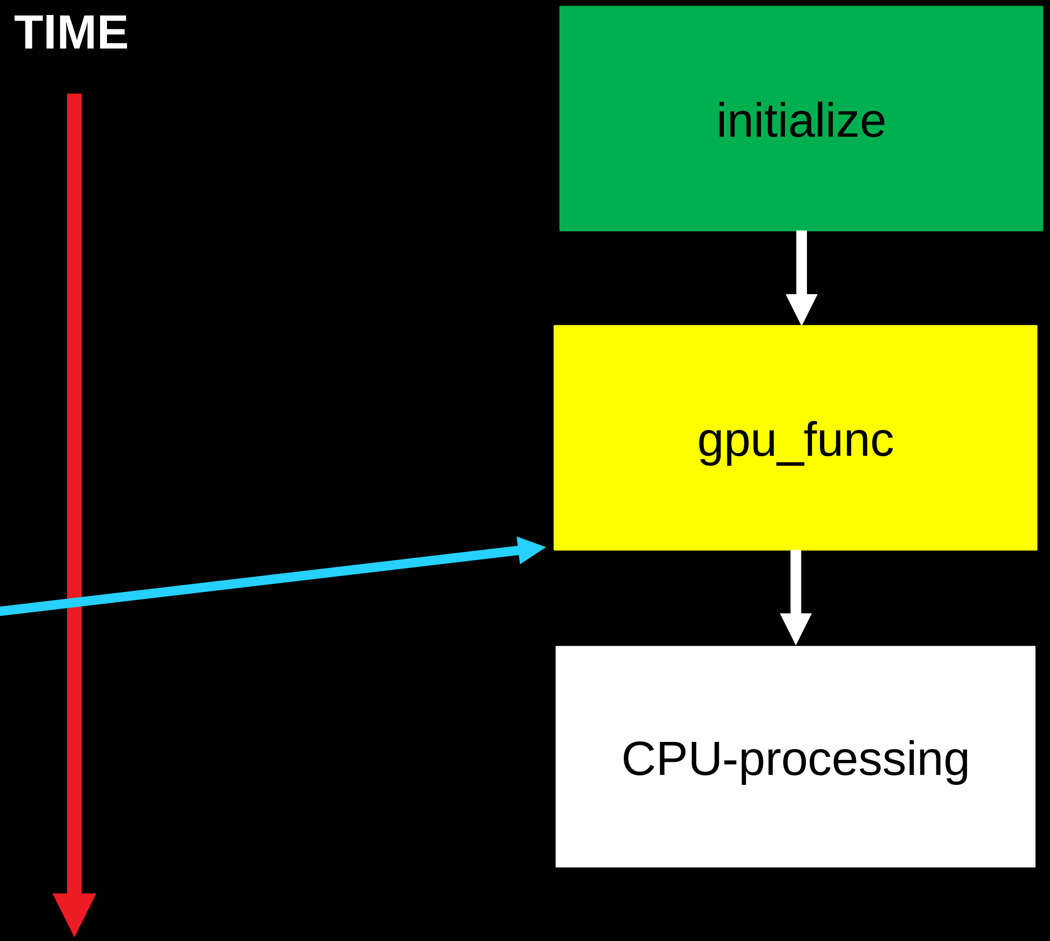
```
#pragma omp requires unified_shared_memory

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];
```



- Runtime knows it can omit copies and map clauses
- (Implicit) Synchronization Barrier

SYNCHRONIZATIONS ON AN APU

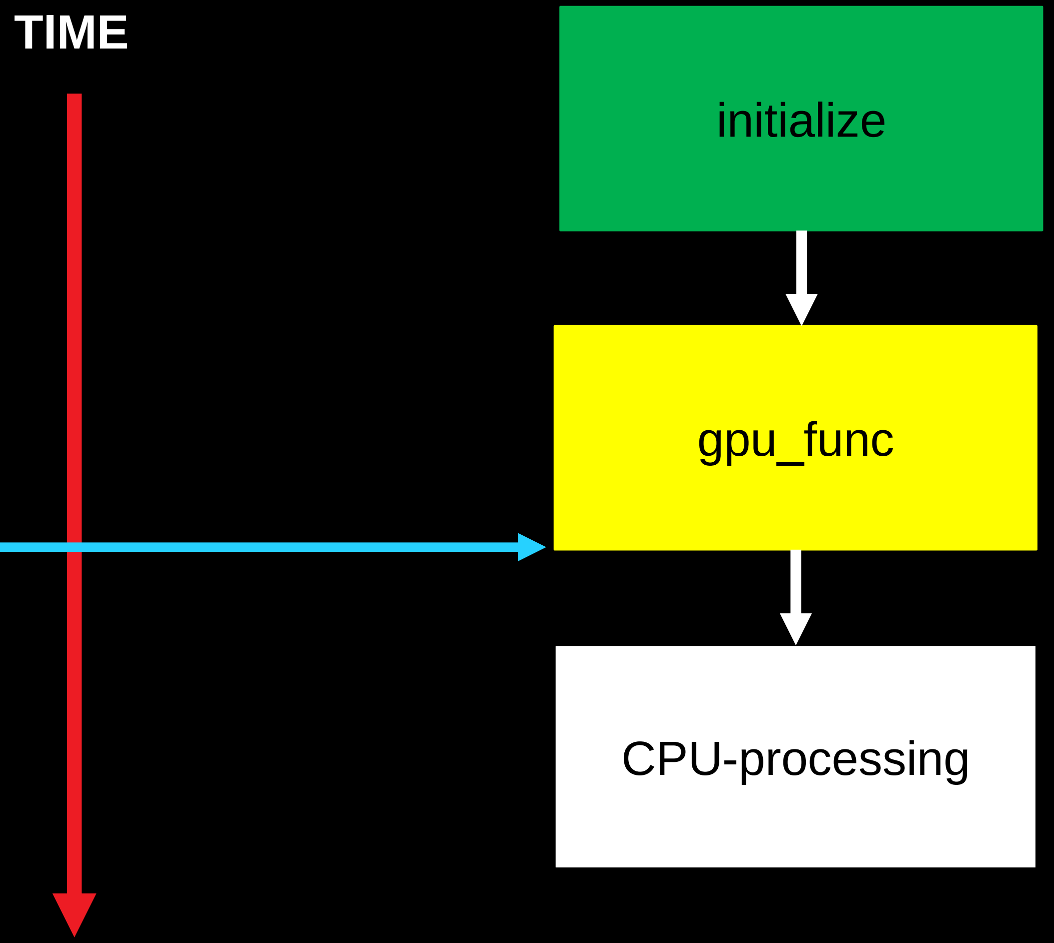
GPU CODE W/ MI300A

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

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

gpu_func<< >>(in_h, out_h, M);
hipDeviceSynchronize();


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



- Synchronization barrier requiring GPU kernel to complete

RESEARCH: OVERLAP CPU & GPU AT CACHE LINE GRANULARITY

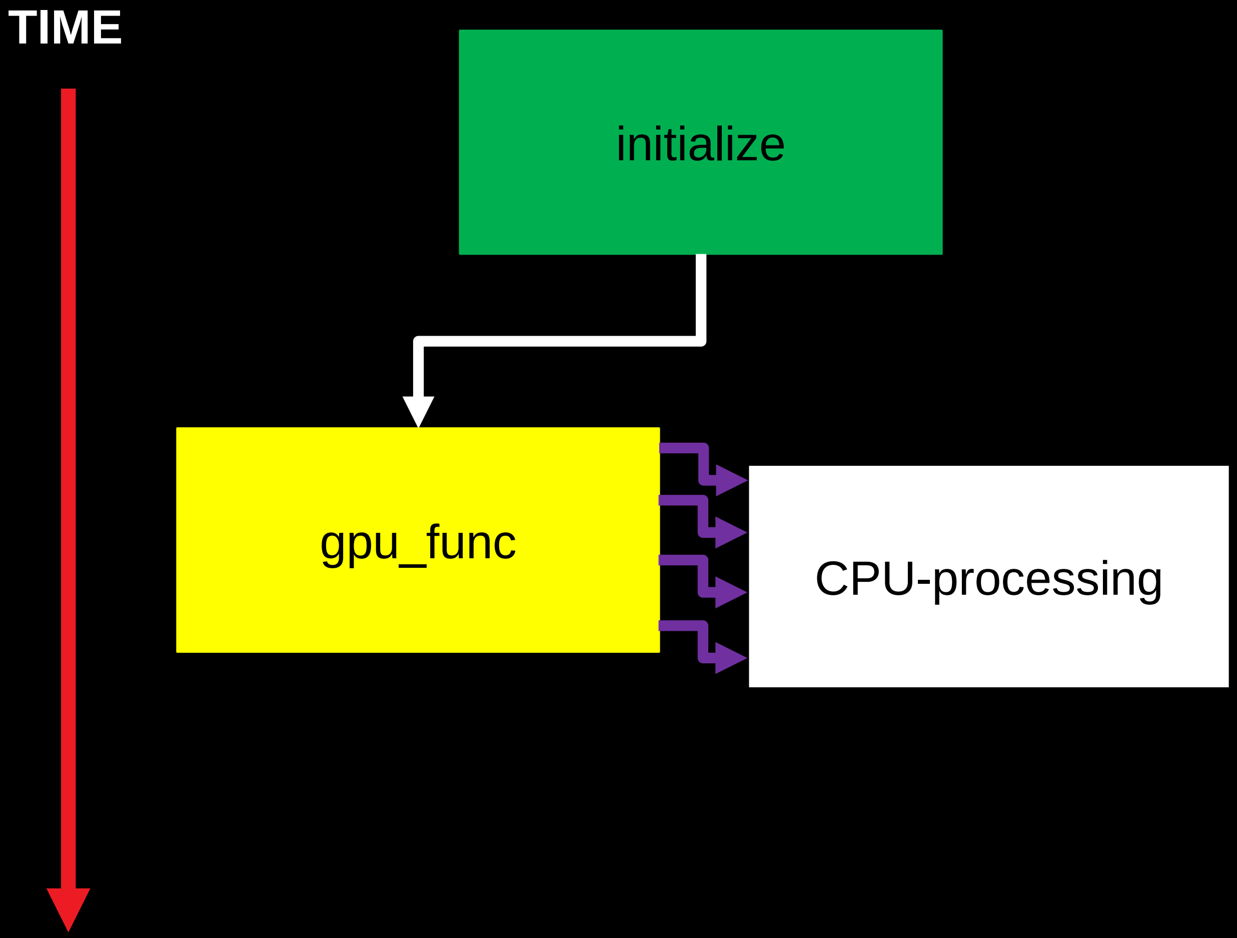
APU CODE W/ MI300A

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);
int*     flag = (int*)malloc(Isize);

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

gpu_func<< >>(in_h, out_h, M);
~~hipDeviceSynchronize();~~

for(int j = 0; j<Isize; j++){
  while(atomic_load(&flag[j]) == 0){}
  post_process(out_h[j]); // CPU-process
}
```



- Spin-loop on flag
- No Synchronization barrier; GPU kernel not required to complete
- Requires fine-grained memory allocations (128B)

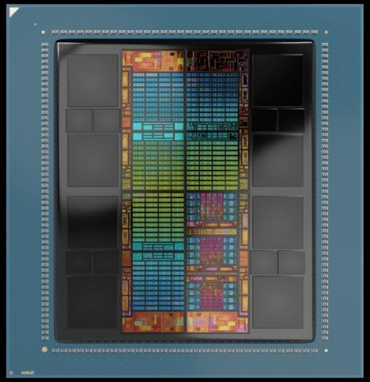
ACCELERATING LIBRARY PERFORMANCE

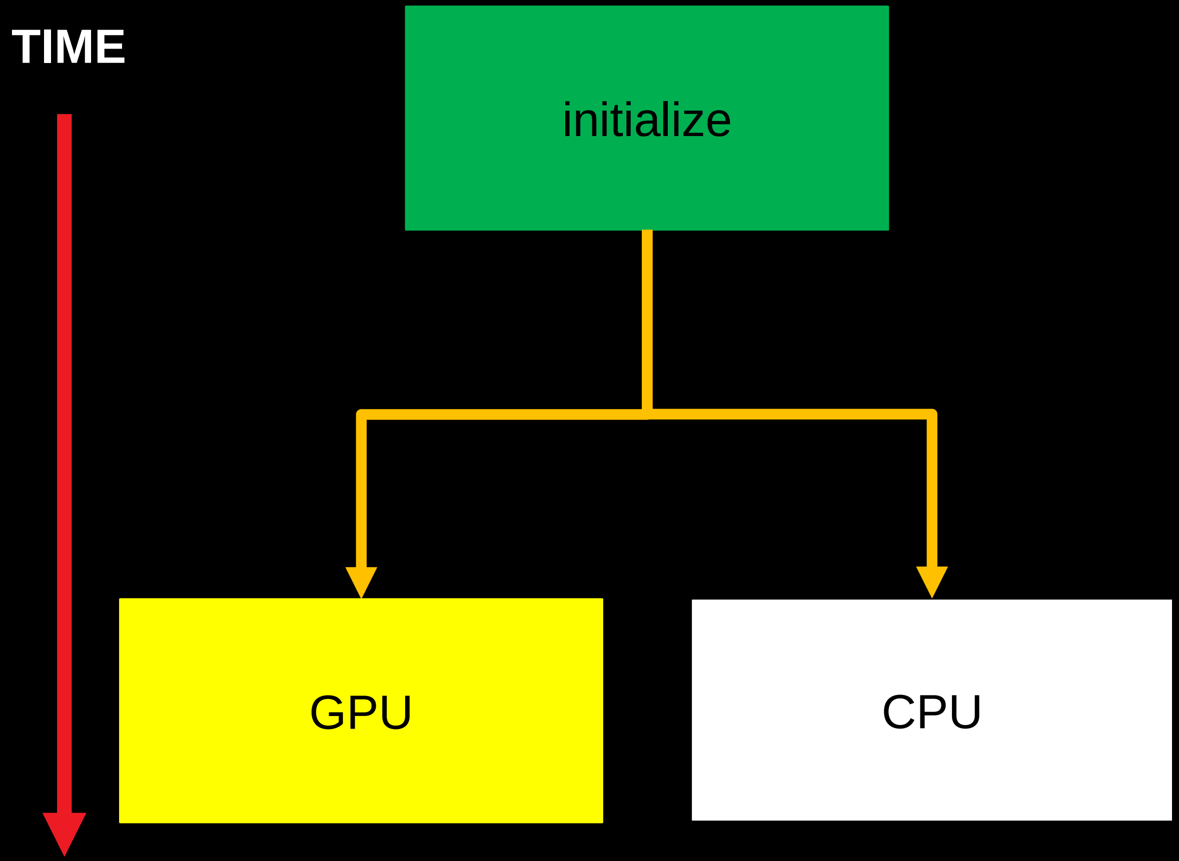
INTELLIGENT OFFLOAD

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

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

Library_Function (in_h, out_h, M);
```





HIP STANDARD PARALLELISM

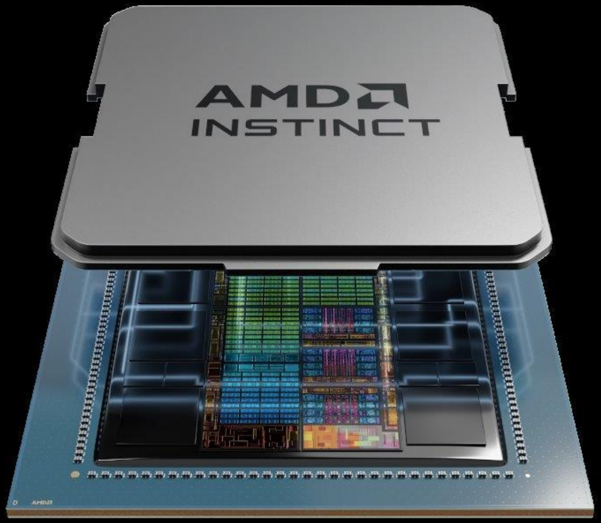
- AMD providing support for advanced C++ in LLVM: today, entirely Open Source Software (OSS)
  - Only supports par_unseq acceleration currently
  - **https://discourse.llvm.org/t/rfc-adding-c-parallel-algorithm-offload-support-to-clang-llvm/72159/3**
  - Re-uses HIP support in CLANG/LLVM and algorithms from libraries (rocThrust)
  - Available today: https://github.com/ROCmSoftwarePlatform/roc-stdpar
    - More than 23 applications tested and running (LULESH, etc.)

```
std::transform( // needs <algorithm>
    std::execution::par_unseq, // <-- needs <execution>
    indices.begin(), indices.end(), grid.begin(),
    [](size_t index){
        return expensive_calculation(index);
    }
);
```

AMD
together we advance_

CONCLUSIONS AND RECOMMENDATIONS

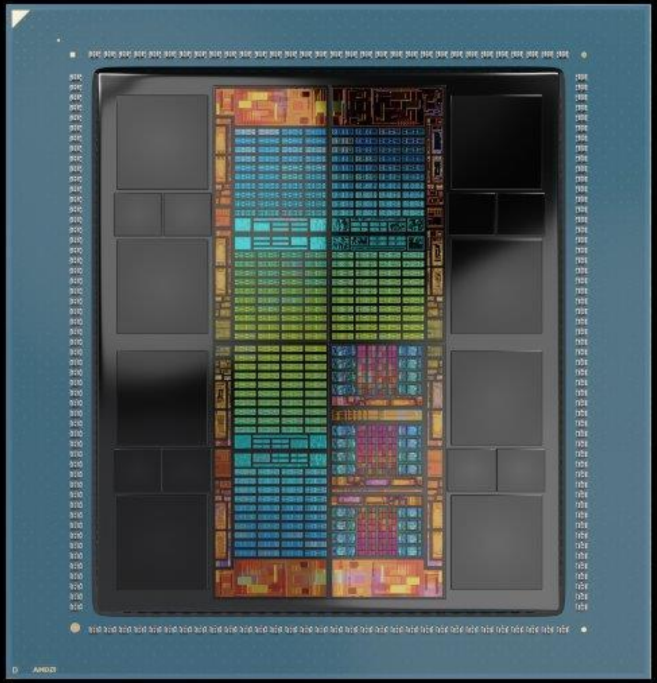
- MI300A is an APU with a unified memory space between CPUs and GPUs
- Portable programming paradigms such as RAJA, Kokkos, and OpenMP® should not require code refactoring
- HIP code should compile and run ‘out-of-the-box’
- HIP code can be optimized to:
  1) Remove redundant memory allocations and migrations
  2) Perform fine-grained (cache line) sharing between CPU and GPU
- Stay tuned for advanced capabilities such as:
  - Automatic library offload
  - Standard parallelism



Thank You and Attribution

This content was developed by many at AMD:

- Nick Malaya
- Sam Antao
- Ian Bogle
- Paul Bauman
- Bill Brantley
- Noel Chalmers
- Nick Curtis
- Austin Ellis
- Alessandro Fanfarillo
- Joe Greathouse
- Leopold Grinberg
- Michael Klemm
- Felix Kuehling
- Jakub Kurzak
- George Markomanolis
- Damon McDougall
- Jose Noudohouenou
- Corbin Robeck
- Michael Rowan
- Gina Sitaraman
- Noah Wolfe
- Johannes Dieterich
- Alex Voicu



DISCLAIMER

AMD

together we advance_

Pillars of Exascale Software

OpenMP

Established Leader in Compiler Directives

HIP

Open & Portable GPU Programming

TensorFlow PYTORCH

ML Frameworks

100% Open Software on Community Standards

Unified CPU + GPU Tools