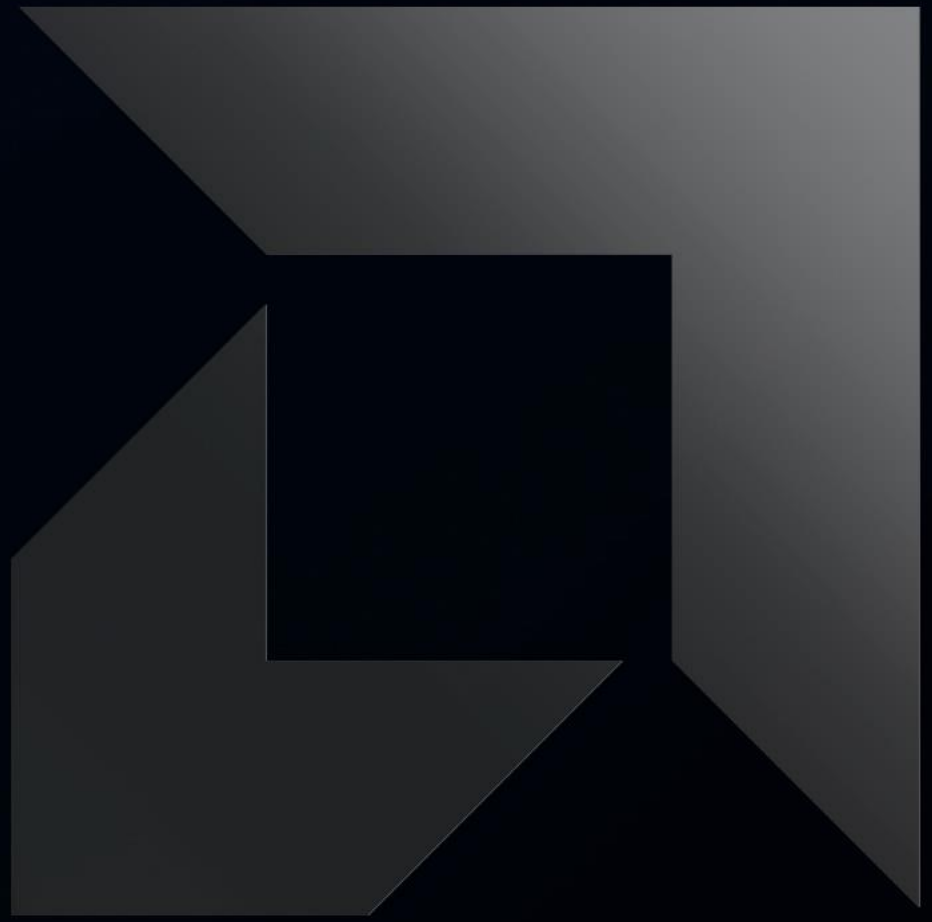


MI300A Architecture and Programming model

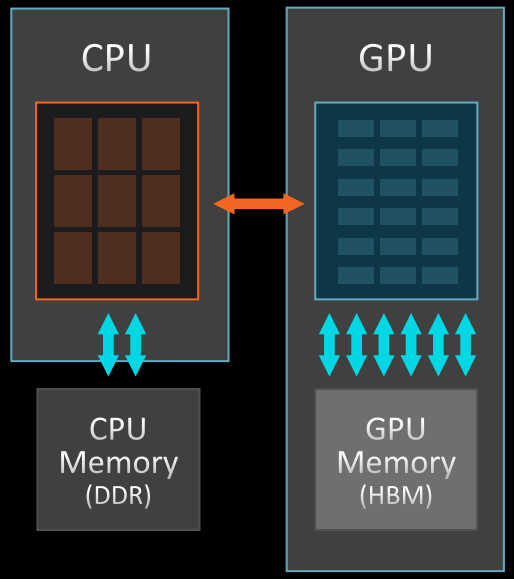
HLRS , Jan 22, 2025

Johanna Potyka, Igor Pasichnyk

AMD
together we advance_

APU ARCHITECTURE BENEFITS FOR CPU TO GPU PORTING

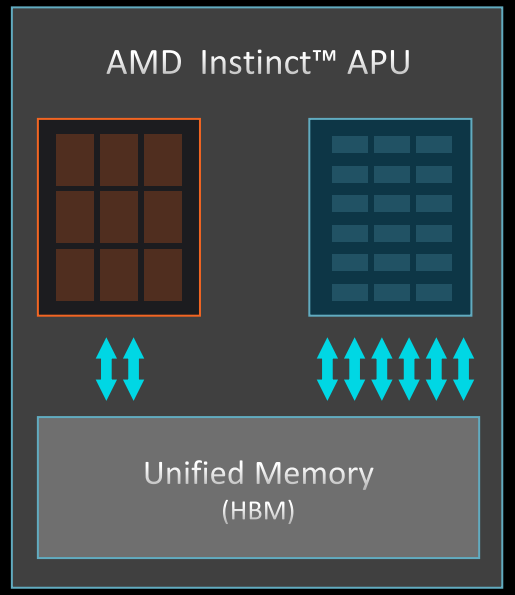
AMD CDNA™ 2
Coherent Memory Architecture



AMD
DISCRETE GPUS

AMD CDNA™ 3 Unified Memory APU Architecture

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



AMD
MI300A

AMD
together we advance_

AMD Instinct™ MI300A Accelerator

AMD INSTINCT

6 XCDs
228 AMD CDNA™ 3 compute units

AMD CDNA 3

3 CCDs
24 “Zen 4” x86 cores CPU

ZEN 4

4 IODs

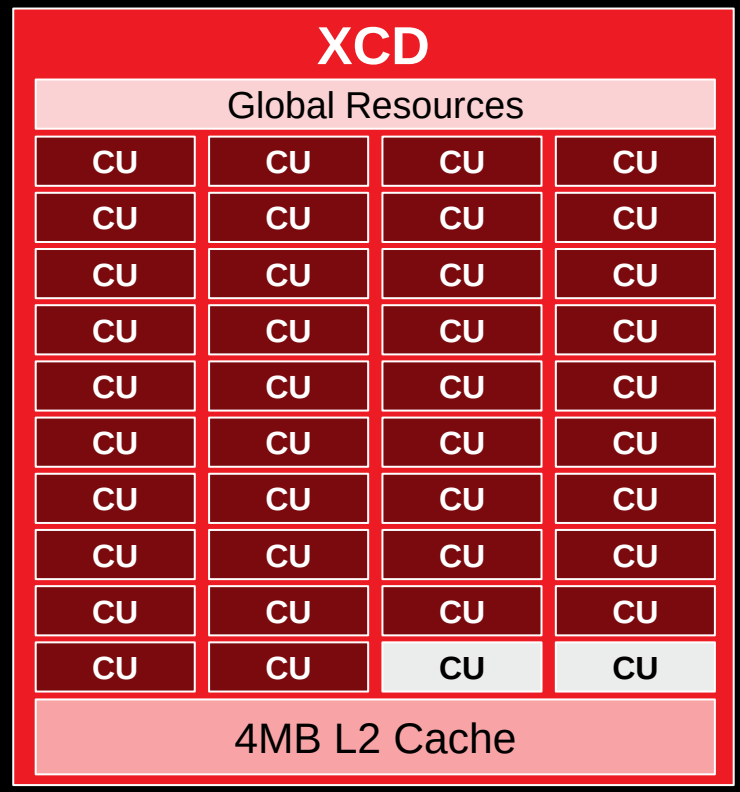
8 HBM3 stacks

256 MB
AMD Infinity Cache™ technology

3.5D packaging

AMD
together we advance_

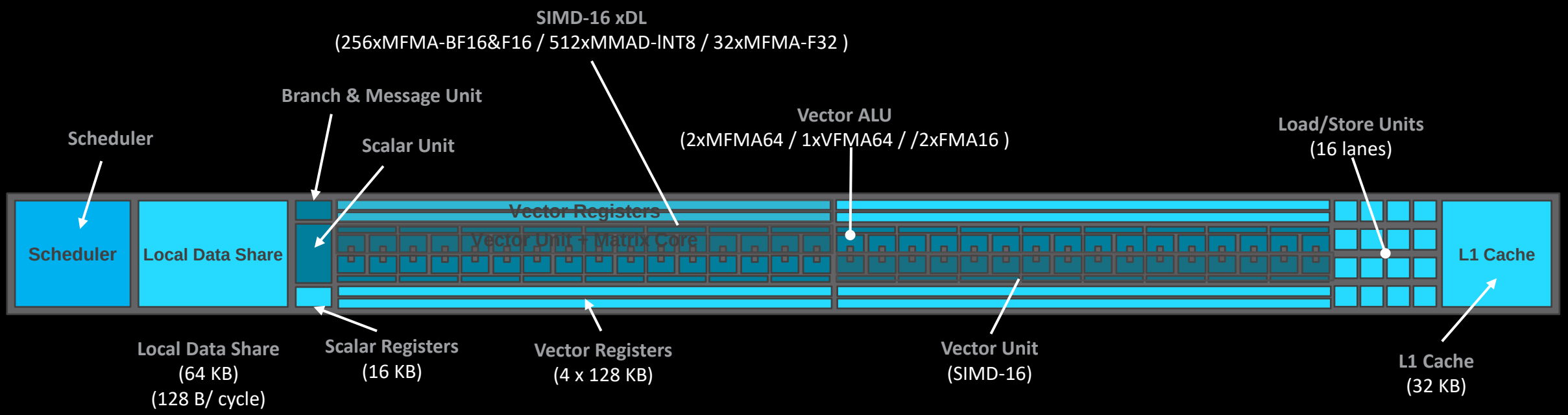
AMD Instinct™ MI300A Accelerated Processing Unit



- XCD – Accelerator Complex Die
- 38 CUs per XCD, 228 total AMD CDNA™ 3 architecture

AMD
together we advance_

AMD Instinct™ MI300A Accelerated Processing Unit



64 work-items grouped into wavefront executing “in one pass”
Each pair of **CUs** shares a 64KB, 8-way set associative instruction cache

AMD
together we advance_

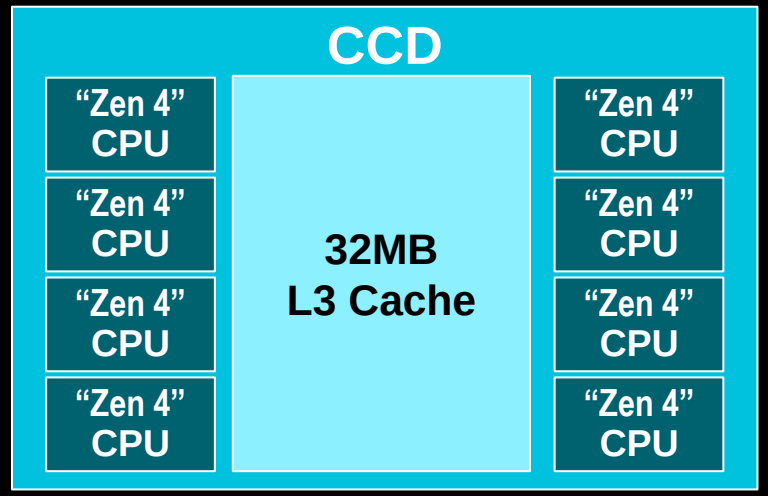
AMD Instinct™ MI300A Accelerated Processing Unit

	AMD Instinct™ MI300A
# Active CU / XCD	38
CDNA™ 3 Accelerated Compute Dies (XCD)	6
Stream Processors	38 * 6 * 64 = **14,592**
L1 Cache / CU	32 KB
L2 Cache Shared Between CUs	4 MB

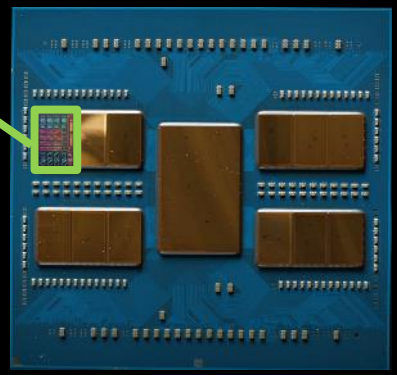
- A multiple of **14,592** threads need to run concurrently to efficiently use the GPU part of the MI300A!
- Parallelism needed to use a full node (4 APUs): 4 x 14,592 x “a few”

AMD
together we advance_

AMD Instinct™ MI300A Accelerated Processing Unit

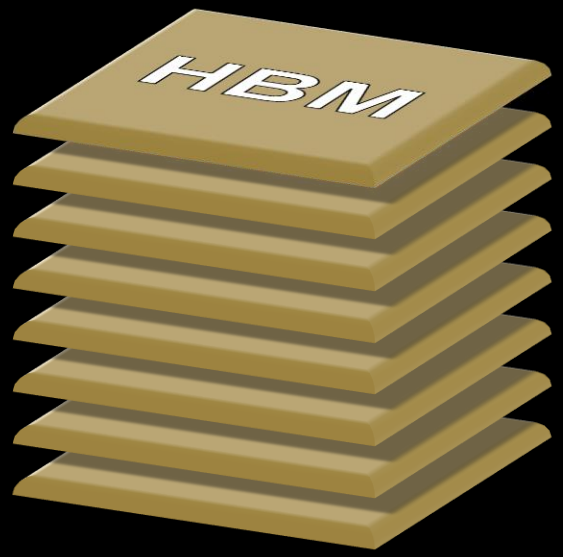


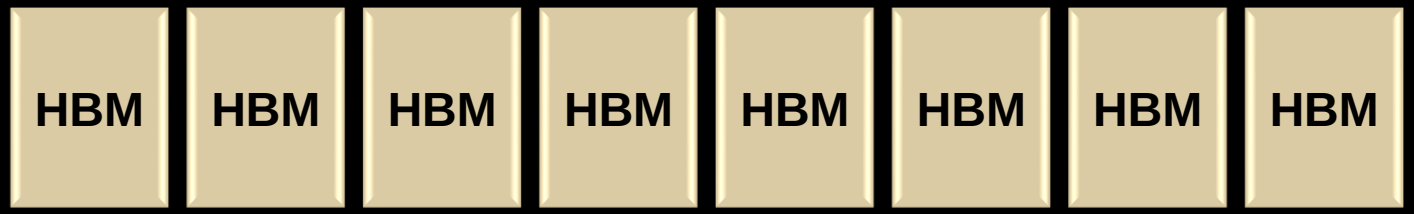
- CCD – CPU Complex Die
- 8 "Zen 4" cores per CCD, 24 total
- Leverage CCD from EPYC



AMD
together we advance_

AMD Instinct™ MI300A Accelerated Processing Unit

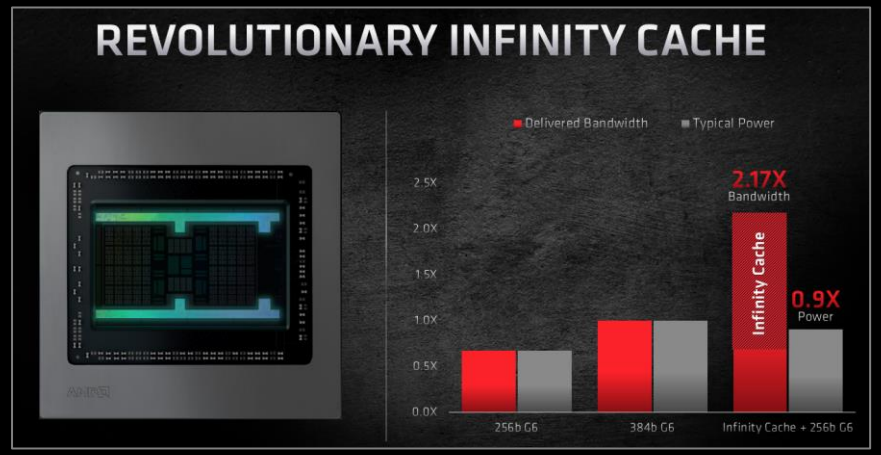


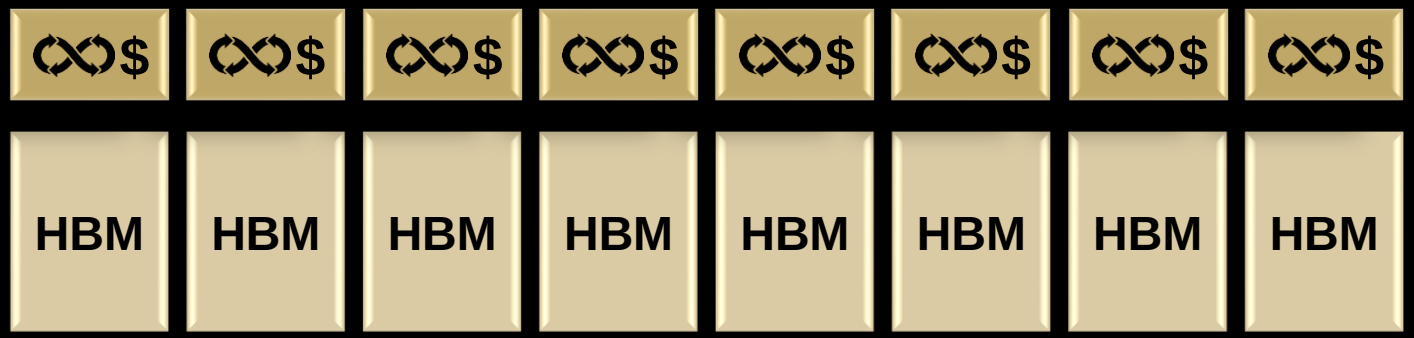


- HBM – High Bandwidth Memory
- HBM gen 3, 16GB (128 GB total)
- 665 GB/s/stack (5.3 TB/s total)
- 128 total memory channels

AMD
together we advance_

AMD Instinct™ MI300A Accelerated Processing Unit

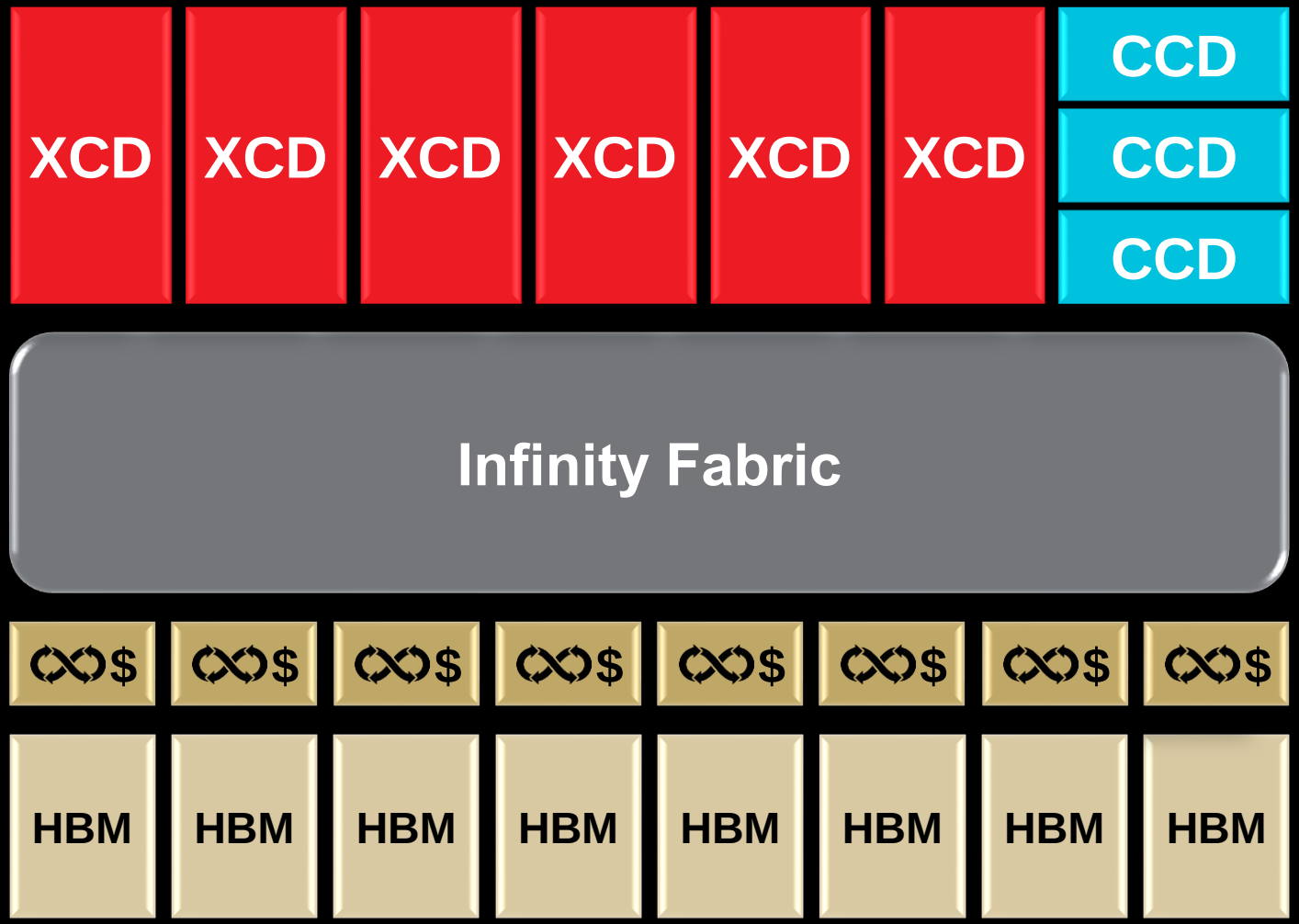


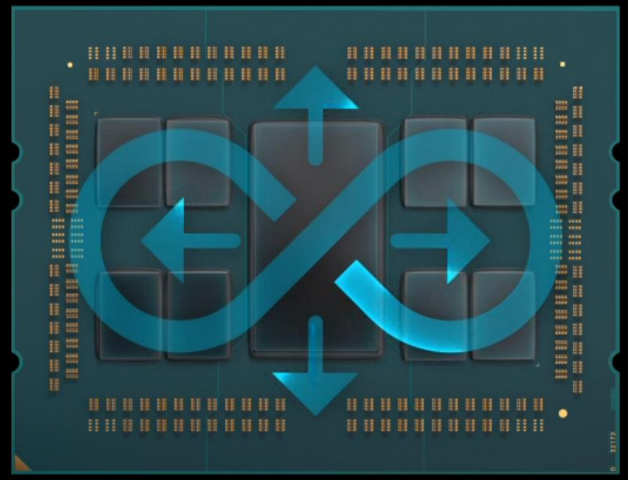


- Memory-side Infinity Cache
- 2MB/channel (256 MB total)
- BW amplification (up to 17 TB/s)

AMD
together we advance_

AMD Instinct™ MI300A Accelerated Processing Unit





- Infinity Fabric (IF)
- Fully-coherent fabric (CPU+GPU)
- Provides I/O connectivity
  - Four x16 links IF to other MI300A
    Four x16 links IF or PCIe® gen5
  - Each link at 64 GB/s/dir

AMD
together we advance_

AMD Instinct™ MI300A Accelerator

AMD INSTINCT

6 XCDs
228 AMD CDNA™ 3 compute units

AMD CDNA 3

3 CCDs
24 “Zen 4” x86 cores CPU

ZEN 4

4 IODs

8 HBM3 stacks

256 MB
AMD Infinity Cache™ technology

3.5D packaging

228 CUs can execute 64 wide wavefronts in parallel → Much parallelism needed to use the full device!

AMD
together we advance_

OpenMP offload for APUs

Recap: OpenMP® on CPUs

```
void saxpy(int n, float a, float *x, float *y) {
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();

for (int i = 0; i < n; i++) {
        y[i] = a * x[i] + y[i];
    }
    te = omp_get_wtime();
    t = te - tb;
    printf("Time of kernel: %lf\n", t);
}
```


AMD
together we advance_

Recap: OpenMP® on CPUs

```
void saxpy(int n, float a, float *x, float *y) {
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();
#pragma omp parallel for private(i) shared(x,y,a,n)
for (int i = 0; i < n; i++) {
        y[i] = a * x[i] + y[i];
    }
    te = omp_get_wtime();
    t = te - tb;
    printf("Time of kernel: %lf\n", t);
}
```


AMD
together we advance_

Recap: OpenMP® on CPUs

Tell the compiler which variables are shared among threads and which have private copies.

Start a parallel region
“Use OMP_NUM_THREADS number of threads to execute the following code”

“this line is OpenMP”

```
void sax                                    loat *y)
    doub
                                    ime();
#pragma omp parallel for private(i) shared(x,y,a,n)
for (int i = 0; i < n; i++) {
        y[i] = a *          + y[i];
                                    f\n", t);
```

Distribute the for iterations among the threads
Thread 0 gets a chunk
Thread 1 gets a chunk
…
Distribution depends on schedule

AMD
together we advance_

OpenMP® on APUs

GPU and CPU share the memory!

"This needs the unified shared memory model"

```
#pragma omp requires unified_shared_memory
void saxpy(int n, float a, float *x, float *y) {
    double t = 0.0;
    double tb, te;
    tb = omp_get_wtime();
#pragma omp target teams distribute parallel for private(i) shared(x,y,a,n)
for (int i = 0; i < n; i++) {
        y[i] = a * x[i] + y[i];
    }
    te = omp_get_wtime();
    t = te - tb;
    printf("Time of kernel: %lf\n", t);
}
```

Move this to the GPU

Distribute the work among workgroups distributed to CUs

Parallelize within workgroups (use full wavefronts)

AMD
together we advance_

OpenMP® on APUs : some flexibility and kernel tuning possible

```
!$OMP TARGET TEAMS DISTRIBUTE THREAD_LIMIT(64)
  DO JKGLO=1,NGPTOT,NPROMA
     IBL=(JKGLO-1)/NPROMA+1
     ICEND=MIN(NPROMA,NGPTOT-JKGLO+1)

!$OMP PARALLEL DO SIMD
    DO JL=1,ICEND
      CALL CLOUDSC_SCC_HOIST &
       & (1, ICEND, NPROMA, NLEV, PTSPHY,&
       & PT(:,:,IBL), PQ(:,:,IBL), &
       & BUFFER_TMP(:,:,1,IBL), BUFFER_TMP(:,:,3,IBL), BUFFER_TMP(:,:,2,IBL), BUFFER_TMP(:,:,4:8,IBL), &
       & BUFFER_LOC(:,:,1,IBL), BUFFER_LOC(:,:,3,IBL), BUFFER_LOC(:,:,2,IBL), BUFFER_LOC(:,:,4:8,IBL), &
       & PVFA(:,:,IBL), PVFL(:,:,IBL), PVFI(:,:,IBL), PDYNA(:,:,IBL), PDYNL(:,:,IBL), PDYNI(:,:,IBL), &
       & PHRSW(:,:,IBL),    PHRLW(:,:,IBL),&
       & PVERVEL(:,:,IBL),  PAP(:,:,IBL),      PAPH(:,:,IBL),&
       & PLSM(:,IBL),       LDCUM(:,IBL),      KTYPE(:,IBL), &
       & PLU(:,:,IBL),      PLUDE(:,:,IBL),    PSNDE(:,:,IBL),    PMFU(:,:,IBL),     PMFD(:,:,IBL),&
                              !---PROGNOSTIC FIELDS
       & PA(:,:,IBL),       PCLV(:,:,:,IBL),   PSUPSAT(:,:,IBL),&
                              !-- ARRAYS FOR AEROSOL-CLOUD INTERACTIONS
       & PLCRIT_AER(:,:,IBL),PICRIT_AER(:,:,IBL),&
       & PRE_ICE(:,:,IBL),&
       & PCCN(:,:,IBL),     PNICE(:,:,IBL),&
                              !---DIAGNOSTIC OUTPUT
       & PCOVPTOT(:,:,IBL), PRAINFRAC_TOPRFZ(:,IBL),&
                              !---RESULTING FLUXES
       & PFSQLF(:,:,IBL),   PFSQIF (:,:,IBL),  PFCQNNG(:,:,IBL),  PFCQLNG(:,:,IBL),&
       & PFSQRF(:,:,IBL),   PFSQSF (:,:,IBL),  PFCQRNG(:,:,IBL),  PFCQSNG(:,:,IBL),&
       & PFSQLTUR(:,:,IBL), PFSQITUR (:,:,IBL), &
       & PFPLSL(:,:,IBL),   PFPLSN(:,:,IBL),   PFHPSL(:,:,IBL),   PFHPSN(:,:,IBL),&
       & LOCAL_YRECLDP, &
       & ZFOEALFA(:,:,IBL), ZTP1(:,:,IBL), ZLI(:,:,IBL), ZA(:,:,IBL), ZAORIG(:,:,IBL), &
       & ZLIQFRAC(:,:,IBL), ZICEFRAC(:,:,IBL), ZQX(:,:,:,IBL), ZQX0(:,:,:,IBL), ZPFPLSX(:,:,:,IBL), &
       & ZLNEG(:,:,:,IBL), ZQXN2D(:,:,:,IBL), ZQSMIX(:,:,IBL), ZQSLIQ(:,:,IBL), ZQSICE(:,:,IBL), &
       & ZFOEEWMT(:,:,IBL), ZFOEEW(:,:,IBL), ZFOEELIQT(:,:,IBL), JL=JL)
    ENDDO
!$OMP END PARALLEL DO SIMD
  ENDDO
!$OMP END TARGET TEAMS DISTRIBUTE
```


AMD
together we advance_

Introspection of Data Movement on MI300A

Discrete GPU (or HSA_ XNACK=0)

```
ACC: Start kernel cloudsc_driver_gpu_scc_hoist$cloudsc_driver_gpu_omp_scc_hoist_mod_$ck_L166_1_cce$no
ACC:        flags: CACHE_MOD CACHE_FUNC AUTO_ASYNC
ACC:    mod cache:  0x462a40
ACC: kernel cache:  0x461600
ACC:   async info:  0x7f8669d0d120
ACC:    arguments: GPU argument info
ACC:            param size:  552
ACC:         param pointer:  0x7fff7a5cefa0
ACC:       blocks:  4096
ACC:      threads:  64
ACC:     event id:  0
ACC:    using cached module
ACC:    getting function cloudsc_driver_gpu_scc_hoist$cloudsc_driver_gpu_omp_scc_hoist_mod_$ck_L166_1
ACC:       stats threads=64 threadblocks per cu=1 shared=0 total shared=0
ACC:       prefer equal shared memory and L1 cache
ACC:     kernel information
ACC:              num registers :      412
ACC:         max theads per block :       64
ACC:                 shared size :        0 bytes
ACC:                  const size :        0 bytes
ACC:                  local size :      864 bytes
ACC:
ACC:     launching kernel new
ACC:     caching function
ACC: End kernel
ACC:
ACC: Start wait async(auto) from ../../../home/users/pmullown/ECMWF/dwarf-p-cloudsc/src/cloudsc_gpu/c
ACC:   async_info: 0x7f8669d0d120
ACC:    Freeing delayed free for async(auto)
ACC: End wait
ACC:
ACC: Start transfer 66 items from ../../../home/users/pmullown/ECMWF/dwarf-p-cloudsc/src/cloudsc_gpu/
ACC:   flags:
ACC:
ACC:   Trans 1
ACC:       Simple transfer of 'buffer_loc(:,:,:,:)' (2298478592 bytes)
ACC:            host ptr 203a2f200
ACC:            acc  ptr 7f6486e00000
ACC:            flags: COPY_ACC_TO_HOST FREE REL_PRESENT REG_PRESENT IMPLICIT_MAP
ACC:            release acc 7f6486e00000 from present table index 47 (ref_count 1)
ACC:            release present (acc 7f6486e00000)
ACC:            new acc ptr 0
```

APU with HSA_XNACK=1

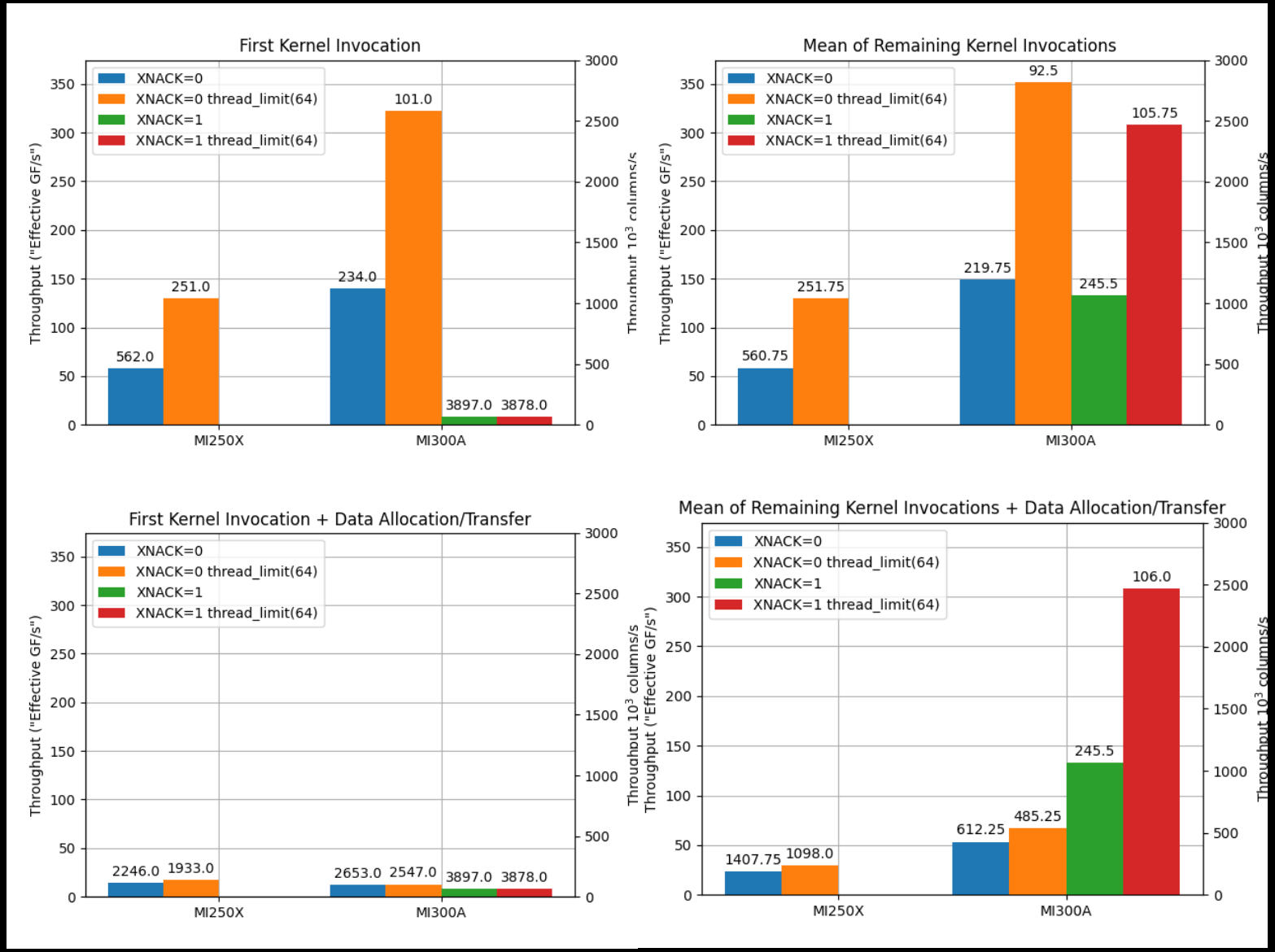
```
ACC: Start kernel cloudsc_driver_gpu_scc_hoist$cloudsc_driver_gpu_omp_scc_hoist_mod_$ck_L166_1_cce$no
ACC:        flags: CACHE_MOD CACHE_FUNC
ACC:    mod cache:  0x462a40
ACC: kernel cache:  0x461600
ACC:   async info:  (nil)
ACC:    arguments: GPU argument info
ACC:            param size:  552
ACC:         param pointer:  0x7ffc2005a060
ACC:       blocks:  4096
ACC:      threads:  64
ACC:     event id:  0
ACC:    using cached module
ACC:    getting function cloudsc_driver_gpu_scc_hoist$cloudsc_driver_gpu_omp_scc_hoist_mod_$ck_L166_1
ACC:       stats threads=64 threadblocks per cu=1 shared=0 total shared=0
ACC:       prefer equal shared memory and L1 cache
ACC:     kernel information
ACC:              num registers :      412
ACC:         max theads per block :       64
ACC:                 shared size :        0 bytes
ACC:                  const size :        0 bytes
ACC:                  local size :      864 bytes
ACC:
ACC:     launching kernel new
ACC:     synchronize
ACC:     caching function
ACC: End kernel
ACC:
ACC: Start transfer 66 items from ../../../home/users/pmullown/ECMWF/dwarf-p-cloudsc/src/cloudsc_gpu/
ACC:   flags:
ACC:
ACC:   Trans 1
ACC:       Simple transfer of 'buffer_loc(:,:,:,:)' (2298478592 bytes)
ACC:            host ptr 20482c200
ACC:            acc  ptr 20482c200
ACC:            flags: COPY_ACC_TO_HOST FREE REL_PRESENT REG_PRESENT UNIFIED_MEM IMPLICIT_MAP
ACC:            release acc 20482c200 from present table index 47 (ref_count 1)
ACC:            release present (acc 20482c200)
ACC:            new acc ptr 0
```

- When XNACK=1, we see the same host and device pointers and UNIFIED_MEM usage

AMD
together we advance_

Performance on AMD discrete GPUs (MI250X) vs. APU (MI300A)

- Distinguish between first call and mean of the final N-1 calls
  - “First touch penalty” with export HSA_XNACK=1
  - Following kernels using the same data are fast!
  - Memory pools help to avoid first touch penalty!
- For MI300A run with export **HSA_XNACK=1**



APU PROGRAMMING MODEL WITH OPENMP

CPU CODE

```
!allocation on host
ALLOCATE(var(1:N))

!compute on host
!$omp parallel do &
!$omp  private(i), shared(var)
DO i=1,N
   var(i) = …
END DO
!$omp end parallel do
!sync barrier at omp end …
…
!deallocation
DEALLOCATE(var)
```

GPU CODE

```
!allocation on host
ALLOCATE(var(1:N))

!compute on device, expl. mem movement!
!$omp target teams distribute parallel do &
!$omp  map(tofrom:var) private(i),shared(var)
 DO i=1,N
   var(i) = …
 END DO
!$omp end target teams distribute parallel do
!host-device sync barrier at omp end …
…
!deallocation
DEALLOCATE(var)
```

Costly to switch CPU -> GPU!

APU CODE

```
!$omp requires unified_shared_memory
!allocation of unified memory
ALLOCATE(var(1:N))

!compute on device, no expl. mem movement!
!$omp target teams distribute parallel do &
!$omp             private(i),shared(var)
 DO i=1,N
   var(i) = …
 END DO
!$omp end target teams distribute parallel do
!host-device sync barrier at omp end …

!deallocation of unified memory
DEALLOCATE(var)
```

Cheap CPU -> GPU with APU

- Compute kernel
- Special directive to enable unified memory
- Explicit memory management between CPU & GPU -> not needed for APU!
- Synchronization Barrier

AMD
together we advance_

Native or Low-level Languages

Heterogeneous Interface for Portability (HIP)

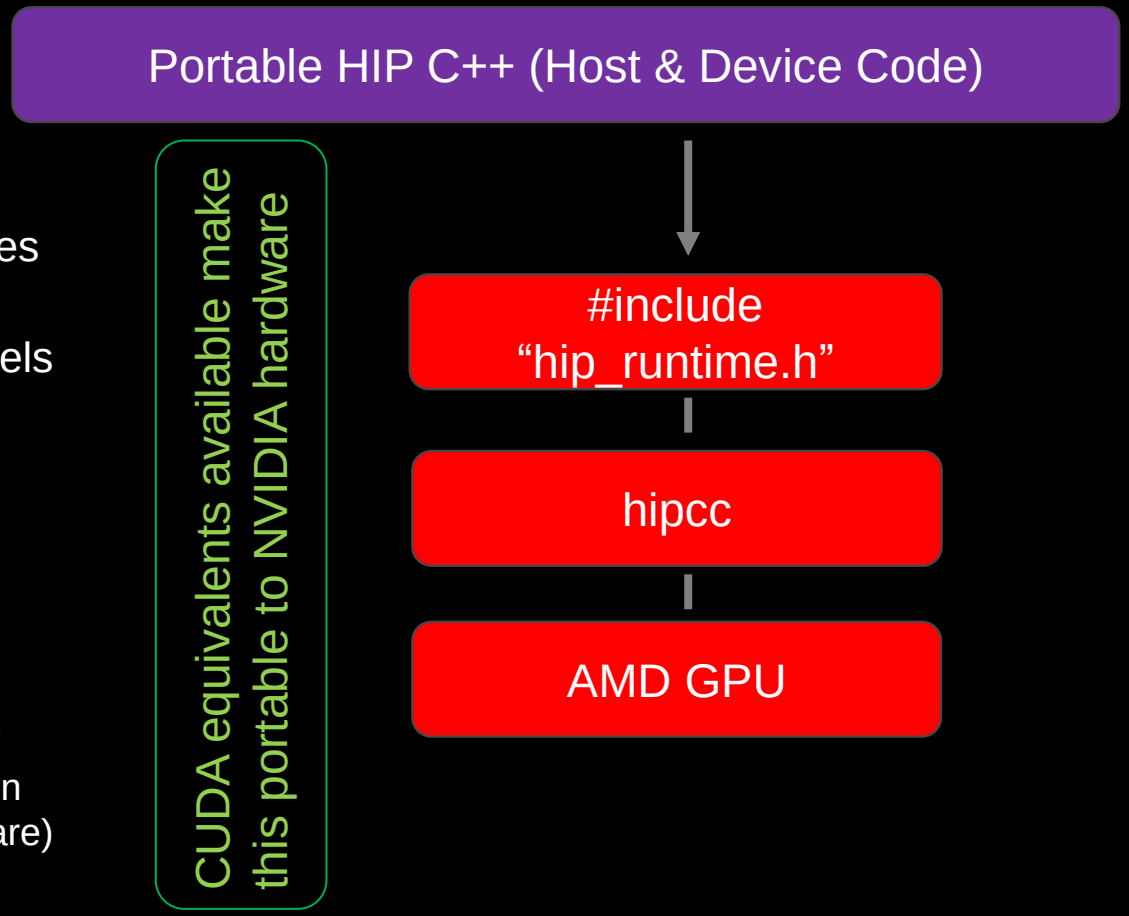
- A portable layer on top of ROCm and CUDA

Requires a different source on CPU and GPU

- Larger effort for porting
- No equivalent of CUDA Fortran available: Fortran requires C interfaces

Reccomendation: HIP for *hottest loops* and complex kernels only if you start from a CPU code

- If already CUDA ported: Converting CUDA to HIP is straightforward
  - Hipify scripts do majority of the work
  - Still requires optimization effort to get best performance
    - e.g. a wavefront has 64 threads executing the same instruction (different compared to 32 threads per warp on NVIDIA hardware)



AMD
together we advance_

APU PROGRAMMING MODEL WITH HIP

C++ example, Fortran only possible with C bindings and Interface to C for GPU kernels

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

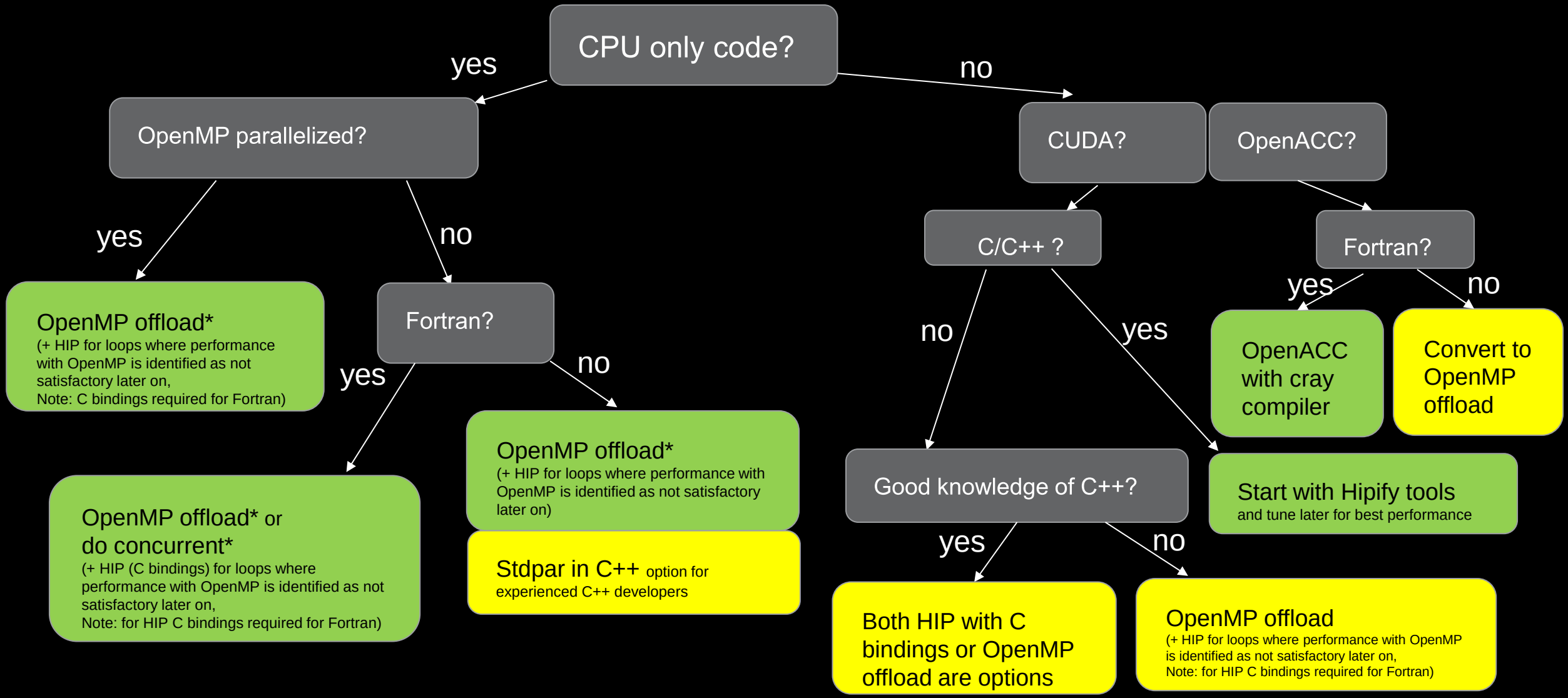
```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);

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];
```

- Compute kernel
- GPU memory allocation on Device -> no copies for host and device on APU!
- Explicit memory management between CPU & GPU -> not needed for APU!
- Synchronization Barrier

AMD
together we advance_

*Note that OpenMP target and do concurrent can be compiled for the host -> allows initial porting preparation on CPU

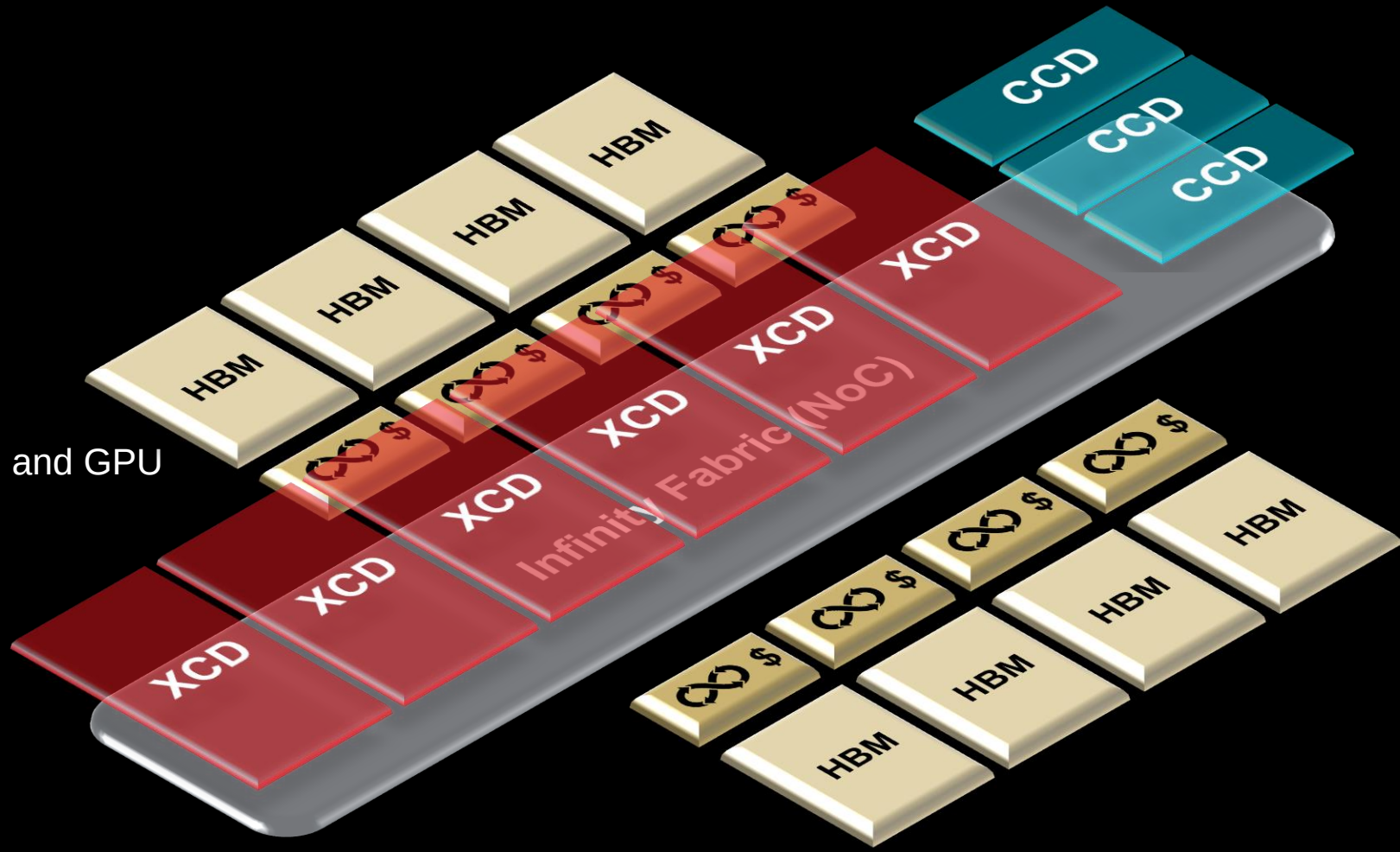
Most common decisions how to port to Hunter



AMD
together we advance_

Summary

- AMD Instinct MI300A
  - Unified memory
  - 228 CUs (wavefront: 64)
  - 128 GB HBM
  - Last level cache shared between CPU and GPU



- Programming the APU:
  - Unified shared memory / APU programming model
  - No data movement needed with HSA_XNACK=1
  - Most common: OpenMP offload, HIP, stdpar / do concurrent

AMD
together we advance_

Further Resources

GPU / APU Training:

- AMD GPU programming course: https://fs.hlrs.de/projects/par/events/2024/GPU-AMD/
- AMD GPU OpenMP programming course: https://fs.hlrs.de/projects/par/events/2024/GPU-AMD2/
- Training examples: https://github.com/AMD/HPCTrainingExamples

MI300 White Paper:

- https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-3-white-paper.pdf

AMD Blog:
https://rocm.blogs.amd.com/

ROCm Documentation:

https://rocm.docs.amd.com/

OpenMP ported example code:

- OpenFoam on APU Code: https://github.com/ROCm/OpenFOAM_HMM
- OpenFoam on APU Paper: https://ieeexplore.ieee.org/abstract/document/10528925

AMD
together we advance_

DISCLAIMERS AND ATTRIBUTIONS

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2024 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, Radeon™, Instinct™, EPYC, Infinity Fabric, ROCm™, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

HPE is a registered trademark of Hewlett Packard Enterprise Company and/or its affiliates.

The OpenMP ® name and the OpenMP ® logo are registered trademarks of the OpenMP Architecture Review Board

AMD
together we advance_


AMD INSTINCT
AMD EPYC
AMD