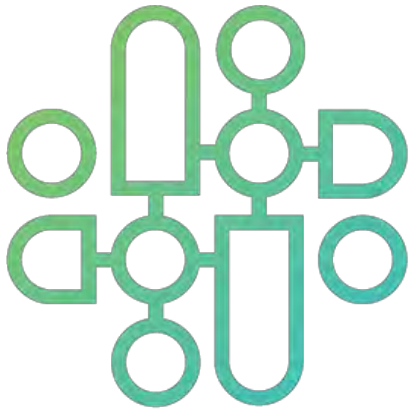
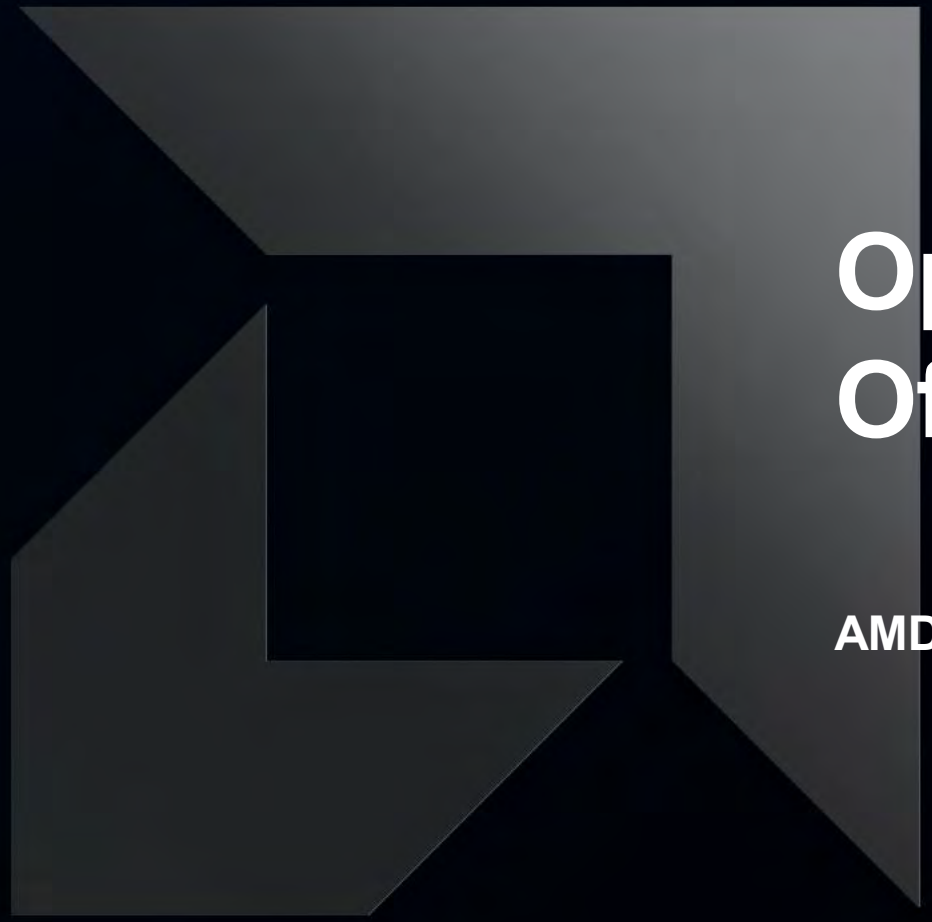
OpenMP

SC23 Booth Talk Series



OpenMP Target Offloading for AMD GPUs

Jan-Patrick Lehr, AMD

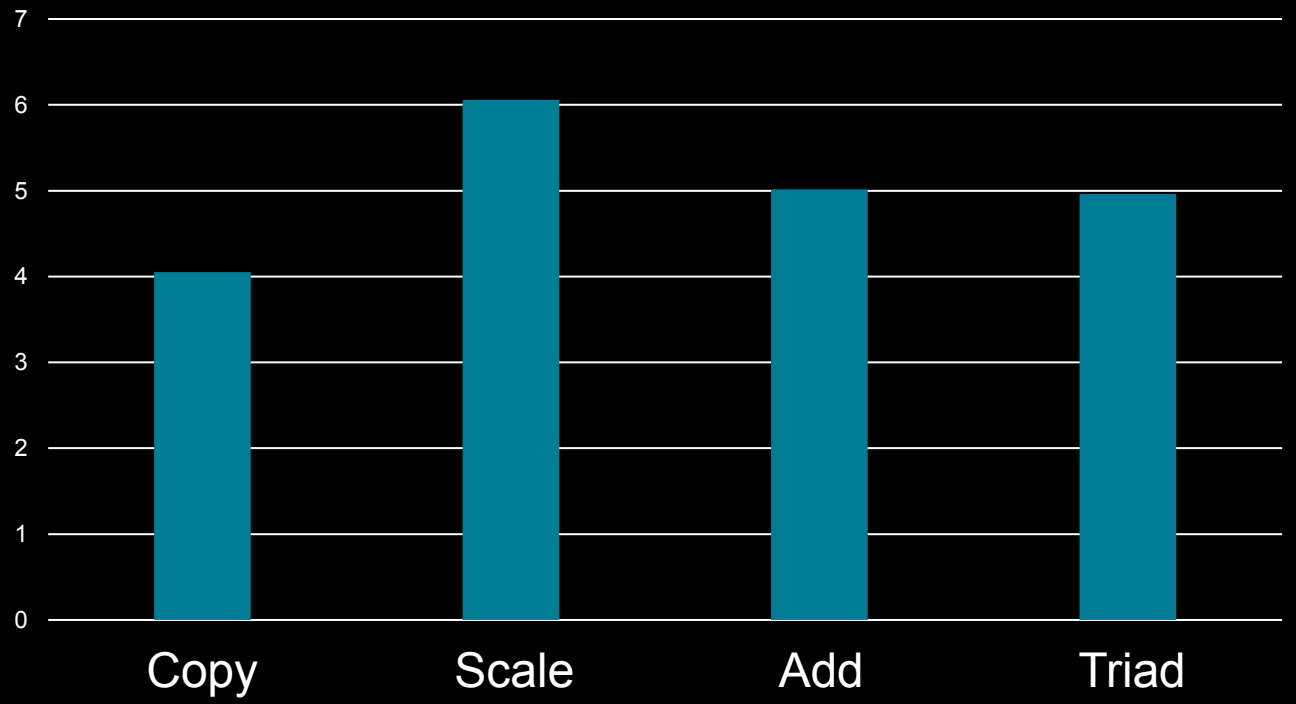


OpenMP® Target Offloading for AMD GPUs

AMD ROCm OpenMP Team

AMD
together we advance_

Relative Bandwidth Increase GPU vs CPU



Agenda

1. Overview of the ROCm™ software stack
2. OpenMP target offload
3. OpenMP® target offload example
4. OpenMP® target offload example w/ unified shared memory
5. Data environments, zero copy, and unified memory

ROCm™

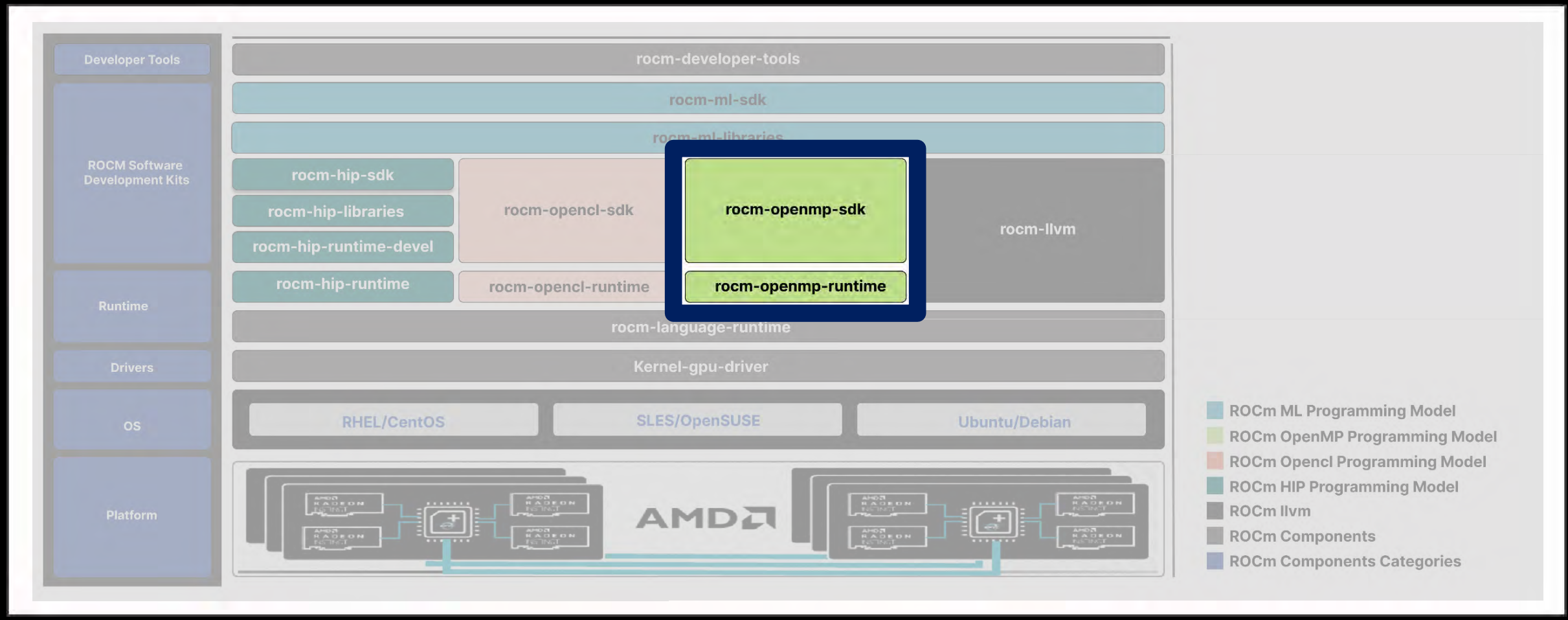
ROCm™ integrates components for **HPC** and **AI** workloads

OpenMP®-based **GPU** offloading is an integral part of **ROCm**™ releases

OpenMP®-based **GPU** offloading provides **standardized**, **portable** access to GPU compute

AMD is invested in **open source** and **open technology**

ROCm™ Software Stack (Meta Packages)



OpenMP® Target Offload

OpenMP® Target Offload

- OpenMP® language-feature to use an accelerator (e.g., a GPU) for parts of your program
- Enables standardized way for offloading to an accelerator as opposed to CUDA® or HIP
- Expressed in terms of `target` regions
- Target regions have a data environment that is maintained/manipulated via `map` clauses
- Target regions execute one or multiple teams of threads on a device
- And much more …

OpenMP® Target Offload Code Example

```
int main(int argc, char **argv) {
  int *vals = new int[1024];

  #pragma omp target teams distribute parallel for map(vals[0:1024])
  for(int i = 0; i < 1024; ++i) {
    vals[i] = 1;
  }

  for(const auto vi : vals) {
    std::cout << vi << '\n';
  }
  return 0;
}
```

OpenMP® Target Offload Code Example

```
int main(int argc, char **argv) {
  int *vals = new int[1024];

  #pragma omp target teams distribute parallel for map(tofrom: vals[0:1024])
  for(int i = 0; i < 1024; ++i) {
    vals[i] = 1;
  }

  for(const auto vi : vals) {
    std::cout << vi << '\n';
  }
  return 0;
}
```

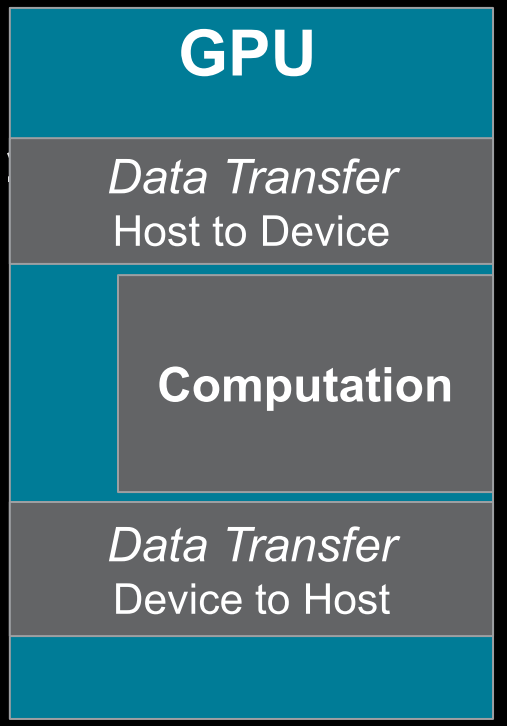
OpenMP® Target Offload Code Example

```
int main(int argc, char **argv) {
  int *vals = new int[1024];

  #pragma omp target teams distribute parallel for map(vals[0
  for(int i = 0; i < 1024; ++i) {
    vals[i] = 1;
  }

  for(const auto vi : vals) {
    std::cout << vi << '\n';
  }
  return 0;
}
```

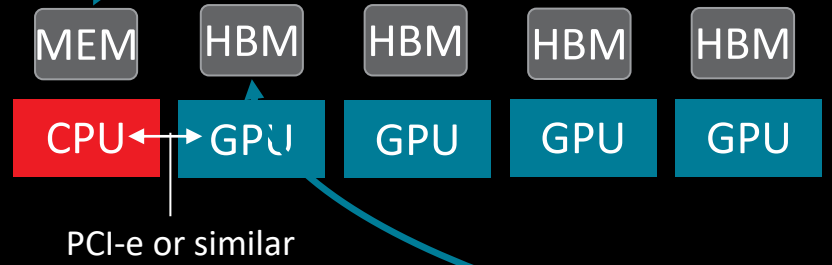
Executes on the GPU

Data Environments and Memory Transfer

```
int main(int argc, char **argv) {
  int *vals = new int[1024];

  #pragma omp target teams distribute parallel for map(vals[0:
  for(int i = 0; i < 1024; ++i) {
    vals[i] = 1;
  }

  for(const auto vi : vals) {
    std::cout << vi << '\n';
  }
  return 0;
}
```



OpenMP® Target Offload on Discrete GPUs (Default Mode)



OS allocators allocate host memory

```
int main() {
  int *vals = new int[1024];

  #pragma omp target map(tofrom: vals[0:1024])
  {
    for(int i = 0; i < 1024; i++)
      vals[i] = i;
  }
}
```

- Allocate `d_vals` 1024 integers size on GPU HBM
- (before kernel launch) copy a into `d_vals`
- (after kernel completion) copy `d_vals` into `vals`

For best performance, programmers minimize transfers between host and device by placing map clauses at the beginning and ending of an application

Unified Shared Memory (USM)

CPU CODE

```
double* in = (double*)malloc(Msize);
double* out = (double*)malloc(Msize);

 for (int i=0; i<M; i++)
   in[i] = …;



 for (int i=0; i<M; i++)
   out[i] = ... in[i] ...;




for (int i=0; i<M; i++)
  … = out[i];
```

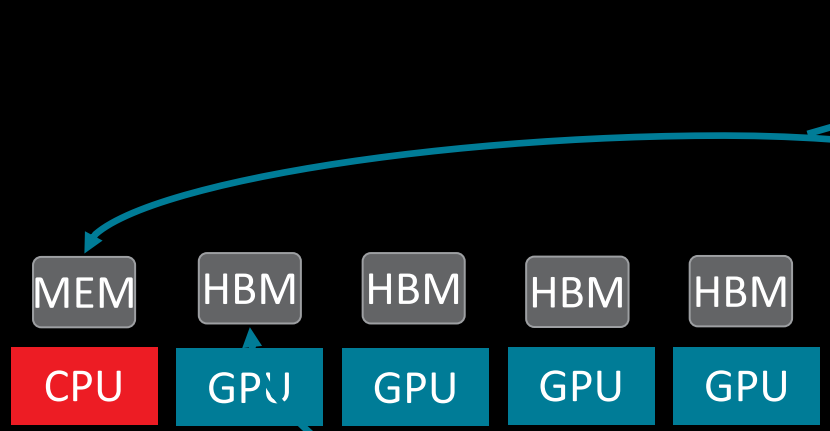
W/O UNIFIED SHARED MEMORY

```
double* in = (double*)malloc(Msize);
double* out = (double*)malloc(Msize);

for (int i=0; i<M; i++)
   in[i] = …;

#pragma omp target teams distribute \
            parallel for \
            map(to:in[0:Msize]) \
            map(from:out[0:Msize])
  for (int i=0; i<M; i++)
    out[i] = ... in[i] ...;



for (int i=0; i<M; i++)
  … = out[i];
```

UNIFIED SHARED MEMORY

```
#pragma omp require unified_shared_memory

double* in = (double*)malloc(Msize);
double* out = (double*)malloc(Msize);

for (int i=0; i<M; i++)
   in[i] = …; //writes GPU mem directly

#pragma omp target teams distribute \
            parallel for
  for (int i=0; i<M; i++)
    out[i] = ... in[i] ...;




for (int i=0; i<M; i++)
  … = out[i]; //reads GPU mem directly
```

OpenMP® Target Offload on Discrete GPUs (USM Mode)



OS allocators allocate host memory

```
#pragma omp requires unified_shared_memory
int main() {
  int *vals = new int[1024];

  #pragma omp target map(tofrom: vals[0:1024])
  {
    for(int i = 0; i < 1024; i++)
      vals[i] = i;
  }
}
```

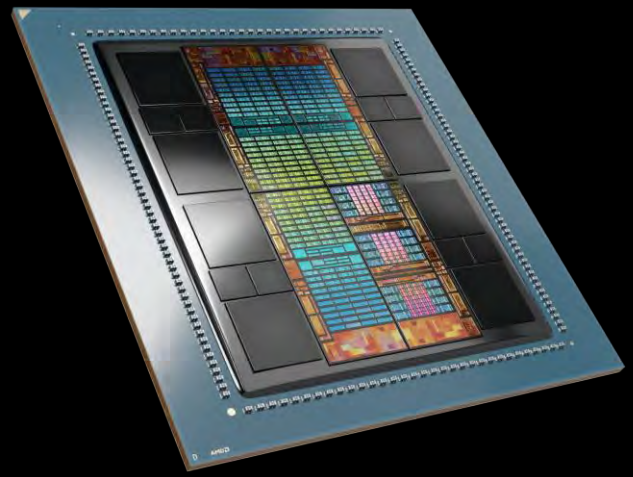
map: just pass host pointer to kernel

Accessing a page address located on system memory provokes page migration to HBM

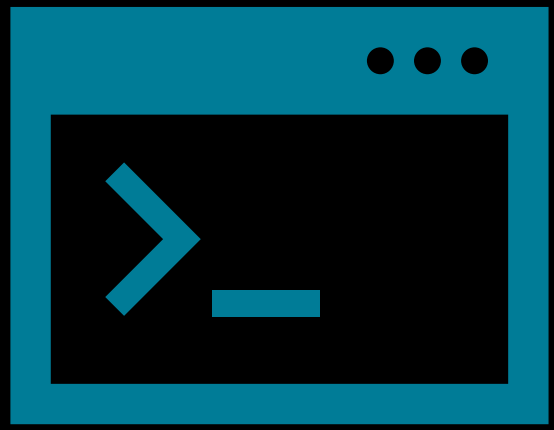
Driver handles page migrations. Migration depends on allocator being used on host

Zero Copy Benefits on Unified Memory Architecture

No need to explicitly transfer data to the GPU and back



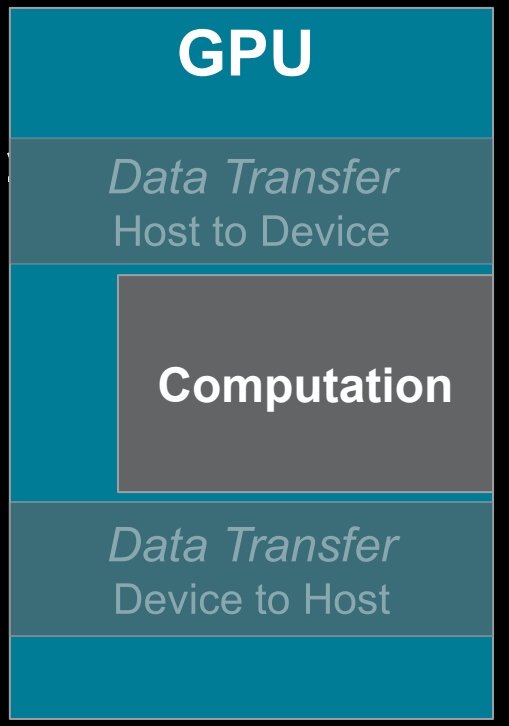
Single memory shared between host and GPU



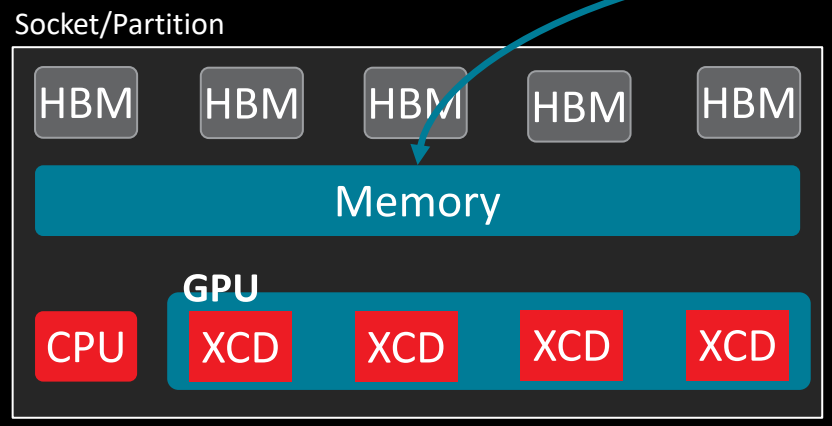
Transparently enabled in the runtime: No special flags needed

Data Environments and Zero Copy

```
int main(int argc, char **argv) {
  int *vals = new int[1024];

  #pragma omp target teams distribute parallel for map(vals[0:
  for(int i = 0; i < 1024; ++i) {
    vals[i] = 1;
  }

  for(const auto vi : vals) {
    std::cout << vi << '\n';
  }
  return 0;
}
```



OpenMP® on APUs (MI300A): Zero-Copy Mode



OS allocators allocate unified memory*

```
int main() {
  int *vals = new int[1024];

  #pragma omp target map(tofrom: vals[0:1024])
  {
    for(int i = 0; i < 1024; i++)
      vals[i] = i;
  }
}
```

map: just pass pointer to kernel

No page migration necessary upon page touch
(in this single socket example)

* Interleaved allocation across the HBMs on the socket

ROCm™ Compiler Behavior (MI300A)

Compiler Flag: --offload-arch=gfx942		Programming Mode	
		Default (non-unified_shared_memory)	unified_shared_memory*
Runtime State	Unified-Memory-Enabled	Zero-copy	Zero-copy
	Unified-Memory-Disabled	Copy	RT Error

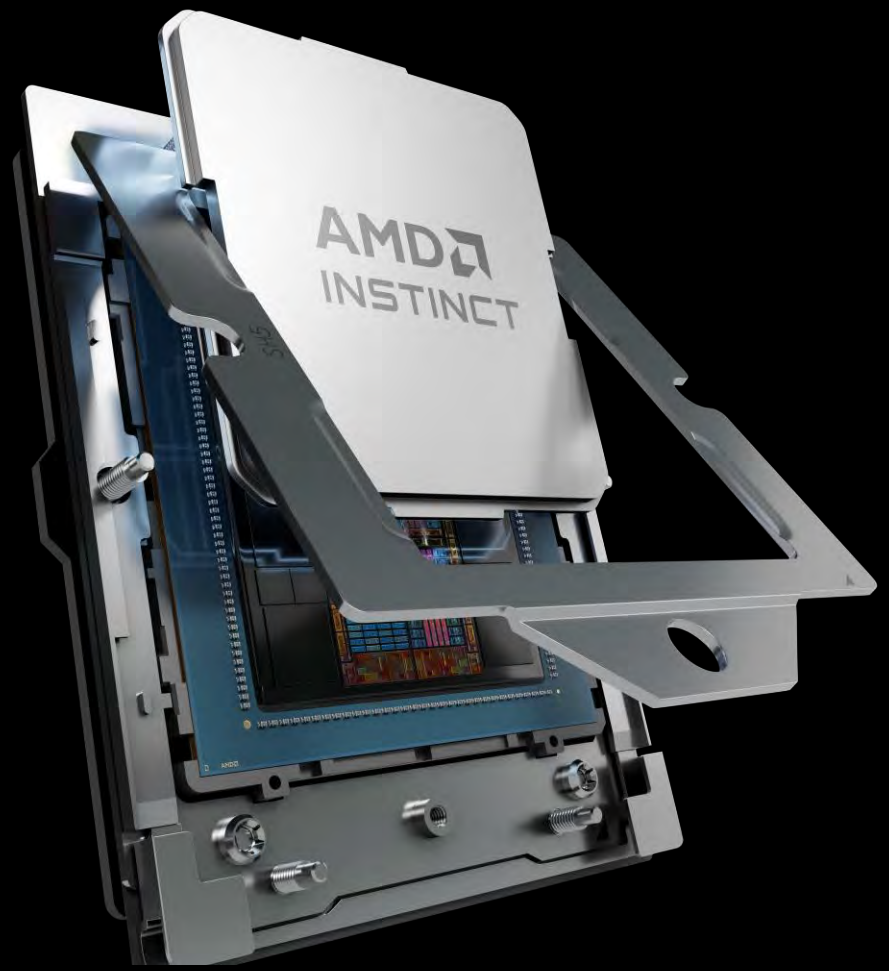
* One of the options:
  - `#pragma omp requires unified_shared_memory`
  - `-fopenmp-force-usm` (preview for future ROCm™ release)

Details of AMD Zero Copy vs. OpenMP® Unified Shared Memory

- Zero Copy is a ROCm™ OpenMP® offloading-runtime feature
- Enables execution of OpenMP® programs without explicit data copies*
- Code generation is unaffected
  - OpenMP® program uses explicit map clauses
- Requires hardware/driver support and may not work across all existing devices
- Enabled via environment variable on supported devices

*Except for a specific case of global variables

- Unified Shared Memory is a concept in the OpenMP® standard
- Eliminates the need for data environments via explicit map clauses
- Unified Shared Memory implies code generation that assumes host memory can be accessed
- Requires hardware/driver support and may not work across all existing devices
- Enabled via
  - #pragma omp requires unified_shared_memory
  - -fopenmp-force-usm (future ROCm™ release)



Learn more

- **Programming AMD GPUs with OpenMP®**
  - Nov. 15 / 2pm @ AMD booth
- **Specialized Kernels for Optimizing GPU Offload in OpenMP**
  - Workshop on Accelerator Programming and Directives

ROCm™ Website

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

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.


AMD

OpenMP®

SC23 Booth Talk Series

openmp.org — OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc23 — OpenMP SC23 booth talk videos and presentations