AMD Compilers for AMD Instinct™ Accelerators

AMD ROCm Compiler Team

Speaker: JP Lehr

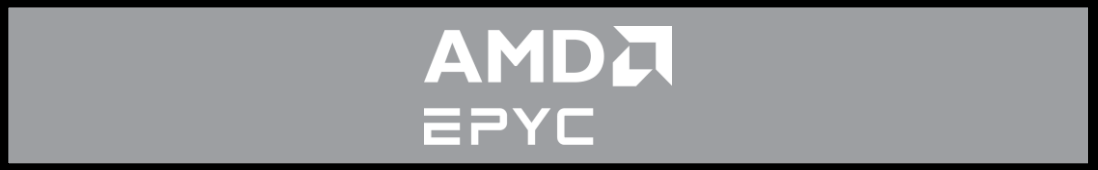
AMD
together we advance_

Agenda

1. AMD Compiler Ecosystem
2. AMD Instinct™ Accelerators
3. OpenMP® Target Offload
4. USM via Zero Copy

AMD Compiler Ecosystem

Developing for AMD Hardware



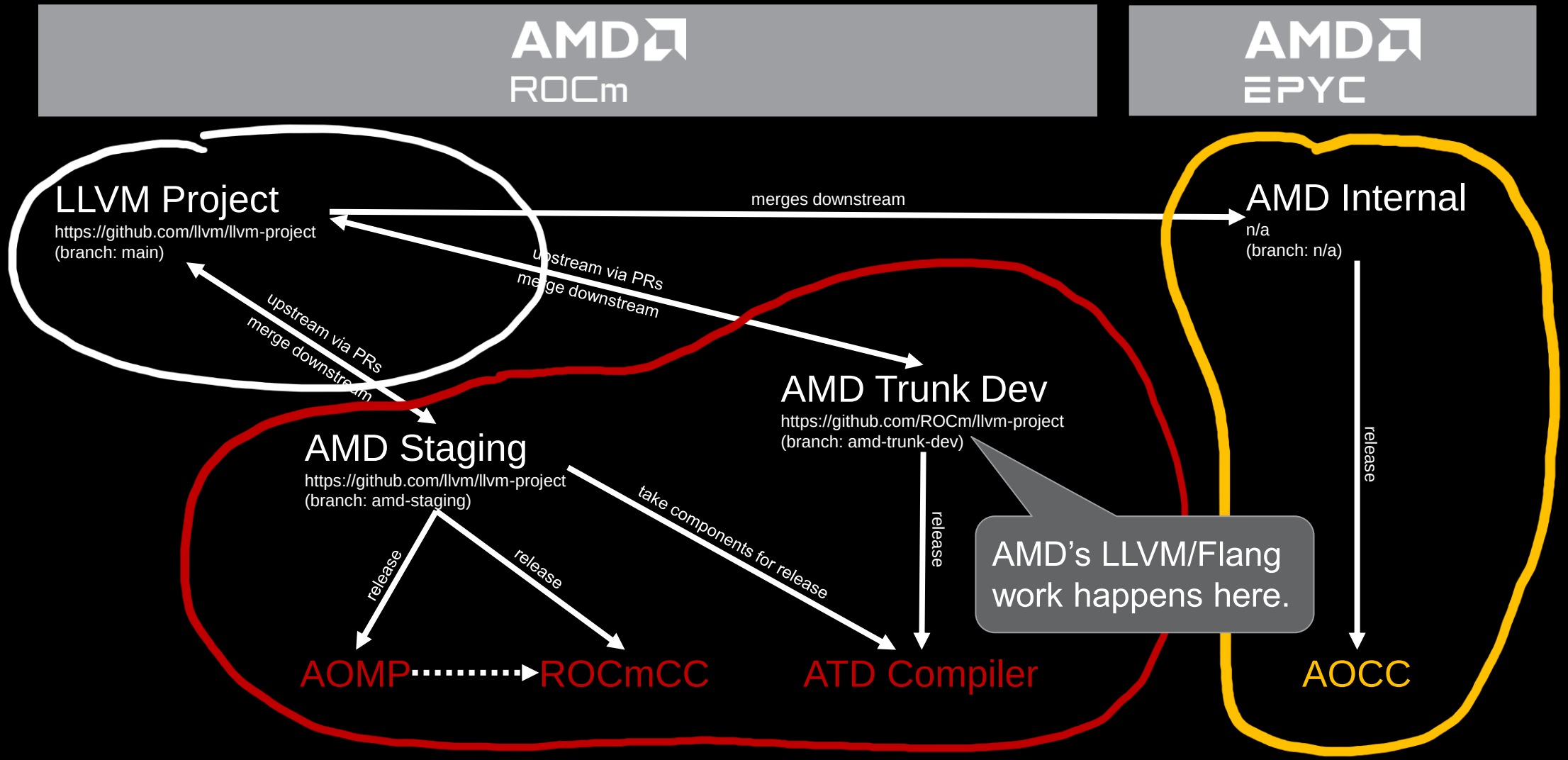
Heterogenous-computing Interface for Portability (HIP)
OpenMP® API
Machine Learning Frameworks Acceleration Libraries
ROCm™ Communication Libraries (RCCL)

AMD Optimized CPU Compiler (AOCC)
AMD Optimized CPU Libraries (AOCL)
AMD ZenDNN
AMD µProf

In addition to numerous options with open source, community tools

AMD Compiler Relationship – C, C++, Fortran

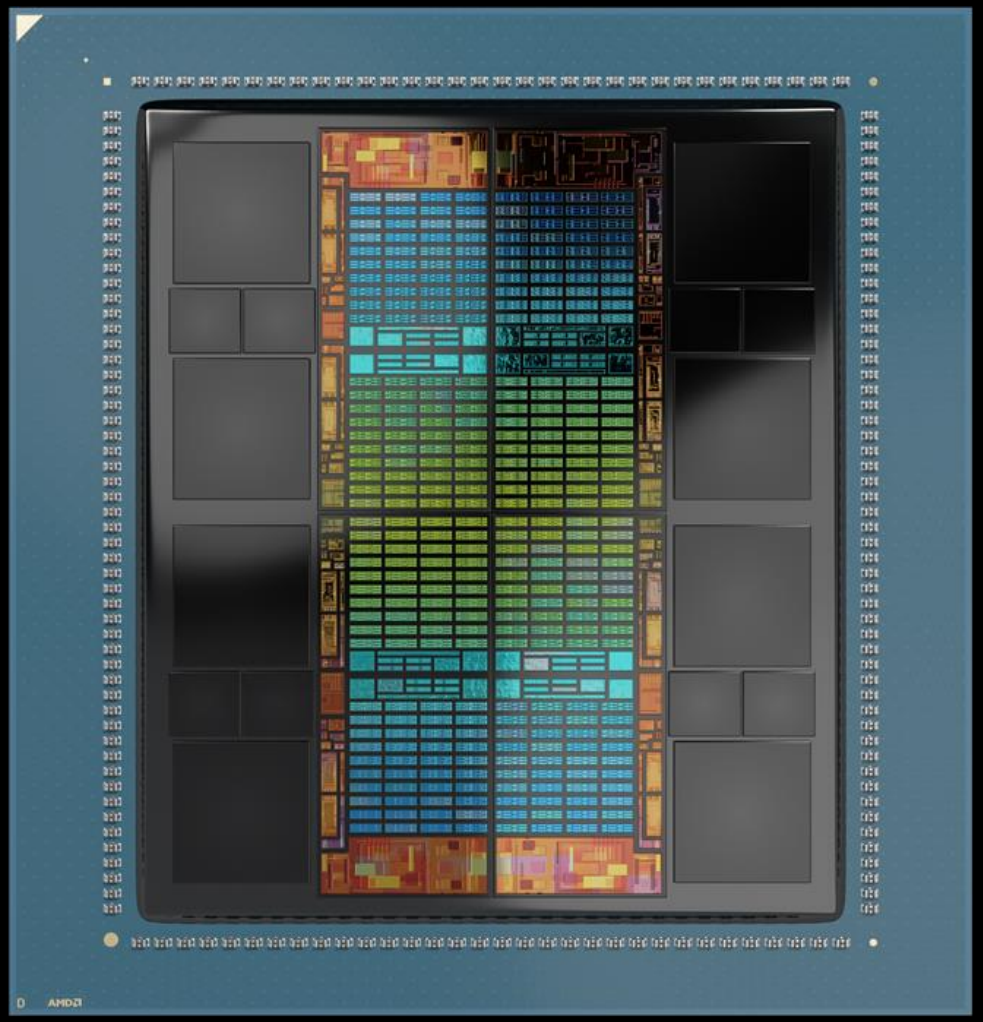
OSS by community
OSS by AMD
Closed-source by AMD



AOMP: AMD OpenMP (Research Compiler) AOCC: AMD Optimizing C/C++ and Fortran Compilers
ROCmCC: AMD ROCm™ Compilers

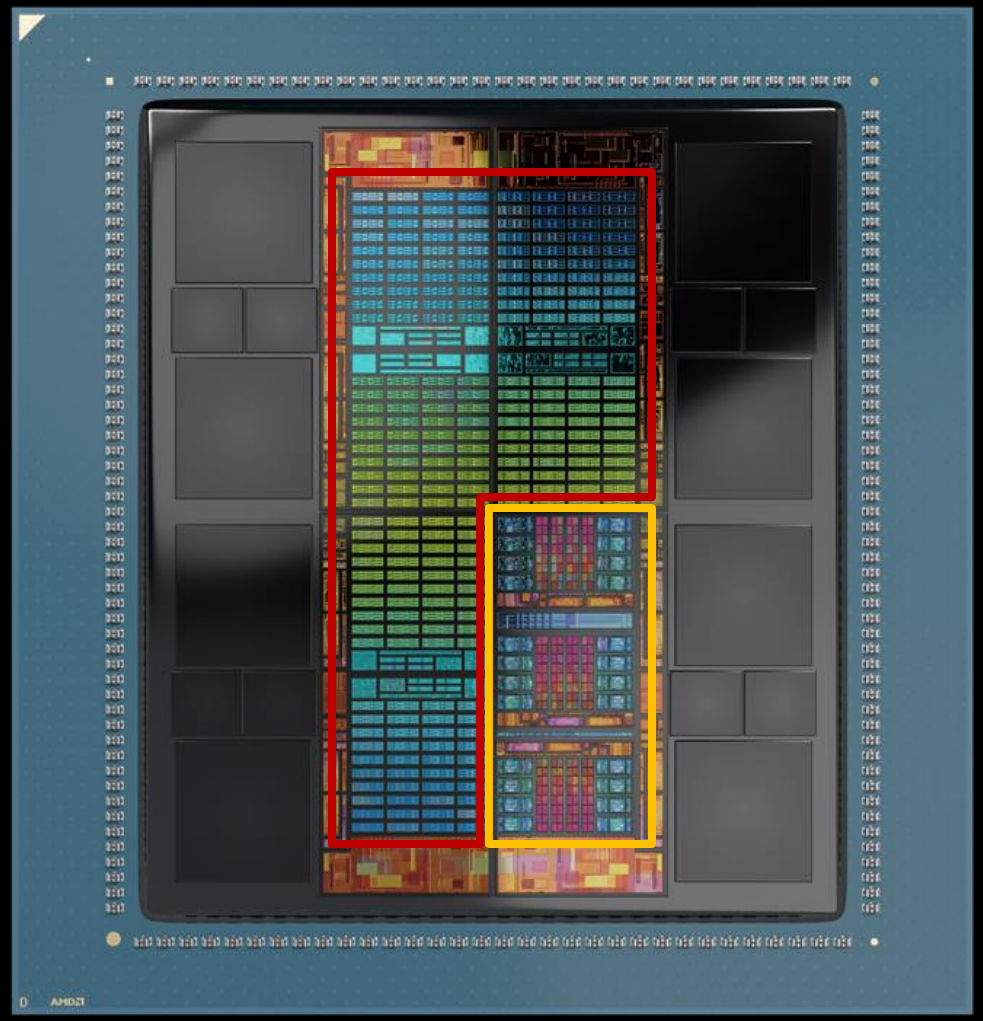
AMD Instinct™ Accelerators

AMD CDNA™ 3 Architecture – AMD Instinct™ MI300X



- Discrete GPU
- 304 AMD CDNA™ 3 compute units
- 192 GB HBM3
- Discrete GPU architecture, OAM
  - AMD EPYC™ Processor as the host system
  - Use host system for memory capacity beyond GPU HBM capacity
  - Connects via
    - To host via PCIe® Gen 5
    - To other GPUs via AMD Infinity Fabric™ Links
  - Supports unified shared memory via page migration (PCIe)

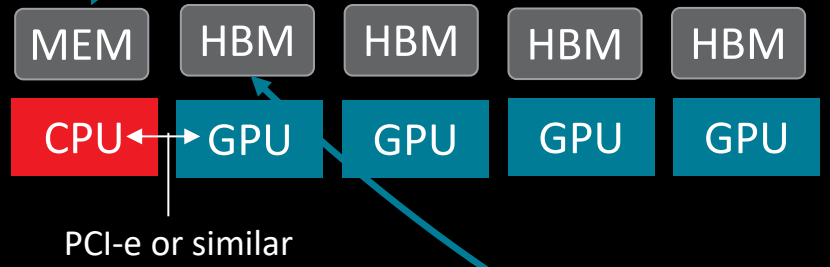
AMD CDNA™ 3 Architecture – AMD Instinct™ MI300A



- APU with unified shared memory
- 24 cores, “Zen 4” architecture
- *228* AMD CDNA™ 3 compute units
- 128 GB HBM3
- Unified memory architecture
  - No discrete CPU and GPU memory
  - CPU and GPU access same physical memory

USM via Zero Copy

OpenMP® Target Offload on Discrete GPUs (Default Mode)



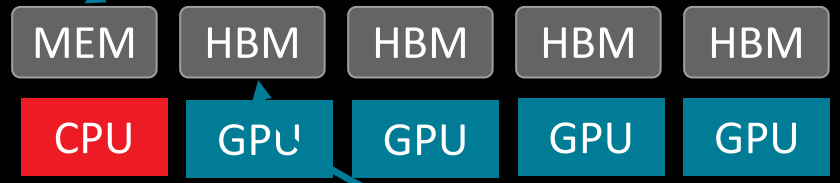
OS allocators allocate host memory

```
program offload
  ! Declarations omitted
  allocate(vals(1024))
  !$omp target loop map(tofrom:vals)
    do i = 1, 1024
      vals = 1
    end do
end program offload
```

- Allocate “d_vals” 1024 integers size on GPU HBM
- Before kernel launch, (DMA) copy vals into “d_vals”
- After kernel completion, (DMA) copy “d_vals” back to vals

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

OpenMP® Target Offload on Discrete GPUs (MI300X, USM Mode)



OS allocators allocate host memory

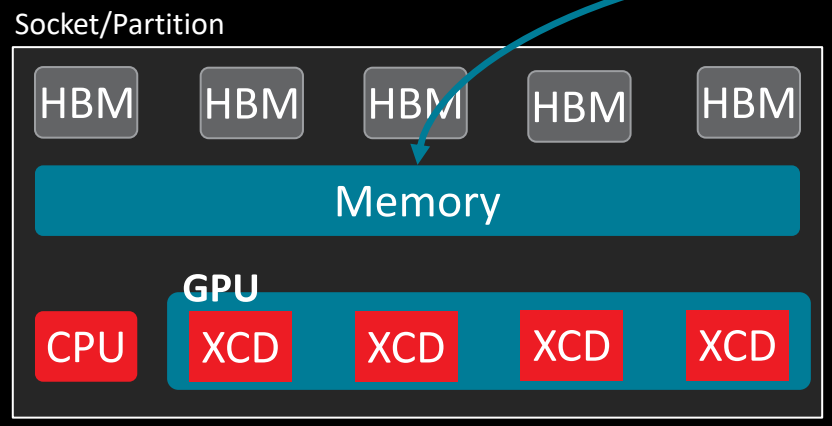
```
program offload
  !$omp requires unified_shared_memory
  ! Declarations omitted
  allocate(vals(1024))
  !$omp target loop map(tofrom:vals)
    do i = 1, 1024
      vals = 1
    end do
end program offload
```

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.

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



OS allocators allocate unified memory*

```
program offload
  !$omp requires unified_shared_
  ! Declarations omitted
  allocate(vals(1024))
  !$omp target loop map(tofrom:vals)
    do i = 1, 1024
      vals = 1
    end do
end program offload
```

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

Side-to-side Compiler Code

```
#include <omp.h>



int main() {
    int* data = (int*) malloc(sizeof(int) * 10);

    #pragma omp target teams loop map(tofrom:data[:10])
    for(int i = 0; i < 10; i++) {
        vals[i] += 1;
    }

    // Print the updated array
    return 0;
}
```

```
#include <omp.h>

#pragma omp requires unified_shared_memory

int main() {
    int* vals = (int*) malloc(sizeof(int) * 10);

    #pragma omp target teams loop
    for(int i = 0; i < 10; i++) {
        vals[i] += 1;
    }

    // Print the updated array
    return 0;
}
```

Side-to-side Compiler Code – Host Binary

without USM

```
struct.ident_t = type { i32, i32, i32, i32, ptr }
%struct.__tgt_offload_entry = type { ptr, ptr, i64, i32, i32 }
%struct.__tgt_kernel_arguments = type { i32, i32, ptr, ptr, ptr, ptr, ptr, ptr, i64, i64, [3 x i32], [3 x i32], i32 }

@0 = private unnamed_addr constant [23 x i8] c";unknown;unknown;0;0;;\00", align 1
@1 = private unnamed_addr constant %struct.ident_t { i32 0, i32 2050, i32 0, i32 22, ptr @0 }, align 8
@2 = private unnamed_addr constant %struct.ident_t { i32 0, i32 514, i32 0, i32 22, ptr @0 }, align 8
@3 = private unnamed_addr constant %struct.ident_t { i32 0, i32 2, i32 0, i32 22, ptr @0 }, align 8
@.__omp_offloading_811_d56211f_main_l16.region_id = weak constant i8 0
@.offload_sizes = private unnamed_addr constant [1 x i64] [i64 40]
@.offload_maptypes = private unnamed_addr constant [1 x i64] [i64 35]
@.offloading.entry_name = internal unnamed_addr constant [38 x i8] c"__omp_offloading_811_d56211f_main_l16\00"
@.offloading.entry.__omp_offloading_811_d56211f_main_l16 = weak constant %struct.__tgt_offload_entry { ptr @.__omp_offloading_811_d56211f_main_l16.region_id,[…]
```

with USM

```
%struct.ident_t = type { i32, i32, i32, i32, ptr }
%struct.__tgt_offload_entry = type { ptr, ptr, i64, i32, i32 }
%struct.__tgt_kernel_arguments = type { i32, i32, ptr, ptr, ptr, ptr, ptr, ptr, i64, i64, [3 x i32], [3 x i32], i32 }

@0 = private unnamed_addr constant [23 x i8] c";unknown;unknown;0;0;;\00", align 1
@1 = private unnamed_addr constant %struct.ident_t { i32 0, i32 2050, i32 0, i32 22, ptr @0 }, align 8
@2 = private unnamed_addr constant %struct.ident_t { i32 0, i32 514, i32 0, i32 22, ptr @0 }, align 8
@3 = private unnamed_addr constant %struct.ident_t { i32 0, i32 2, i32 0, i32 22, ptr @0 }, align 8
@.__omp_offloading_811_d56211f_main_l16.region_id = weak constant i8 0
@.offload_sizes = private unnamed_addr constant [1 x i64] [i64 40]
@.offload_maptypes = private unnamed_addr constant [1 x i64] [i64 35]
@.offloading.entry_name = internal unnamed_addr constant [38 x i8] c"__omp_offloading_811_d56211f_main_l16\00"
@.offloading.entry.__omp_offloading_811_d56211f_main_l16 = weak constant %struct.__tgt_offload_entry { ptr @.__omp_offloading_811_d56211f_main_l16.region_id,[…]
@.offloading.entry_name.1 = internal unnamed_addr constant [1 x i8] zeroinitializer
@.offloading.entry. = weak constant %struct.__tgt_offload_entry { ptr null, ptr @.offloading.entry_name.1, i64 0, i32 16, i32 8 }, section "omp_offloading_entries", align 1
```

Side-to-side Compiler Code – Target Binary

Compiled with #pragma omp requires unified_shared_memory

```
; Function Attrs: alwaysinline mustprogress norecurse nounwind
define weak_odr protected amdgpu_kernel void @__omp_offloading_811_d56211f_main_l16(
ptr noalias noundef %dyn_ptr,
ptr noundef %vals) local_unnamed_addr #0 {
entry:
  %vals.global1 = addrspacecast ptr %vals to ptr addrspace(1)
  %0 = tail call i32 @__kmpc_get_hardware_thread_id_in_block() #1
  %nvptx_num_threads = tail call i32 @__kmpc_get_hardware_num_threads_in_block()
  %gpu_block_id = tail call i32 @llvm.amdgcn.workgroup.id.x()
  %1 = mul i32 %nvptx_num_threads, %gpu_block_id
  %2 = add i32 %1, %0
  %cmp6 = icmp slt i32 %2, 10
  br i1 %cmp6, label %for.body, label %

for.cond.cleanup:                                 ;
  ret void

for.body:                                         ; preds
  %.omp.iv.07 = phi i32 [ %6, %for.body ],
  %idxprom = sext i32 %.omp.iv.07 to i64
  %arrayidx = getelementptr inbounds i32, p
  %3 = load i32, ptr addrspace(1) %arrayidx,
  %add1 = add nsw i32 %3, 1
  store i32 %add1, ptr addrspace(1) %arrayidx
  %nvptx_num_threads2 = tail call i32 @__kmpc_get_hardware_num_threads_in_block() #1
  %4 = tail call i32 @__kmpc_get_hardware_num_blocks() #1
  %5 = mul i32 %4, %nvptx_num_threads2
  %6 = add i32 %5, %.omp.iv.07
  %cmp = icmp slt i32 %6, 10
  br i1 %cmp, label %for.body, label %for.cond.cleanup, !llvm.loop !13
```

Diff between the two LLVM IR snippets:

Compiled without #pragma omp requires unified_shared_memory

```
; Function Attrs: alwaysinline mustprogress norecurse nounwind
define weak_odr protected amdgpu_kernel void @__omp_offloading_811_d56211f_main_l16(
ptr noalias noundef %dyn_ptr,
ptr noundef %vals) local_unnamed_addr #0 {
entry:
  %vals.global1 = addrspacecast ptr %vals to ptr addrspace(1)
  %0 = tail call i32 @__kmpc_get_hardware_thread_id_in_block() #1
  %nvptx_num_threads = tail call i32 @__kmpc_get_hardware_num_threads_in_block() #1
  %gpu_block_id = tail call i32 @llvm.amdgcn.workgroup.id.x()
  %1 = mul i32 %nvptx_num_threads, %gpu_block_id
  %2 = add i32 %1, %0
  %cmp6 = icmp slt i32 %2, 10
  br i1 %cmp6, label %for.body, label %entry

for.body:                                         ; preds = %entry, %for.body
  %.omp.iv.07 = phi i32 [ %6, %for.body ], [ %2, %entry ]
  %idxprom = sext i32 %.omp.iv.07 to i64
  %arrayidx = getelementptr inbounds i32, ptr addrspace(1) %vals.global1, i64 %idxprom
  %3 = load i32, ptr addrspace(1) %arrayidx, align 4, !tbaa !9
  %add1 = add nsw i32 %3, 1
  store i32 %add1, ptr addrspace(1) %arrayidx, align 4, !tbaa !9
  %nvptx_num_threads2 = tail call i32 @__kmpc_get_hardware_num_threads_in_block() #1
  %4 = tail call i32 @__kmpc_get_hardware_num_blocks() #1
  %5 = mul i32 %4, %nvptx_num_threads2
  %6 = add i32 %5, %.omp.iv.07
  %cmp = icmp slt i32 %6, 10
  br i1 %cmp, label %for.body, label %for.cond.cleanup, !llvm.loop !13
```

Difference in Runtime Behavior

… without using zero copy

```
omptarget --> Looking up mapping(HstPtrBegin=0x000000000238b850, Size=40)...
TARGET AMDGPU RTL --> MemoryManagerTy::allocate: size 40 with host pointer 0x000000000238b850.
TARGET AMDGPU RTL --> findBucket: Size 40 is floored to 32.
TARGET AMDGPU RTL --> Cannot find a node in the FreeLists. Allocate on device.
TARGET AMDGPU RTL --> Node address 0x00000000023af530, target pointer 0x00007fe5a6220000, size 40
omptarget --> Creating new map entry with HstPtrBase=0x000000000238b850, HstPtrBegin=0x000000000238b850, TgtAllocBegin=0x00007fe5a6220000,
TgtPtrBegin=0x00007fe5a6220000, Size=40, DynRefCount=1, HoldRefCount=0, Name=unknown
omptarget --> Notifying about new mapping: HstPtr=0x000000000238b850, Size=40
```

Allocate memory on the device

```
omptarget --> Moving 40 bytes (hst:0x000000000238b850) -> (tgt:0x00007fe5a6220000)
omptarget --> There are 40 bytes allocated at target address 0x00007fe5a6220000 - is new
```

Shipping the data to the device

```
omptarget --> Looking up mapping(HstPtrBegin=0x000000000238b850, Size=40)...
omptarget --> Mapping exists with HstPtrBegin=0x000000000238b850, TgtPtrBegin=0x00007fe5a6220000, Size=40, DynRefCount=1 […]
```

Pointer Translation

```
omptarget --> Launching target execution __omp_offloading_811_d562480_main_l15 with pointer 0x0000000002371820 (index=0).
PluginInterface --> Launching kernel __omp_offloading_811_d562480_main_l15 with 2 blocks and 8 threads in SPMD-Big-Jump-Loop mode
```

… using zero copy

```
omptarget --> Looking up mapping(HstPtrBegin=0x0000000000c60850, Size=40)...
omptarget --> Memory pages for HstPtrBegin 0x0000000000c60850 Size=40 switched to coarse grain
omptarget --> Return HstPtrBegin 0x0000000000c60850 Size=40 for unified shared memory
omptarget --> There are 40 bytes allocated at host address 0x0000000000c60850 - is not new
omptarget --> Looking up mapping(HstPtrBegin=0x0000000000c60850, Size=40)...
omptarget --> Get HstPtrBegin 0x0000000000c60850 Size=40 for unified shared memory
omptarget --> Obtained target argument 0x0000000000c60850 from host pointer 0x0000000000c60850
omptarget --> Launching target execution __omp_offloading_811_d56211f_main_l15 with pointer 0x0000000000c46820 (index=0).
PluginInterface --> Launching kernel __omp_offloading_811_d56211f_main_l15 with 2 blocks and 8 threads in SPMD-Big-Jump-Loop mode
```

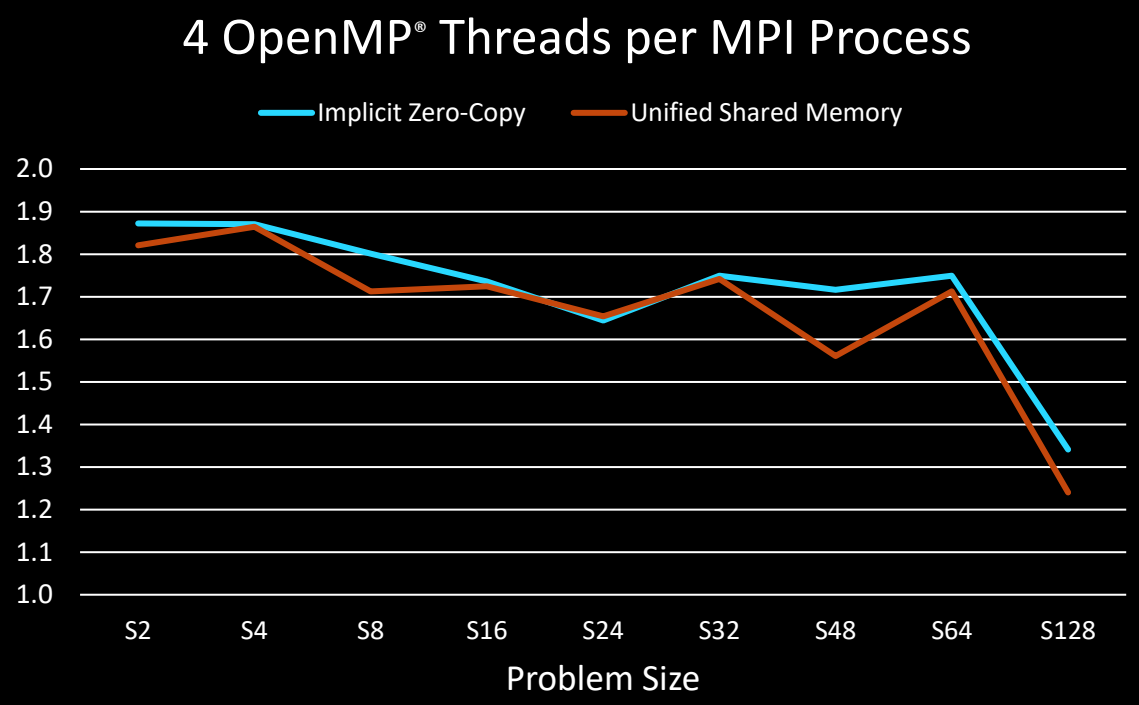
Host-pointer is returned. No memory allocation or copy operation is necessary.

Programming an MI300A ‘APU’ with OpenMP®

		Programming Mode	
Compiler Flags -fopenmp -offload-arch=gfx942		Default non-unified_shared_memory using map clauses	unified_shared_memory #pragma omp requires unified_shared_memory or --fopenmp-force-usm
Runtime State	Unified Memory Enabled HSA_XNACK=1	Zero-copy	Zero-copy
	Unified Memory Disabled HSA_XNACK=0	Copy	Runtime Error

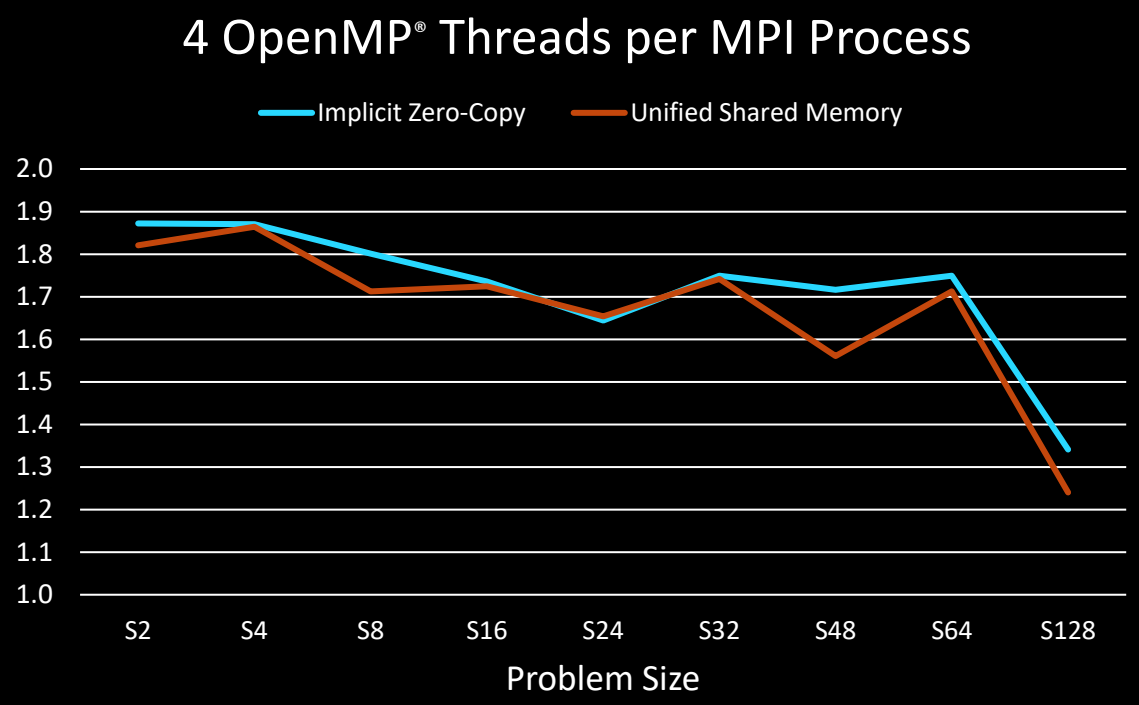
Performance for Quantum Monte Carlo production application called QMCPack
https://qmcpack.org

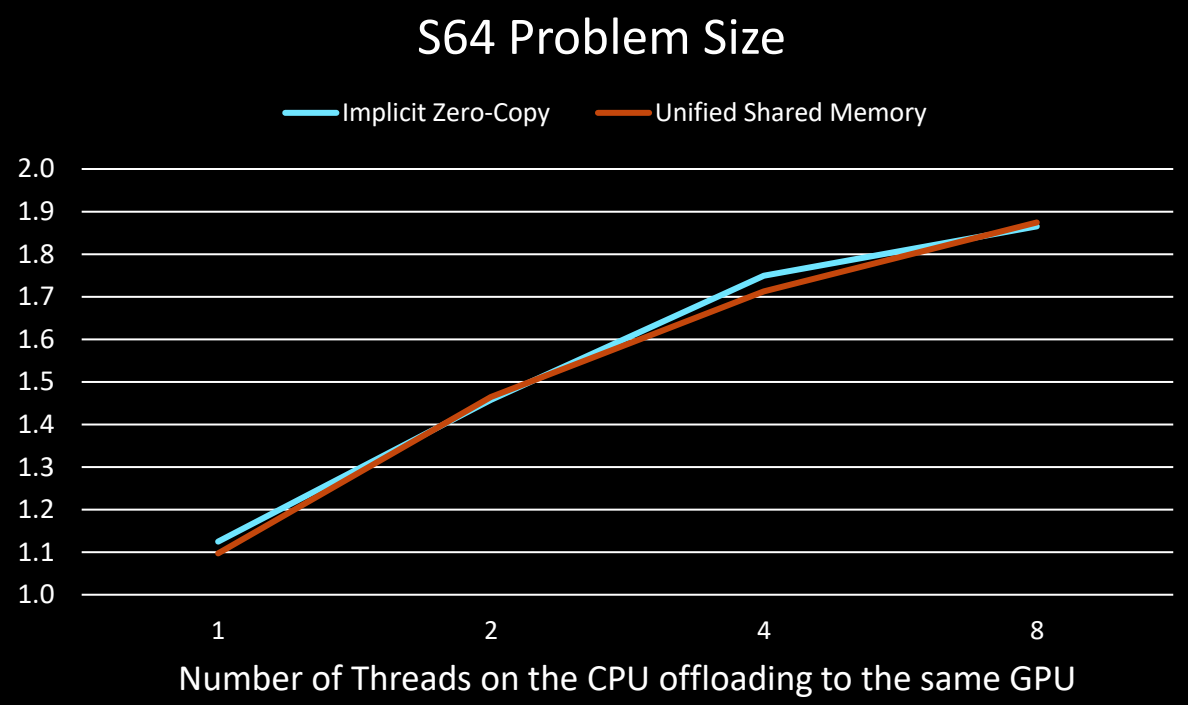
Performance of Zero-Copy for QMCPack



Ratio Copy/Implicit Zero-Copy and USM
(higher means Zero-Copy performs better than Copy)

Performance of Zero-Copy for QMCPack





Ratio Copy/Implicit Zero-Copy and USM
(higher means Zero-Copy performs better than Copy)

Summary

- 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

- 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

*Except for a specific case of global variables

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, the AMD Arrow logo, EPYC, Instinct and combinations thereof are trademarks of Advanced Micro Devices, Inc. PCIe is a registered trademark of PCI-SIG Corporation. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.


AMD