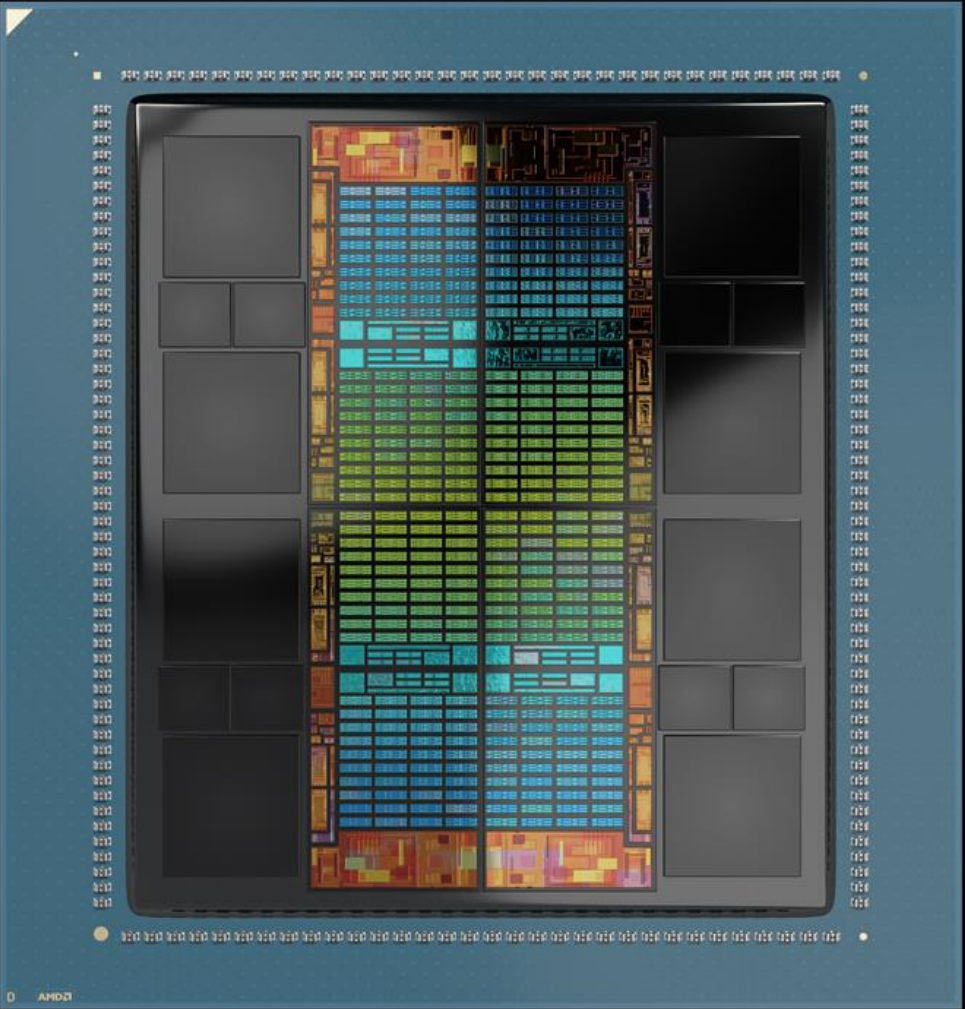




OpenMP® Offload Programming for Discrete GPUs

Presenter: Giacomo Rossi

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
 - to host via PCIe® Gen 5
 - to other GPUs via AMD Infinity Fabric™ Links
- Supports unified shared memory via page migration (PCIe)

OAM: OCP Accelerator Module
OCP: Open Compute Platforms

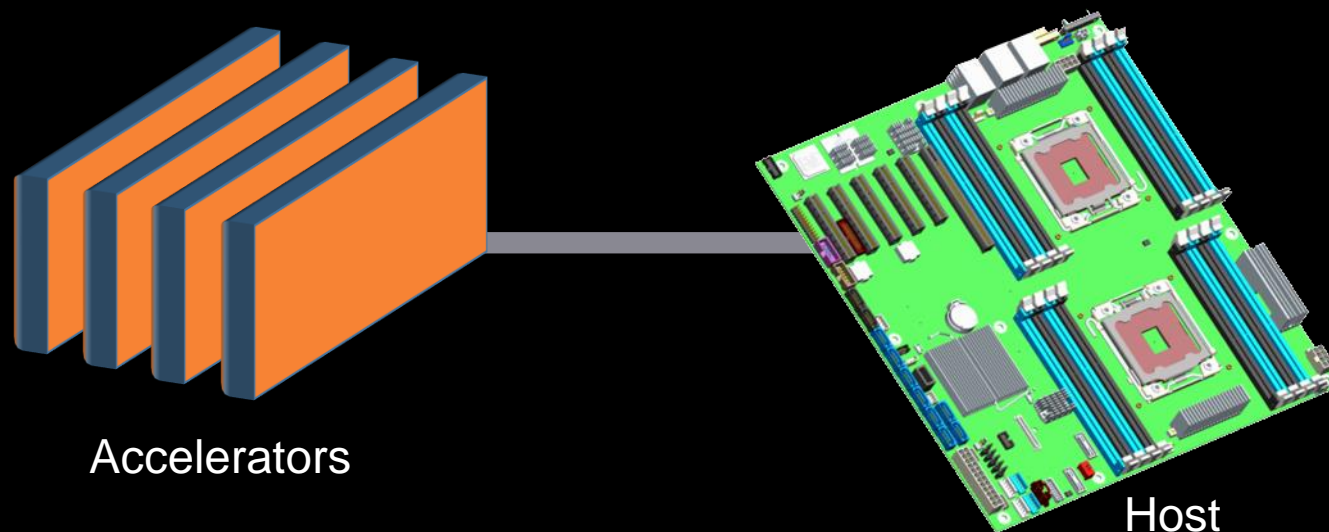
MI300A as discrete GPU

- MI300A APU can execute a program as a discrete GPU setting the environment variable `HSA_XNACK=0`
- This can be useful to ensure that code will run correctly on discrete GPUs
- On the other hand, setting `HSA_XNACK=1` on discrete GPUs will enable the page fault mechanism, with performance penalties

Dealing with Discrete Memory Spaces

OpenMP Device Model (Recap)

- As of version 4.0 the OpenMP API supports accelerators/coprocessors.
- Device model:
 - One host for “traditional” multi-threading
 - Multiple accelerators/coprocessors of the same kind for offloading
 - Devices are accessible through a device ID (from 0 to $n-1$ for n devices)
- OpenMP device model is agnostic of actual technology. In theory, devices only need to
 - be able to receive data from the host and send data back and
 - perform computation upon request.



OpenMP Data Mapping

- Data mapping is one of the OpenMP mechanism for controlling data allocation on device memory environment
- Data must be already available on CPU memory
- No data duplication needed
 - OpenMP runtime creates a device copy of the CPU variable and associates it to the device memory space



Running Example for this Presentation: saxpy

```
int main(int argc, char *argv[]){
    int N=100000;
    float x[N], y[N];

    a, x and y initialized on the CPU

#pragma omp target teams distribute parallel for
    for (int i = 0; i < N; i++) {
        y[i] += a * x[i];
    }
```

```
void saxpy(float a, float* x, float* y, int n) {
#pragma omp target teams distribute parallel for
    for (int i = 0; i < n; i++) {
        y[i] = a * x[i] + y[i];
    }
}
```

Will they run
on discrete GPUs???

Running Example for this Presentation: saxpy

```
int main(int argc, char *argv[]){
    int N=100000;
    float x[N], y[N];

    a, x and y initialized on the CPU

#pragma omp target teams distribute parallel for
    for (int i = 0; i < N; i++) {
        y[i] += a * x[i];
    }
```

It will run!

```
void saxpy(float a, float* x, float* y, int n) {
#pragma omp target teams distribute parallel for
    for (int i = 0; i < n; i++) {
        y[i] = a * x[i] + y[i];
    }
}
```

- OpenMP offload relies on presence check: data is transferred to the device only if is not yet allocated
- If data is not allocated and there's no mapping, OpenMP runtime will apply implicit data mapping attribute rules, as follows:
 - scalars are firstprivate;
 - arrays (statically allocated) are tofrom;
 - reduction variables are tofrom

It won't run!

Running Example for this Presentation: saxpy

```
int main(int argc, char *argv[]){
    int N=100000;
    float x[N], y[N];

    a and x are initialized on the CPU

#pragma omp target teams distribute parallel for
    for (int i = 0; i < N; i++) {
        y[i] += a * x[i];
    }
```

- a is firstprivate, so is allocated automatically on the device memory and its value is copied from the host
- x and y are mapped tofrom, so are allocated automatically on the device memory, their values are copied from host to device on target region entry and copied back from device to host on target region exit

Relying on implicit data mapping is sub-optimal:

- x is not changed inside the target region – no need to copy its value back
- y is initialized inside the target region – no need to copy its value on entry

Optimizing Data Transfers

map Clause

- Specifies how a list item is mapped from the host memory to a corresponding list item in the device data environment
- It applies to the following directives: target, target data, target enter data, target exit data

- Syntax

```
map([map_modifier,][{alloc | to | from | tofrom | release | delete}:]  
list)
```

target Construct Syntax

- Transfer control (and data) from the host to the device

- Syntax (C/C++)

```
#pragma omp target [clause[[,] clause],...]  
structured-block
```

- Syntax (Fortran)

```
!$omp target [clause[[,] clause],...]  
structured-block  
!$omp end target
```

- Clauses

```
device(scalar-integer-expression)  
map([{alloc | to | from | tofrom | release | delete}:] list)  
if(scalar-expr)
```

target Construct Syntax with map clause

```
void saxpy(float a, float* x, float* y, int n) {
#pragma omp target teams distribute parallel for map(to:x[0:N]) map(from:y[0:N])
for (int i = 0; i < n; i++) {
    y[i] = a * x[i];
}
}
```

```
subroutine saxpy(a, x, y, n)
    ! Declarations omitted
!$omp omp target teams loop map(to:x) map(from:y)
do i=1,n
    y(i) = a * x(i)
end do
!$omp end target teams loop
end subroutine
```

At target region entry, if not already present in the device memory space:

- x and y are allocated on the device
- x data is copied from host to device

At target region exit, if not already present in the device memory space:

- x is deallocated from device
- y data is copied from device to host and then y is deallocated from device

If data is already on the device, no data transfer occurs!!!

target data Construct Syntax

- Create scoped data environment and transfer data from the host to the device and back
- Syntax (C/C++)

```
#pragma omp target data [clause[[, clause],...]  
structured-block
```
- Syntax (Fortran)

```
!$omp target data [clause[[, clause],...]  
structured-block  
!$omp end target data
```
- Clauses

```
device(scalar-integer-expression)  
map([{alloc | to | from | tofrom | release | delete}:] list)  
if(scalar-expr)
```

target data Construct Syntax (continued)

```
!$omp target data map(alloc:tmp) &
!$omp& map(to:input)
```

host

```
!$omp target teams distribute parallel do simd
  do i=1, N
    tmp(i) = some_computation(input(i), i)
  enddo
```

target

```
  update_input_array_on_the_host(input);
```

host

```
!$omp target teams parallel do simd reduction(+:res) &
!$omp& map(always,to:input)
  do i=1, N
    res = res + final_computation(input(i), tmp(i), i)
  enddo
```

target

```
!$omp end target data
```

host

At target data region entry:

- tmp and input are allocated on the device
- input data is copied from host to device

At first target region, presence check finds data on the device

At second target region, input data must be updated on the device, but the presence check recognizes that data is already on the device:

- always modifier force data transfer in case of data already allocated on the device
- res is object of a sum reduction, so it's automatically initialized to zero in the device memory space and its final value will be copied back on the host at target region exit

At the end of target data region:

- tmp and input are deallocated from device

target enter/exit data Construct Syntax

- Create unscoped data environment and transfer data from the host to the device and back
- Syntax (C/C++)
`#pragma omp target [enter | exit] data [clause[[,] clause],...]`
- Syntax (Fortran)
`!$omp target [enter | exit] data [clause[[,] clause],...]`
- Clauses
`device(scalar-integer-expression)`
`map([{alloc | to}:] list) - target enter data`
`map([{from | release | delete}:] list) - target exit data`
`if(scalar-expr)`

target enter/exit data Constructs Syntax (continued)

```
!$omp target enter data map(alloc:tmp) map(to:input)
```

```
!$omp target teams distribute parallel do
```

```
do i = 1, N
```

```
    tmp(i) = some_computation(input(i), i)
```

```
end do
```

```
call update_input_array_on_the_host(input)
```

```
!$omp target teams distribute parallel do reduction(+:res) &
```

```
!$omp& map(always,to:input)
```

```
do i = 1, N
```

```
    res = res + final_computation(input(i), tmp(i), i)
```

```
    input(i) = input(i) + tmp(i)
```

```
end do
```

```
!$omp target exit data map(delete:tmp) map(from:input)
```

host

target

host

target

host

At target enter data:

- tmp and input are allocated on the device
- input data is copied from host to device

At first target region, presence check finds data on the device

At second target region, input data must be updated on the device, but the presence check recognizes that data is already on the device:

- always modifier force data transfer in case of data already allocated on the device
- res is object of a sum reduction, so it's automatically initialized to zero in the device memory space and its final value will be copied back on the host at target region exit

At target exit data:

- input data is copied back on the host
- tmp and input are deallocated from device

Map types

	Not Present			Present		
	Allocation	Deallocation	Data transfer	Allocation	Deallocation	Data transfer
alloc	YES	NO ¹	NO	Check size ⁵	NO ⁶	NO ⁷
to	YES	NO ¹	H->D ²	Check size ⁵	NO ⁶	NO ⁷
from	YES	NO ¹	D->H ³	Check size ⁵	NO ⁶	NO ⁷
tofrom	YES	NO ¹	H->D, D->H ⁴	Check size ⁵	NO ⁶	NO ⁷
release						
delete	NO	YES	NO	NO	YES	NO

1. For scoped data regions (target, target data) data is deallocated from the device memory space if it wasn't present at scoped data region entry
2. For scoped data regions data is transferred from host to device at data region entry
3. For scoped data regions data is transferred from device to host at data region exit
4. For scoped data regions, data is transferred from host to device at data region entry and from device to host at data region exit
5. For scoped data regions, reference counter is incremented
6. For scoped data regions, reference counter is decremented
7. This behavior can be modified using the always map type modifier

target update Construct Syntax

- Issue data transfers to or from existing data device environment

- Syntax (C/C++)

```
#pragma omp target update [clause[[,] clause],...]
```

- Syntax (Fortran)

```
!$omp target update [clause[[,] clause],...]
```

- Clauses

```
device(scalar-integer-expression)
```

```
to(list)
```

```
from(list)
```

```
if(scalar-expr)
```

target update Construct Syntax (continued)

```

!$omp target enter data map(alloc:tmp) map(to:input)
!$omp target teams distribute parallel do
  do i = 1, N
    tmp(i) = some_computation(input(i), i)
  end do

  call update_input_array_on_the_host(input)

!$omp target update to(input)

!$omp target teams distribute parallel do reduction(+:res)
  do i = 1, N
    res = res + final_computation(input(i), tmp(i), i)
    input(i) = input(i) + tmp(i)
  end do

!$omp target update from(input)

!$omp target exit data map(delete:tmp,input)

```

host

target

host

target

host

input is already in the device memory space, so only the data needs to be copied on the device and later on the host

Comparing USM vs. Discrete Memory

Plain 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];
```

for Discrete GPU

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

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

#pragma omp target data \
    map(to:in[0:Msize]) \
    map(from:out[0:Msize])
{
    #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];
```

for Unified 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
```

AMD Compiler Behavior

Compiler Flag: gfx942	Default (non-unified_shared_memory)			unified_shared_memory		
	xnack-	xnack-any	xnack+	xnack-	xnack-any	xnack+
XNACK-Enabled	Mismatch	Zero-copy	Zero-copy	Mismatch	Zero-copy	Zero-copy
XNACK-Disabled	Copy	Copy	Mismatch	Zero-copy (RT-Warn)	Zero-copy (RT-Warn)	Mismatch

Mismatch: code object requirement does not match available hardware capability (it's as if you built for a different GPU target)

OpenMP Miscellaneous Topics

OpenMP Resources

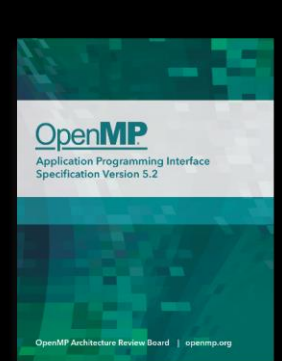
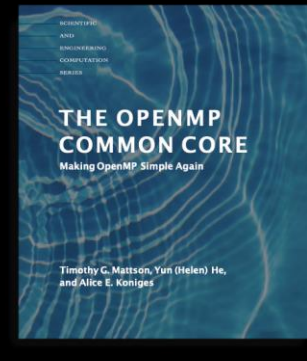
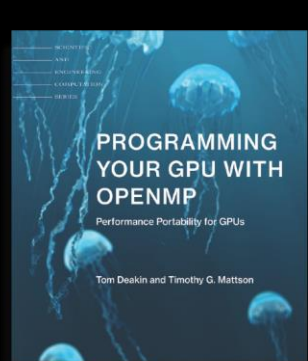


Reference guides

<https://www.openmp.org>

YouTube Channel

OpenMP Books



Summary

- OpenMP API is ready to use AMD discrete GPUs for offloading compute
 - Mature offload model w/ support for asynchronous offload/transfer
 - Tightly integrates with OpenMP multi-threading on the host
 - Better to avoid relying on implicit data mapping attributes
 - To get performance, explicit data management on discrete GPUs is mandatory
- More, advanced features (not covered here)
 - Memory management API
 - Interoperability with native data management
 - Interoperability with native streaming interfaces

Acknowledgements

- Michael Klemm
- Johanna Potyka

- More colleagues from HPC Solutions and Performance Analysis group in AMD

- Participants and Mentors of the past HLRS Trainings, Hunter preparation Hackathons and other Hackathons 😊

Hands-on / Exercises

Exercise: Porting with map clauses

- C/C++:

https://github.com/amd/HPCTrainingExamples/tree/main/Pragma_Examples/OpenMP/C/README.md

- Fortran:

https://github.com/amd/HPCTrainingExamples/tree/main/Pragma_Examples/OpenMP/Fortran/README.md

What to do:

- 1) Access aac6.
- 2) Setup your environment (See README in the top C and Fortran exercise folders, recommendation: view in browser)
- 3) Each sub-folder 1-6 has instructions for a unified shared memory version (last lecture) and one with map clauses (this lecture).
- 4) Work on porting a version with map clauses of exercises 1-6. Make sure you set HSA_XNACK=0 that map clauses are not ignored.

For Fortran: Note that the asynchronous offload version which is part of 3_vecadd does only work with ftn and the device routine exercise (with nohost) only works with amdflang-new.

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.

© 2025 Advanced Micro Devices, Inc and OpenMP® Architecture Review Board. All rights reserved.

AMD, the AMD Arrow logo, 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.

AMD 