Accelerating HPC Applications on AMD Instinct™ GPUs with OpenMP® offloading: An Overview

Suyash Tandon, Member of Technical Staff

May 2023
CAUTIONARY STATEMENT

This presentation contains forward-looking statements concerning Advanced Micro Devices, Inc. (AMD) such as the features, functionality, performance, availability, timing and expected benefits of AMD’s current products, future products and markets, which are made pursuant to the Safe Harbor provisions of the Private Securities Litigation Reform Act of 1995. Forward-looking statements are commonly identified by words such as "would," "may," "expects," "believes," "plans," "intends," "projects" and other terms with similar meaning. Investors are cautioned that the forward-looking statements in this presentation are based on current beliefs, assumptions and expectations, speak only as of the date of this presentation and involve risks and uncertainties that could cause actual results to differ materially from current expectations. Such statements are subject to certain known and unknown risks and uncertainties, many of which are difficult to predict and generally beyond AMD’s control, that could cause actual results and other future events to differ materially from those expressed in, or implied or projected by, the forward-looking information and statements. Investors are urged to review in detail the risks and uncertainties in AMD’s Securities and Exchange Commission filings, including but not limited to AMD’s most recent reports on Forms 10-K and 10-Q.

AMD does not assume, and hereby disclaims, any obligation to update forward-looking statements made in this presentation, except as may be required by law.
Agenda

1. Introduction to MI 200 hardware
2. Software stack and tools
3. Basics of OpenMP® offloading
4. HIP & OpenMP® - compatibility
5. Case studies
6. Heterogenous memory management (HMM)
COMPUTE GPU ARCHITECTURE ROADMAP

2H’20
CDNA
AMD INSTINCT MI 100
7nm

2H21
CDNA 2
AMD INSTINCT MI 200 SERIES
6nm

5nm
CDNA 3
AMD INSTINCT MI 300

2020
2024

Roadmaps Subject to Change
2ND GENERATION CDNA ARCHITECTURE
TAILORED-BUILT FOR HPC & AI

- TSMC 6NM TECHNOLOGY
- UP TO 110 CU PER GRAPHICS CORE DIE
- 4 MATRIX CORES PER COMPUTE UNIT
- MATRIX CORES ENHANCED FOR HPC
- 8 INFINITY FABRIC LINKS PER DIE
- SPECIAL FP32 OPS FOR DOUBLE THROUGHPUT
MULTI-CHIP DESIGN
TWO GPU DIES IN PACKAGE TO MAXIMIZE COMPUTE & DATA THROUGHPUT

INFINITY FABRIC FOR CROSS-DIE CONNECTIVITY

4 LINKS RUNNING AT 25GBPS

400GB/S OF BI-DIRECTIONAL BANDWIDTH
From AMD MI100 to AMD MI210

**MI 100**
- 32GB of HBM2 memory
- 11.5 TFLOPS peak performance
- 1.2 TB/s peak memory bandwidth
- 120 CU


**MI 210**
- 64GB of HBM2e memory
- 26.5 TFLOPS peak performance
- 1.6 TB/s peak memory bandwidth
- 108 CU
- 128 single precision FMA operations per cycle
- AMD CDNA 2 Matrix Core supports double-precision data
## SCIENTISTS TARGET APPLICATIONS FOR WIDE RANGE OF SYSTEMS

### Pre-Exascale Systems [Aggregate Linpack (Rmax) = 323 PF!]

<table>
<thead>
<tr>
<th>Year</th>
<th>System</th>
<th>Vendor/Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>Titan (9)</td>
<td>ORNL Cray/AMD/NVIDIA</td>
</tr>
<tr>
<td>2016</td>
<td>Sequoia (10)</td>
<td>ANL IBM BG/Q</td>
</tr>
<tr>
<td>2018</td>
<td>Trinity (6)</td>
<td>LANL/SNL Cray/Intel Xeon®/ KNL</td>
</tr>
<tr>
<td>2020</td>
<td>Sierra (2)</td>
<td>LLNL IBM/NVIDIA</td>
</tr>
</tbody>
</table>

### First U.S. Exascale Systems

<table>
<thead>
<tr>
<th>Year</th>
<th>System</th>
<th>Vendor/Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-2023</td>
<td>Summit (1)</td>
<td>ORNL Cray/Intel Xeon®/ KNL</td>
</tr>
<tr>
<td></td>
<td>ORNL</td>
<td>IBM/NVIDIA</td>
</tr>
<tr>
<td></td>
<td>LBNL</td>
<td>Cray/AMD/NVIDIA</td>
</tr>
<tr>
<td></td>
<td>ANL</td>
<td>Cray/Intel KNL</td>
</tr>
<tr>
<td></td>
<td>LLNL</td>
<td>IBM/AMD/Intel</td>
</tr>
<tr>
<td></td>
<td>LANL/SNL</td>
<td>HPE/Intel</td>
</tr>
<tr>
<td></td>
<td>LLNL</td>
<td>HPE/AMD/AMD</td>
</tr>
</tbody>
</table>

IDEAL APPLICATION DEVELOPMENT FROM THE SCIENTIST’S PERSPECTIVE

Performant
- Efficient use of hardware resources for energy consumed
- Scale from single to multi-node

Portable
- Support both CPUs and GPUs
- Execute application on various platform architectures

Productive
- Optimize time to solution for new research
- Abstract the computer science (code, data movement, scaling, etc)
PERFORMANCE VS PORTABILITY TRADEOFF

Portability drops as software is tuned for specific HW features.
GPU PROGRAMMING IS DIFFICULT – BUT EASIER IF HW IS ABSTRACTED

- Domain Specific Language
  - GridTools, Devito

- Frameworks
  - Kokkos, Legion, TF, PyTorch

- Libraries
  - Math, Communication

- ISO Standard Languages
  - C++, Fortran, Python

- Directive Based Extensions
  - OpenMP, OpenACC

- Accelerator Languages
  - OpenCL, HIP, CUDA, SYCL

- HW Specific
  - GCN-ISA, PTX, oneAPI-L0
Agenda

1. Introduction to MI 200 hardware
2. Software stack and tools
3. Basics of OpenMP® offloading
4. HIP & OpenMP® - compatibility
5. Case studies
6. Heterogeneous memory management (HMM)
# Open Software Platform For GPU Compute

**AMD ROCm**

- Unlocked GPU Power To Accelerate Computational Tasks
- Optimized for HPC and Deep Learning Workloads at Scale
- Open Source Enabling Innovation, Differentiation, and Collaboration

## Key Features

<table>
<thead>
<tr>
<th>Benchmark &amp; App Support</th>
<th>Optimized Training/Inference Models &amp; Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Systems Support</td>
<td>MLPERF, HPL/HPCG, Life Science, Geo Science, Physics</td>
</tr>
<tr>
<td>Cluster Deployment</td>
<td>RHEL, CentOS, SLES, Ubuntu</td>
</tr>
<tr>
<td>Framework Support</td>
<td>Singularity, Kubernetes*, Docker*, SLURM</td>
</tr>
<tr>
<td>Libraries</td>
<td>Kokkos/RAJA, PyTorch, TensorFlow</td>
</tr>
<tr>
<td>Programming Models</td>
<td>BLAS, RAND, FFT, MiGraphX, MiVisionX, PRIM</td>
</tr>
<tr>
<td>Development Toolchain</td>
<td>SOLVER, ALUTION, SPARSE, THRUST, MiOpen, RCCL</td>
</tr>
<tr>
<td>Drivers &amp; Runtime</td>
<td>OpenMP®, OpenCL™, HIP API</td>
</tr>
<tr>
<td>Deployment Tools</td>
<td>Compiler, Profiler, Tracer, Debugger, hipify, GPUFort</td>
</tr>
</tbody>
</table>

## Additional Tools
- GPU Device Drivers and ROCm Run-Time
- ROCm Validation Suite, ROCm Data Center Tool, ROCm SMI
## Compiler with OpenMP® support on AMD GPUs

### AOMP
- **Stack**: Packaged separately from ROCm
  - Prototype new features for ROCmCC
  - Same command line options as ROCmCC
- **Specifications**: Complete 4.5 and partial 5.0 support
- **Enable OpenMP®**: -fopenmp
- **Select GPU Targets** (AOMP-15.0-2 or newer):
  - --offload-arch=<arch-name>
  - Replaces -fopenmp-targets, -Xopenmp-target, and -march options
- **Optimizations**:
  - -O0
  - -O1 & above
  - -famd-opt
  - -ffast-math
- **Debug**: -g

### CRAY
- **Stack**: Packaged with Cray Compiling Environment
- **Specifications**: Complete 4.5 and partial 5.0 support
- **Enable OpenMP®**: -fopenmp or -homp
- **Select GPU Targets**:
  - --offload-arch=<arch-name>
  - Replaces -fopenmp-targets, -Xopenmp-target, and -march options
- **Optimizations**:
  - -O0
  - -O1 & above
  - -hfpN, -ffp=N
- **Debug**: -g

### GNU (OG-12)
- **Stack**: Siemens' free GCC-based compiler
  - Based on GCC-12 branch
  - Support offloading to AMD CDNA™
- **Specifications**: Complete 4.5 and partial 5.0 support
- **Enable OpenMP®**: -fopenmp
- **Select GPU Targets** (GCC-12):
  - --offload-arch=<arch-name>
  - Replaces -fopenmp-targets, -Xopenmp-target, and -march options
- **Optimizations**:
  - -O0
  - -O1 & above
  - -famd-opt
  - -ffast-math
- **Debug**: -g
AMD development tools

**ROC-profiler (rocpprof)**
- Raw collection of GPU counters and traces
- Counter collection with user input files
- Counter results printed to a CSV

**OmniTrace**
- Comprehensive trace collection
  - CPU
  - GPU

**OmniPerf**
- Performance Analysis
  - Automated collection of hardware counters
  - Analysis
  - Visualisation

**Hardwre Counters**
- Trace collection support for
  - CPU copy
  - HIP API
  - HSA API
  - GPU Kernels

**Supports**
- Supports
  - CPU copy
  - HIP API
  - HSA API
  - GPU Kernels
  - OpenMP®
  - MPI
  - Kokkos
  - p-threads
  - multi-GPU

**Traces visualized with Perfetto**

**Visualisation**
- With Grafana or standalone GUI
Agenda

1. Introduction to MI 200 hardware
2. Software stack and tools
3. Basics of OpenMP® offloading
4. HIP & OpenMP® - compatibility
5. Case studies
6. Summary
Basics of OpenMP® offloading

**OMP TARGET**
Defines a target region to be offloaded on device

```fortran
program target_example
  complex :: M,N
  N=2,2
  M=0,0
  !$omp target map(from:M) map(to:N)
  M=N
  !$omp end target
  write(*,*) "M= ", M
end program target_example
```

**PARALLEL**
Defines a parallel region within the code, usually loops

```fortran
!$omp target parallel do map(from:A)
  do i=1, 5000
    A(i) = (2,2)
  enddo
!$omp end target parallel do
```

```fortran
!$omp target teams distribute parallel do map(from:A)
  do i=1, 5000
    A(i) = (2,2)
  enddo
!$omp end target teams distribute parallel do
```
Basics of OpenMP® offloading

**OMP TARGET**
Defines a target region to be offloaded on device

```fortran
program target_example
    complex :: M,N
    N=2,2
    M=(0,0)

    !$omp target map(from:M) map(to:N)
    M=N
    !$omp end target
    write(*,*), "M= ", M
end program target_example
```

**DATA**
Ensures that data is correctly exposed to a device

- **$OMP TARGET ENTER DATA MAP(ALLOC:A):** Map A to the device. Initial value on device is undefined!
- **$OMP TARGET ENTER DATA MAP(TO:A):** Map A to the device. Initialize with value from host.
- **$OMP TARGET EXIT DATA MAP(DELETE:A):** Unmap A from device. Set allocation count to zero.
- **$OMP TARGET EXIT DATA MAP(RELEASE:A):** Unmap A from device. Decrement allocation count by one
- **$OMP TARGET UPDATE(TO:A):** Copy A from host to device
- **$OMP TARGET UPDATE(FROM:A):** Copy A from device to host

**PARALLEL**
Defines a parallel region within the code, usually loops

```fortran
!$omp target parallel do map(from:A)
    do i=1, 500
        A(i) = (2,2)
    enddo
!$omp end target parallel do

!$omp target teams distribute parallel do map(from:A)
    do i=1, 500
        A(i) = (2,2)
    enddo
!$omp end target teams distribute parallel do
```
Common errors

HSA_STATUS_ERROR_MEMORY_FAULT: Agent attempted to access an inaccessible address. code: 0x2b

Data is not present on GPU!

Host region (7ffc4df0dd20 to 7ffc4df1dd20) overlaps present region (7ffc4df19e80 to 7ffc4df22e80 index 42) but is not contained for A in hamil.f90

Data is mapped to device but is not deleted/released!
Debugging with AOMP: LIBOMPTARGET_DEBUG

```fortran
1  program target_example
2     complex :: M,N
3     N=(2,2)
4     M=(0,0)
5     !$omp target map(from:M) map(to:N)
6     M=N
7     !$omp end target
8     write(*,*), "M= ", M
9  end program target_example
```
Debugging with Cray compiler: –hlist=aimd

1. Program test
2. integer :: i
3. complex, pointer :: A(:)
4. allocate(A(500))
5. do i=1, 500
6. A(i) = (0,0)
7. enddo
8. !$omp target teams distribute parallel do simd mapfrom:A
9. do i=1, 500
10. A(i) = (2,2)
11. enddo
12. !$omp end target teams distribute parallel do simd
13. write(*,*) "A(1)= ", A(1)
14. end program test

Compiling with Cray Fortran

$ ftn -hnoacc -homp -fopenmp -hlist=aimd -o ./teamsdis ./teamsdis.f90
Profiling OpenMP® offloading code on AMD GPUs

Basic profiling with rocprof:

Compile:

```
$ftn -hnoacc -fopenmp -homp -o ./test ./test.f90
```

Profile and collect HIP trace:

```
$rocprof -hip-trace ./test
```

Open the .json file in chrome://tracing/ or https://ui.perfetto.dev/
Agenda

1. Introduction to MI 200 hardware
2. Software stack and tools
3. Basics of OpenMP® offloading
4. HIP & OpenMP® - compatibility
5. Case studies
6. Heterogenous memory management (HMM)
HIP & OpenMP® – Hybrid programming: compatible & competitive

Hybrid programming here stands for the interaction of OpenMP with a lower-level programming model like HIP. In other words, one can program with OpenMP in the style one might program with HIP.

OpenMP supports the following interactions:

- Calling low-level HIP kernels from OpenMP application code
- Calling HIP/ROCM math libraries (rocBLAS, rocFFT, etc.) from OpenMP application code
- Calling OpenMP kernels from low-level HIP application code
Basic profiling with rocprof:

```c
void example() {
    float a = 2.0;
    float * x;
    float * y;
    #pragma omp target data map(to:x[0:count]) map(tofrom:y [0:count])
    {
        compute_1(n, x);
        compute_2(n, y);
        #pragma omp target update to(x[0:count]) to(y[0:count])
        saxpy(n, a, x, y)
        compute_3(n, y);
    }
}
```

```c
void saxpy(size_t n, float a, float * x, float * y) {
    #pragma omp target teams distribute parallel for ...
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```
HIP & OpenMP® – HIP kernel for saxpy()

A HIP version of the SAXPY kernel:

```c
__global__ void saxpy_kernel (size_t n, float a, float * x , float * y ){
    size_t i = threadIdx.x + blockIdx.x * blockDim.x;
    y[i] = a * x[i] + y[i];
}

void saxpy_hip (size_t n, float a, float * x , float * y ){
    assert(n % 256 == 0);
    saxpy_kernel <<<n/256,256,0,NULL>>> (n, a, x , y);
}
```

We need a way to translate the host pointer that was mapped by OpenMP directives and retrieve the associated device pointer.
HIP & OpenMP® – Putting it together

__global__ void saxpy_kernel(size_t n, float a, float * x, float * y) {
    size_t i = threadIdx.x + blockIdx.x * blockDim.x;
    y[i] = a * x[i] + y[i];
}

void example() {
    float a = 2.0;
    float * x = ...; // assume: x = 0xabcd
    float * y = ...;
    // allocate the device memory
    #pragma omp target data map(to:x [0:count]) tofrom:y [0:count]
    {
        compute_1(n, x); // mapping table: x: [0xabcd, 0xef12], x = 0xabcd
        compute_2(n, y);
        #pragma omp target update to(x[0:count]) to(y[0:count]) // update x and y on the target
        #pragma omp target data use_device_ptr (x,y)
        {
            saxpy_hip(n, a, x, y) // mapping table: 0xabcd, 0xef12], x = 0xef12
        }
    }
    compute_3(n, y);
}
HIP & OpenMP® – Fortran and DGEMM example

subroutine example
    use roc_rmm_interface
    use iso_c_binding
    implicit none
    real(8), allocatable, target, dimension(:,::) :: a, b, c
    type(c_ptr) :: rochblas_handle
    allocate(da(M,N), db(N,K), dc(M,K))
    call init_matrices(da, db, dc, M, K)
    ! Initialize matrices
    call init_rocblas(rochblas_handle)
    ! Initialize rocBLAS
    !$OMP target enter data map(to:a,b,c)
    !$OMP target data use_device_ptr(a,b,c)
    call omp_dgemm(rochblas_handle, modeA, modeB, M, N, K, alpha, &
        c_loc(a), lda, c_loc(b), ldb, beta, c_loc(c), ldc)
    !$OMP target end data
    !$OMP target update from(c)
    !$OMP target exit data map(delete:a,b,c)
end subroutine example
HIP & OpenMP® – Babelstream case study

Full comparison of OpenMP Offloading vs HIP for all kernels in single precision and double precision
All experiments performed on a single Instinct MI100 using AOMP 13.06
Default Threads * Teams configuration already optimal for some kernels

<table>
<thead>
<tr>
<th>Single Precision</th>
<th>Default Threads * Teams</th>
<th>OpenMP/HIP ratio</th>
<th>Optimal Threads * Teams</th>
<th>Optimal OpenMP/HIP ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>256 * 480</td>
<td>1.48</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Write</td>
<td>256 * 480</td>
<td>1.96</td>
<td>1024 * 1440</td>
<td>2.05</td>
</tr>
<tr>
<td>Copy</td>
<td>256 * 480</td>
<td>0.92</td>
<td>128 * 1920</td>
<td>0.97</td>
</tr>
<tr>
<td>Mul</td>
<td>256 * 480</td>
<td>0.92</td>
<td>128 * 1440</td>
<td>0.97</td>
</tr>
<tr>
<td>Add</td>
<td>256 * 480</td>
<td>0.89</td>
<td>128 * 1680</td>
<td>0.93</td>
</tr>
<tr>
<td>Triad</td>
<td>256 * 480</td>
<td>0.88</td>
<td>128 * 1440</td>
<td>0.92</td>
</tr>
<tr>
<td>Dot</td>
<td>256 * 480</td>
<td>0.57</td>
<td>64 * 1920</td>
<td>0.72</td>
</tr>
<tr>
<td>Double Precision</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>256 * 480</td>
<td>1.01</td>
<td>1024 * 960</td>
<td>1.06</td>
</tr>
<tr>
<td>Write</td>
<td>256 * 480</td>
<td>0.90</td>
<td>1024 * 60</td>
<td>0.95</td>
</tr>
<tr>
<td>Copy</td>
<td>256 * 480</td>
<td>0.93</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mul</td>
<td>256 * 480</td>
<td>0.92</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Add</td>
<td>256 * 480</td>
<td>0.93</td>
<td>64 * 1440</td>
<td>0.94</td>
</tr>
<tr>
<td>Triad</td>
<td>256 * 480</td>
<td>0.92</td>
<td>64 * 1440</td>
<td>0.94</td>
</tr>
<tr>
<td>Dot</td>
<td>256 * 480</td>
<td>0.64</td>
<td>256 * 960</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Optimization for BabelStream would require a different number of Threads*Teams for each of the sub-benchmarks

Courtesy: Justin Chang et al. (2022)
Agenda

1. Introduction to MI 200 hardware
2. Software stack and tools
3. Basics of OpenMP® offloading
4. HIP & OpenMP® - compatibility
5. Case studies
6. Heterogenous memory management (HMM)
Case Study 1 – VASP (Vienna Ab Initio Simulation Package)

- A computer program for atomic scale materials modelling, e.g., electronic structure calculations and quantum-mechanical molecular dynamics
- Currently used by more than 1400 research groups in academia and industry worldwide
- Software license agreements with the University of Vienna
- ~550K lines of FORTRAN 90 code (some FORTRAN 77)
Supporting concurrent directive-based paradigms in VASP

- Switch between different directive-based paradigms without letting them impact on each other
- Take advantage of source preprocessing
  - Pros: switch between different directive-based paradigms
  - Cons: makes the code messy

used when VASP is compiled with OpenACC

used when OpenMP (host) is enabled and OpenMP offloading/OpenACC is disabled

used when OpenMP offloading is enabled
Enable/disable offloading in different code paths

- Many of the VASP subroutines are called from different code paths
  - How can we enable offloading for a subroutine in one path and disable offloading for others
    - It would be useful for code development and debugging

We can call OMP_PUSH_EXEC_ON(.TRUE.) or OMP_PUSH_EXEC_ON(.FALSE.) to enable or disable offloading in different code paths.

```fortran
#include "symbol.inc"
MODULE moffload_struct_def
#ifndef _OFFLOAD
PUBLIC :: OMP_PUSH_EXEC_ON, OMP_POP_EXEC_ON
INTEGER, PARAMETER :: MAXLEVEL=20
INTEGER :: OMP_EXEC_ON_LEVEL=0
LOGICAL :: OMP_EXEC_ON_STACK(MAXLEVEL)=.FALSE.
LOGICAL, PUBLIC :: OMP_EXEC_ON=.TRUE.
CONTAINS
SUBROUTINE OMP_PUSH_EXEC_ON(VAR)
LOGICAL :: VAR
IF (OMP_EXEC_ON_LEVEL==MAXLEVEL) THEN
  WRITE(*,*) "OMP_PUSH_EXEC_ON: ERROR: stack is full"
ENDIF
OMP_EXEC_ON_LEVEL=OMP_EXEC_ON_LEVEL+1
OMP_EXEC_ON_STACK(OMP_EXEC_ON_LEVEL)=OMP_EXEC_ON
OMP_EXEC_ON=VAR
END SUBROUTINE OMP_PUSH_EXEC_ON

SUBROUTINE OMP_POP_EXEC_ON
IF (OMP_EXEC_ON_LEVEL==0) THEN
  WRITE(*,*) "OMP_POP_EXEC_ON: ERROR: stack is empty"
ENDIF
OMP_EXEC_ON=OMP_EXEC_ON_STACK(OMP_EXEC_ON_LEVEL)
OMP_EXEC_ON_LEVEL=OMP_EXEC_ON_LEVEL-1
END SUBROUTINE OMP_POP_EXEC_ON
#endif
END MODULE moffload_struct_def
```
Interface OMP offloading with ROCM libraries

- VASP uses FFT, BLAS, and LAPACK extensively
- Developed a wrapper to interface OMP target regions with ROCM libraries
  - rocFFT
  - rocBLAS
  - rocSolver

```fortran
SUBROUTINE OFF_ZGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
USE MROCBLAS
USE moffload_struct_def
USE moffload
INTEGER :: M, N, K, LDA, LDB, LDC
COMPLEX(q) :: A(LDA, COLNUM(TRANSA, K, M)), B(LDB, COLNUM(TRANSB, N, K)), C(LDC, N)
DO OFF !$OMP TARGET DATA USE_DEVICE_PTR(A, B, C)
  CALL HIP_ZGEMM(ROCBLAS_HANDLE, CHAR_TO_OP(TRANSA), CHAR_TO_OP(TRANSB), M, N, K, &
                  ALPHA, C_LOC(A), LDA, C_LOC(B), LDB, BETA, C_LOC(C), LDC)
DO OFF !$OMP END TARGET DATA
END SUBROUTINE OFF_ZGEMM
```

WOPT% CW_RED(A), CEIG(B), and WA% CW_RED(C) are mapped to device with "omp target enter data map" directive.
Exponential of Complex Variables

Original Code:

```fortran
program loop_test
  implicit none
  integer :: i
  complex(8) :: D,R

  D=(1,2)
  !$OMP TARGET MAP(FROM:R) MAP( TO:D) R=EXP(D)
  !$OMP END TARGET

  write(*,*) "R= ", R
end program loop_test
```

Note: This is a non-debug compiler; technical support should continue problem isolation using a debugger built for debugging.

```fortran
!------------------------------------------------------------------!
!
!  Error message :  Implement complex lib call for AMDGCN
!  Optimizer built : 2022-02-31 (production)
!
!------------------------------------------------------------------!
```
Exponential of Complex Variables

Workaround:

```plaintext
program loop_test
    implicit none
    integer :: i
    complex(8) :: D,R
    REAL :: CE_img

    D=(1,2)

    !$OMP TARGET MAP(FROM:R) MAP(TO:D)
    CE_img= AIMAG(D)
    R= 2.71828**(REAL(D))
    R=R*cmplx(COS(CE_img),SIN(CE_img))
    !$OMP END TARGET

    write(*,*) "R= ", R

end program loop_test
```

```plaintext
$ftn -hnoacc -fopenmp -homp -o ./exp ./exp_workaround2.f90
$ ./exp
R= (-1.1312035958327016,2.4717250246105067)

Exp(a+bj)=e^a*(cos(b)+sin(b))j
```
Mapping Scalar Variables

Original Code:

```fortran
program test
  real(8), target :: CE
  CE=0
  !$OMP TARGET ENTER DATA MAP(TO:CE)

  !$OMP TARGET
  CE=1
  !$OMP END TARGET

  !$OMP TARGET UPDATE FROM(CE)
  write(*,*) "CE= ", CE

  !$OMP TARGET EXIT DATA MAP(RELEASE:CE)
end program test
```

```bash
$ftn -hnoacc -fopenmp -homp -o ./enter_scalar ./enter_scalar.f90
$./enter_scalar
CE= 0.
```

Workaround:

```fortran
program test
  real(8), target :: CE
  CE=0
  !$OMP TARGET MAP(FROM:CE)

  !$OMP TARGET
  CE=1
  !$OMP END TARGET

  write(*,*) "CE= ", CE

end program test
```

```bash
$ftn -hnoacc -fopenmp -homp -o ./enter_scalar_workaround ./enter_scalar_workaround.f90
$./enter_scalar_workaround
CE= 1.
```
Pointer aliasing occurs a lot in VASP
- It can be challenging for the compilers to deal with pointer aliasing on device
- Set CRAY_ACC_DEBUG=3 as environment variable to get the log
- This issue is resolved in CCE15
Pointer aliasing (alternative methods)

Launch a kernel

Using target data construct
Pointer mismatch in subroutine calls

```fortran
MODULE wave_struct_def
  TYPE wavespin
    COMPLEX(8), POINTER :: CPTWFP(:,:)
    REAL, POINTER :: CR(:)
  END TYPE wavespin

  TYPE wavefun1
    COMPLEX(8), POINTER, CONTIGUOUS :: CPTWFP(:) => NULL()
    REAL, POINTER, CONTIGUOUS :: CR(:) => NULL()
    COMPLEX(8), POINTER, CONTIGUOUS :: CR(:) => NULL()
  END TYPE wavefun1

END MODULE wave_struct_def

program PointerAliasing
  use wave_struct_def
  TYPE (wavespin) :: W
  TYPE (wavefun1), TARGET :: W1(10)
  INTEGER NP, NSIM
  ALLOCATE(W%CPTWFP(100,100))
  ALLOCATE(W%CR(100,100))
  NSIM=10
  !$OMP TARGET ENTER DATA MAP(W1:W1)
  DO NP=1, NSIM
    CALL NEWAV_R(W1(NP))
  ENDDO
  DO NP=1, NSIM
    DO I=1, 10
      CALL SETWAV(W,NP,W1(I),I)
    ENDDO
  END DO
  CALL ECCP(W1(NP))
  ENDDO
  END DO
  END program
```

```fortran
SUBROUTINE NEWAV_R(W1)
  use wave_struct_def
  TYPE (wavefun1), TARGET(INOUT) :: W1
  INTEGER MPLW
  MPLW=10

  ALLOCATE(W%CR(MPLW))
  !$OMP TARGET ENTER DATA MAP(ALLOC:W%CR)
  !$OMP TARGET
  W%CR(:,1) = W%CR(:,1)
  !$OMP END TARGET
END SUBROUTINE

SUBROUTINE DELAV_R(W1)
  use wave_struct_def
  TYPE (wavefun1) :: W1
  !$OMP TARGET EXIT DATA MAP(DELETE:W1%CR)
  DEALLOCATE(W1%CR)
END SUBROUTINE

SUBROUTINE ECCP(W1)
  use wave_struct_def
  INTEGER MM
  COMPLEX(8), TARGET :: CE
  CE=0
  !$OMP TARGET TEAMs DISTIBUTE PARALLEL DO SIMD REDUCTION(+:CE)
  DO MM = 1, 100
    CE=CE+W%CRM(MM)
  ENDDO
  !$OMP END TARGET TEAMs DISTIBUTE PARALLEL DO SIMD
  write(*,*) "ce = ", CE
END SUBROUTINE

SUBROUTINE SETWAV(W,NP,W1, I)
  use wave_struct_def
  TYPE (wavespin), INTENT(IN) :: W
  TYPE (wavefun1), INTENT(INOUT) :: W1
  INTEGER I, NP
  !$omp target exit data map(DELETE:W1%CPTWFP)
  !$omp target exit data map(DELETE:W1%CRPROJ)
  W%CPTWFP = W%CPTWFP(:,I)
  W%CRPROJ = W%CRPROJ(:,I)
  !$omp target enter data map(TO:W1%CPTWFP,W1%CRPROJ)
END SUBROUTINE
```

Command line:
```bash
$ ./aliasing
ce = (100.,100.)
```

Device: callback Queue aborting with error : HSA_STATUS_ERROR_MEMORY_APERTURE_VIOLATION: The agent attempted to access memory beyond the largest legal address. code: 0x29
Aborted

$consoles/rocdevice.cpp .2660: 1637590862517 us:
86531: [tid:0x7fbe82217700] Device: callback Queue aborting with error : HSA_STATUS_ERROR_MEMORY_APERTURE_VIOLATION: The agent attempted to access memory beyond the largest legal address. code: 0x29
Aborted
Pointer mismatch in subroutine calls (alternative method)

```
MODULE wave_struct_def
  TYPE wavespin
    COMPLEX(8), POINTER :: CPTWFP(:,:)
    REAL , POINTER :: CPROJ(:,:)
  END TYPE wavespin

  TYPE wavefun1
    COMPLEX(8), POINTER, CONTIGUOUS :: CPTWFP(:) => NULL()
    REAL , POINTER, CONTIGUOUS :: CPROJ(:) => NULL()
  END TYPE wavefun1
END MODULE wave_struct_def

program PointerAliasing
  use wave_struct_def
  TYPE (wavespin) :: W
  TYPE (wavefun1), TARGET :: WI(10)
  INTEGER NP, NSIM

  ALLOCATE(W$CPTWFP(100,100))
  ALLOCATE(W$CPROJ(100,100))
  NSIM=10

  !$OMP TARGET ENTER DATA MAP(TO:W1)
  DO NP=1, NSIM
    CALL NEWWAV_R(W1(NP))
  ENDDO

  !$OMP TARGET TEAMs DISTRIBUTE PARALLEL DO SIMO REDUCTION(+:CE)
  DO MM=1,10
    CE=CE+W$CPTWFP(NP,MM)
  ENDDO
  !$OMP END TARGET TEAMs DISTRIBUTE PARALLEL DO SIMO

  WRITE(*,*) "ce=", CE
END SUBROUTINE

SUBROUTINE SETWAV(W,M)
  USE wave_struct_def
  TYPE (wavespin), TARGET(IN) :: W
  TYPE (wavefun1), TARGET(INOUT) :: WI
  INTEGER J, M

  !$OMP TARGET ENTER DATA MAP(DELETE:W$CPTWFP)
  !$OMP TARGET ENTER DATA MAP(DELETE:W$CPROJ)
  W$CPTWFP=$W$CPTWFP(:,1)
  W$CPROJ = W$CPROJ(:,1)
  !$OMP TARGET ENTER DATA MAP(DELETE:W$CPTWFP, W$CPROJ)
  END SUBROUTINE
```

CALL ECCP(W1,NP)
```
Atomic update for complex(8)

Original code

```fortran
program test
    integer :: i,j,N,M,k2,k
    complex(8) :: B(51,42), C(51,42), X
    N=3000
    M=100
    do i=1, 51
        do j=1, 41
            B(i,j)=0
            C(i,j)=0
        enddo
    enddo
    X=(1,1)
!$omp target teams distribute map(tofrom:8) private(k,k2)
    do i=1, M
        !$omp parallel do
        do j=1, N/M
            k=(i*(N/M)+j)
            k2=mod(k,40)+1
            call SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(B(k,k2),X)
        enddo
    enddo
!$omp end target teams distribute
    write(*,*) "B(1,1)@ time=" , B(1,1)@time
end program test
```

Alternative

```fortran
program test
    integer :: i,j,N,M,k2,k
    complex(8) :: B(51,42), C(51,42), X
    N=3000
    M=100
    do i=1, 51
        do j=1, 41
            B(i,j)=0
            C(i,j)=0
        enddo
    enddo
    X=(1,1)
!$omp target teams distribute map(tofrom:8) private(k,k2)
    do i=1, M
        !$omp parallel do
        do j=1, N/M
            k=(i*(N/M)+j)
            k2=mod(k,40)+1
            call SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(B(k,k2),X)
        enddo
    enddo
!$omp end target teams distribute
end program test
```
The overhead of subroutine call assuming there is no need for atomic update

Kernel time = 80 ms

Kernel time = 22 ms
Declare target

```fortran
PROGRAM reproducer

IMPLICIT NONE

INTEGER, PARAMETER :: DP = selected_real_kind(14, 209)
COMPLEX(DP), ALLOCATABLE :: psi(:, :, :, :), ew(:)
INTEGER :: n, notcnev, nbn, npwx, npol, nvecx, ierr, nbase, npw
REAL(DP), EXTERNAL :: MYDDOT_VECTOR_GPU

nbase = 1
n = 10
nbn = 2
notcnev = 1
npwx = 2
npw = 1
npol = 2
nvecx = 1
allocate(ew(n))
!$omp target data map(alloc: ew)
ALLOCATE( psi(npwx*npol, nvecx ), STAT=ierr )
!$omp target enter data map(alloc:psi)

!$omp target teams distribute private(nbn)
DO n = 1, notcnev
 nbn = nbase + n
 ew(n) = ew(n) + MYDDOT_VECTOR_GPU( 2*npw, psi(npwx+1,nbn), psi(npwx+1,nbn) )
END DO
!$omp target update from(ew)

!$omp end target data
deallocate(ew)
deallocate(psi)

END PROGRAM
```

```
DOUBLE PRECISION FUNCTION MYDDOT_VECTOR_GPU(N,DX,DY)
INTEGER, INTENT(IN) :: N
DOUBLE PRECISION, INTENT(IN) :: DX(*), DY(*)
DOUBLE PRECISION :: RES
INTEGER :: I
!$omp declare target
!$omp parallel do simd reduction(+:RES)
DO I = 1, N
 RES = RES + DX(I) * DY(I)
END DO
!$omp end parallel do simd
MYDDOT_VECTOR_GPU = RES
END FUNCTION MYDDOT_VECTOR_GPU
```

```
$make
fn-ffopenmp -c myddot.f90 -o myddot.o
!$omp parallel do simd reduction(+:RES)
fn-7212 fn: WARNING MYDDOT_VECTOR_GPU, File = myddot.f90, Line = 7
Variable "res" is used before it is defined.
fn-7256 fn: WARNING MYDDOT_VECTOR_GPU, File = myddot.f90, Line = 7
An OpenMP parallel construct in a target region is limited to a single thread.
Cray Fortran : Version 15.0.0.3 (20220920162820_088e5928c3724749216ddb5fbbcd2152ed62bb8)
Cray Fortran : Thu Jan 05, 2023 15:58:21
Cray Fortran : Compile time: 0.0472 seconds
Cray Fortran : 13 source lines
Cray Fortran : 13 source lines
Cray Fortran : 0 errors, 2 warnings, 0 other messages, 0 ansi
Cray Fortran : explain fn-message number* gives more information about each message.
fn -ffopenmp -c reproducer.f90 -o reproducer.o
fn -ffopenmp myddot.o reproducer.o -o reproducer.x
error: reproducer.f90:28:0: in function reproducer_$ck_L25_1 void(i64,i64,i64,i64,i64): unsupported call to variadic function myddot_vector_gpu
make: *** [Makefile:8: reproducer] Error 1
```
Declare target (alternative method)

To get around the error, we can define function in the same file as function call
- It would be challenging to apply his workaround in the applications with many function/subroutine calls
Case study 2 - MPAS

The Model for Prediction Across Scales (MPAS) is a collaborative project for developing atmosphere, ocean and other earth-system simulation components for use in climate, regional climate, and weather studies.

- Finite volume solver for non-hydrostatic atmospheric equations.
- Written in FORTRAN. Uses directives for GPU acceleration
  - ~2.5k lines of !$acc code, still an ongoing effort
  - AMD approach: OpenMP® directives

See https://mpas-dev.github.io/ and https://github.com/MPAS-Dev/MPAS-Model for more information
MPAS code structure: Memory and data management

- All GPU memory buffers allocated at the first time step and is reused for subsequent time steps.
- Updating the host from device occurs at the end of every time step.
- Now we can strictly focus on porting and optimizing the compute kernels.
Although the existing OpenACC code may not be efficiently implemented, it still serves as a rough guideline for our OpenMP offloading port.

First step of the porting & optimization process is to add existing OpenMP directives on top of the OpenACC directives.
Example #1: OpenMP® initial port

```cpp
#include <omp.h>

int main() {
  int nVertLevels = 26; // Example value for the JW Baroclinic Wave benchmark

  // OpenMP directives
  #pragma omp target teams distribute
  #pragma omp parallel vector_length(32)
  #pragma omp loop gang
  #pragma omp do iEdge=edgeStart,edgeEnd
  #pragma omp parallel do simd
  #pragma omp do k=nVertLevels
  #pragma omp parallel do simd
  #pragma omp do k=nVertLevels
  #pragma omp parallel do simd

  // Code snippet
  pgrad = ((rtheta_pp(k,cell1)-rtheta_pp(k,cell2))*invDcEdge(iEdge)) / (0.5*(zz(k,cell2)+zz(k,cell1)))
  pgrad = cu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
  pgrad = pgrad + 0.5*zu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
  ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)

  // End of code snippet
}
```

Note: number of vertical levels (nVertLevels) depends on mesh. (e.g., nVertLevels = 26 in the JW Baroclinic Wave benchmark)

This may be okay for a hardware with shorter SIMD (warp). With warp size exceeding the nVertLevels use of recourses will be suboptimal.
Example #1: OpenMP® initial optimization – number of threads

```cpp
!$omp target teams distribute thread_limit(64) 
!$omp parallel vector_length(32) 
!$acc loop gang 
do iEdge=edgeStart,edgeEnd 
  cell1 = cellsOnEdge(1,iEdge) 
  cell2 = cellsOnEdge(2,iEdge) 
  ! update edges for block-owned cells 
  if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then 
  !$omp parallel do simd 
  !$acc loop vector 
do k=1,nVertLevels 
    pgrad = ((rthetap(k,cell2)-rthetap(k,cell1))*invDcEdge(iEdge) )/(0.5*(zz(k,cell1)+zz(k,cell2))) 
    pgrad = cqu(k,iEdge)*0.5*c2*(exn(k,cell1)+exn(k,cell2))*pgrad 
    pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2)) 
    ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad) 
  end do 
  !$omp end parallel do simd 
  !$omp parallel do simd 
  !$acc loop vector 
do k=1,nVertLevels 
  ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge) 
  end do 
  !$omp end parallel do simd 
  end if! end test for block-owned cells 
  end do! end loop over edges 
!$acc end parallel 
!$omp end target teams distribute 
```

Default number of threads is 256

Obvious step, reduce it to 64.

Still <50% utilization. How to ensure most threads are doing useful work for these smaller meshes? One approach could be to collapse the inner do loops
Example #1: OpenMP® better optimization – collapsed do loops

Macro-definitions added at the top of each source file to distinguish do loops for the OpenACC backend from do loops for the new OpenMP offloading backend. End goal is to have one code with few adaptations for optimal use of different directive-based programming models.

Can now use default number of threads (256)
Example #2: OpenACC code

```
!$acc parallel vector_length(64)
!$acc loop gang private(wdzu, tend_wk, eoe_w, we_w)
do iEdge=edgesOnEdge, edgeSolveEnd
!$acc cache(tend_wk, wdzu, eoe_w, we_w)
cell1 = cellsOnEdge(1, iEdge)
cell2 = cellsOnEdge(2, iEdge)
!DIR$ IVDEP
!$acc loop vector
do k=1, nVertLevels
! Compute ...
end do
! Compute ...
!$acc loop vector shortloop
do j=1, nEdgesOnEdge(iEdge)
eoe_w(j) = edgesOnEdge(j, iEdge)
we_w(j) = weightsOnEdge(j, iEdge)
end do
18 !DIR$ IVDEP
19 !$acc loop vector
do k=1, nVertLevels
q1 = pv_edge(k, iEdge)
q2 = 0.0
end do
23 !$acc loop seq
do j = 1, nEdgesOnEdge(iEdge)
eoe = eoe_w(j)
workpv = 0.5 * (q1 + pv_edge(k, eoe))
q2 = q2 + we_w(j) * u(k, eoe) * workpv
end do
29 t_w = - rdzw(k) * (wdzu(k+1) - wdzu(k))
tend_u(k, iEdge) = t_w + rho_edge(k, iEdge) * &
(q2 - (ke(k, cell2) - ke(k, cell1)) * &
invDcEdge(iEdge)) * tend_wk(k) * 0.5 * &
(h_divergence(k, cell1) + h_divergence(k, cell2))
36 !$acc end parallel
```

Caches local arrays into shared memory
No OpenMP equivalent for !$acc cache
Collapsing inner loops not always possible
Example #2: OpenMP® initial port

Directly apply `thread_limit(64)` trick.

Suboptimal performance of RHS loop – register spills and scratch usage according to rocprof.

Collapsing the loops not possible because of the reduction of q2 variable.

However, can we rearrange the order of the parallel and sequential loops?
Example #2: OpenMP® optimization – rearranging and splitting loops

Rearrange order of the sequential and parallel do loops. Multiple parallel loops to minimize global memory reads/writes.
JW Baroclynic wave – Initial performance

Overall GPU port (including the OpenACC backend) still in progress
Only a couple variables copied back to the host about ~7% of time integration
  • The “mpas update GPU data on host” event will significantly increase as more physics/variables are ported
  • HMM can play a big role
1. Introduction to MI 200 hardware
2. Software stack and tools
3. Basics of OpenMP® offloading
4. HIP & OpenMP® - compatibility
5. Case studies
6. Heterogenous memory management (HMM)
Heterogenous memory management (HMM)

HMM allows the same pointer to an object to be used both by the CPU and a device [GPU] even if the physical location of the object were moved by the operating system or device driver. Furthermore, the device driver can control the policy of whether the current physical location of the object is in CPU or device memory.

OpenMP® programming on systems with HMM

A simple test example:

```c
#pragma omp requires unified_shared_memory
int main(){
    double * X, * Y, * Z;
    size_t N = (size_t) 1024*1024* sizeof ( double);
    X = new double[N];
    Y = new double[N];
    #pragma omp target teams distribute parallel for if( target:N >2000)
    for (size_t i = 0; i < N; ++i)
        X[i] = 0.000001*i
    #pragma omp target teams distribute parallel for if( target:N >2000)
    for (size_t i = 0; i < N; ++i)
        Y[i] = X[i]
    delete[] X; delete[] Y;
    return 0;
}
```

Highlights:

- Uses system memory allocators.
- “Pointer is a pointer” data can be accessed by threads running on any device, regardless of the current physical location of the data.
- HMM allows OS, driver, and HW to manage physical memory location, while OpenMP directives are used primarily for expressing parallelism and execution space (HOST, DEVICE 0, DEVICE 1, etc.).

Footnotes: manual management of data, memory location, and expression of parallelism (for example using HIP programming models) may provide higher performance. Some performance optimizations may also be done via using additional directives, clauses, and APIs.
Performance comparison of unified vs non-unified memory

With UNIFIED MEMORY

NO UNIFIED MEMORY
Memory allocations are decoupled from the “rest” of the code. This makes altering memory allocation easier but it also makes it difficult to understand the mapping between variables and memory allocations.
OpenFOAM® initial porting with HMM - choosing the executing device and expressing parallelism

```c
#include <omp.h>
#pragma omp requires unified_shared_memory

#pragma omp target teams distribute parallel for //LG
for (label cell=0; cell<nCells; cell++)
{
    pAPtr[cell] = wAPtr[cell] + beta*pAPtr[cell];
}
```

```
#pragma omp target teams distribute parallel for //map(tofrom:ApspiPtr[0:nCells])
for (label face=0; face<nFaces; face++)
{

#pragma omp atomic hint(AMD_fast_fp_atomics)
        ApspiPtr[uPtr[face]] += lowerPtr[face]*psiPtr[lPtr[face]];
        ApspiPtr[lPtr[face]] += upperPtr[face]*psiPtr[uPtr[face]];

#pragma omp atomic hint(AMD_fast_fp_atomics)
        ApspiPtr[lPtr[face]] += upperPtr[face]*psiPtr[uPtr[face]];
```
**HMM at work with OpenFOAM®**

Current state: after adding about 60 lines of OpenMP® target directives, ~50-60% of the code is executed on GPUs.

Switching execution between the CPU and GPU does not require explicit data transfers – HMM is moving pages as needed.

OpenMP standard and implementation are evolving/improving.
Summary

- OpenMP® offloading is compatible and competitive with HIP
- OpenMP® can interface to ROCm /HIP math libraries
- Performance of OpenMP® regions can be tuned by modifying the number of teams or threads
- Debugging and profiling OpenMP® offloading code on AMD GPUs
- Discussed the challenges in adding OpenMP® offloading support in HPC applications
- Compiler related challenges
  - Having a standard benchmark for capturing the compiler related issues would be helpful
  - Heterogeneous Memory Management (HMM) is available to use on AMD systems
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.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED "AS IS" WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

© 2023 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ROCm™, EPYC™, Instinct™ and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

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