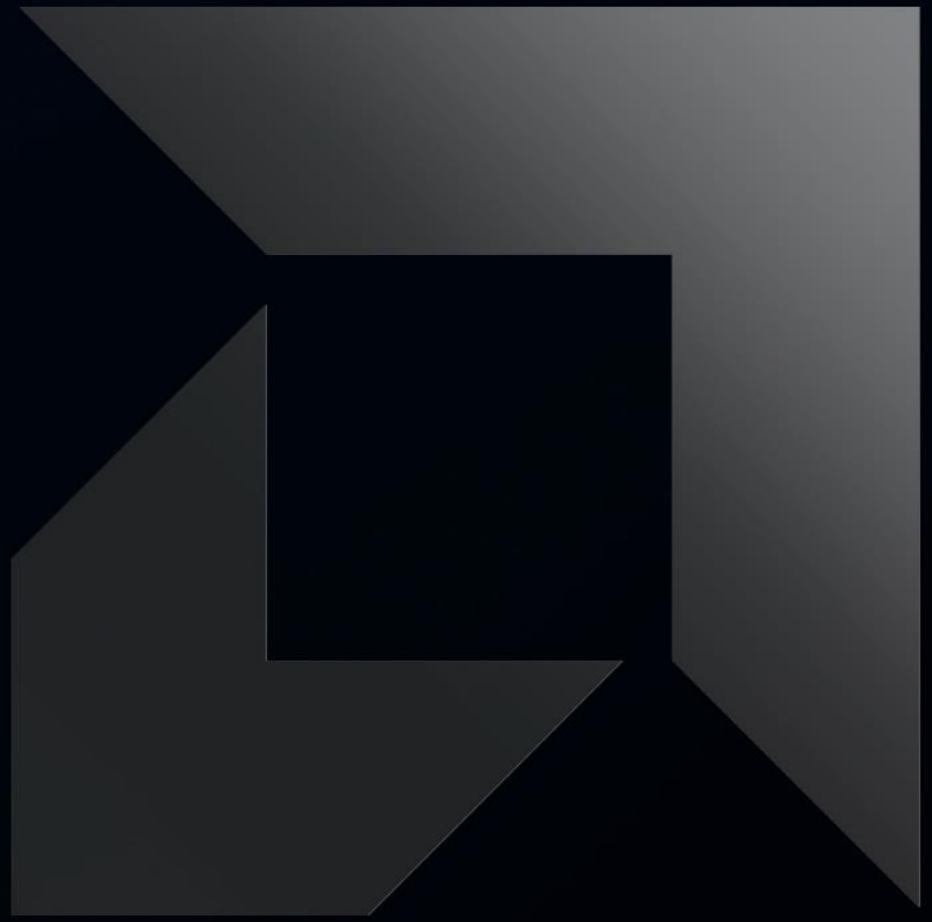


Accelerating HPC Applications on AMD Instinct™ GPUs with OpenMP® offloading:

An Overview

Suyash Tandon, Member of Technical Staff

May 2023

AMD

together we advance_

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.

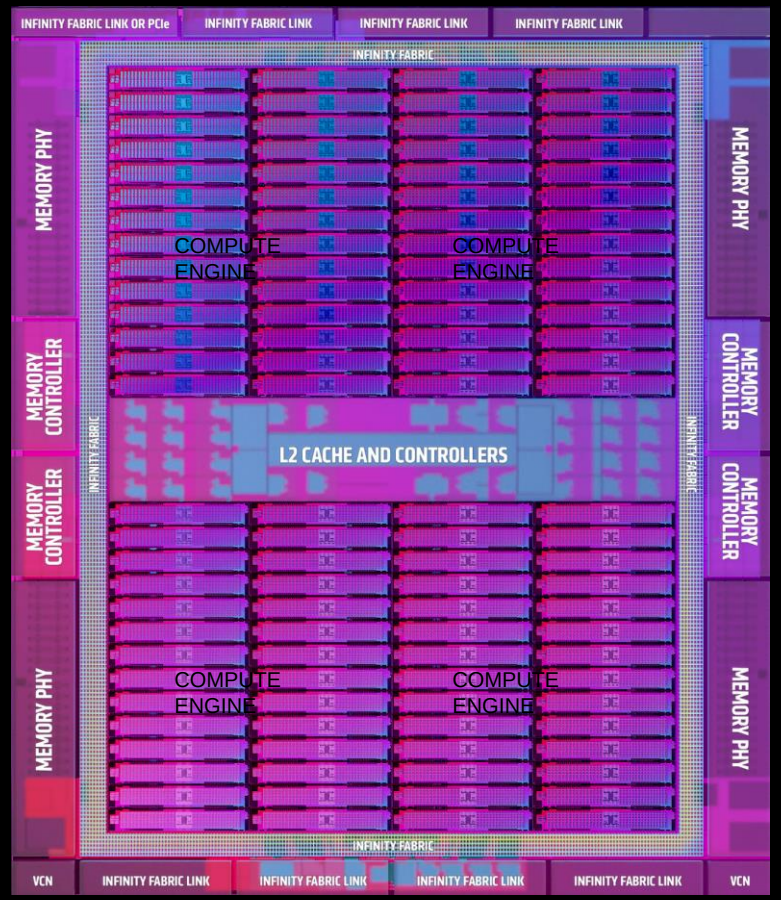
Agenda

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

Roadmaps Subject to Change

AMD
together we advance_



2ND GENERATION CDNA ARCHITECTURE

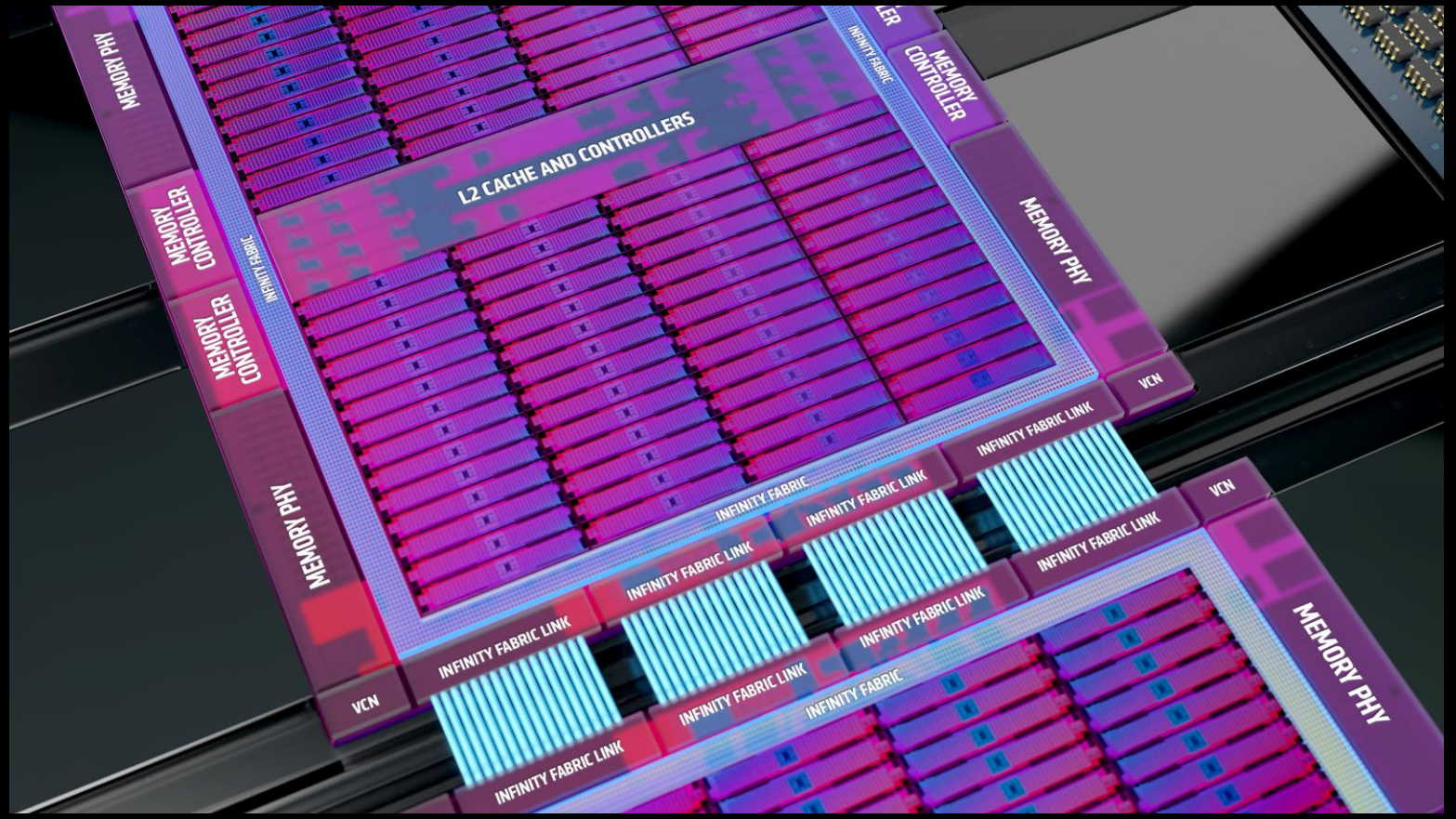
TAILORED-BUILT FOR HPC & AI

- 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-DIRECTIONALBANDWIDTH



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

AMD CDNA™ 2 white paper: https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

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!]

First U.S Exascale Systems

2012	2016	2018	2020	2021-2023

ORNL
Cray/AMD/NVIDIA

ANL
IBM BG/Q

LLNL
IBM BG/Q

ANL
Cray/Intel KNL

LBNL
Cray/Intel Xeon®/ KNL

LANL/SNL
Cray/Intel Xeon®/ KNL

ORNL
IBM/NVIDIA

LLNL
IBM/NVIDIA

LBNL
Cray/AMD/NVIDIA

LANL/SNL
HPE/Intel

ORNL
HPE/AMD/AMD

ANL
Intel/Cray

LLNL
HPE/AMD/AMD

Source: science.osti.gov/-media/ascr/ascac/pdf/meetings/201909/20190923_ASCAC-Helland-Barbara-Helland.pdf

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

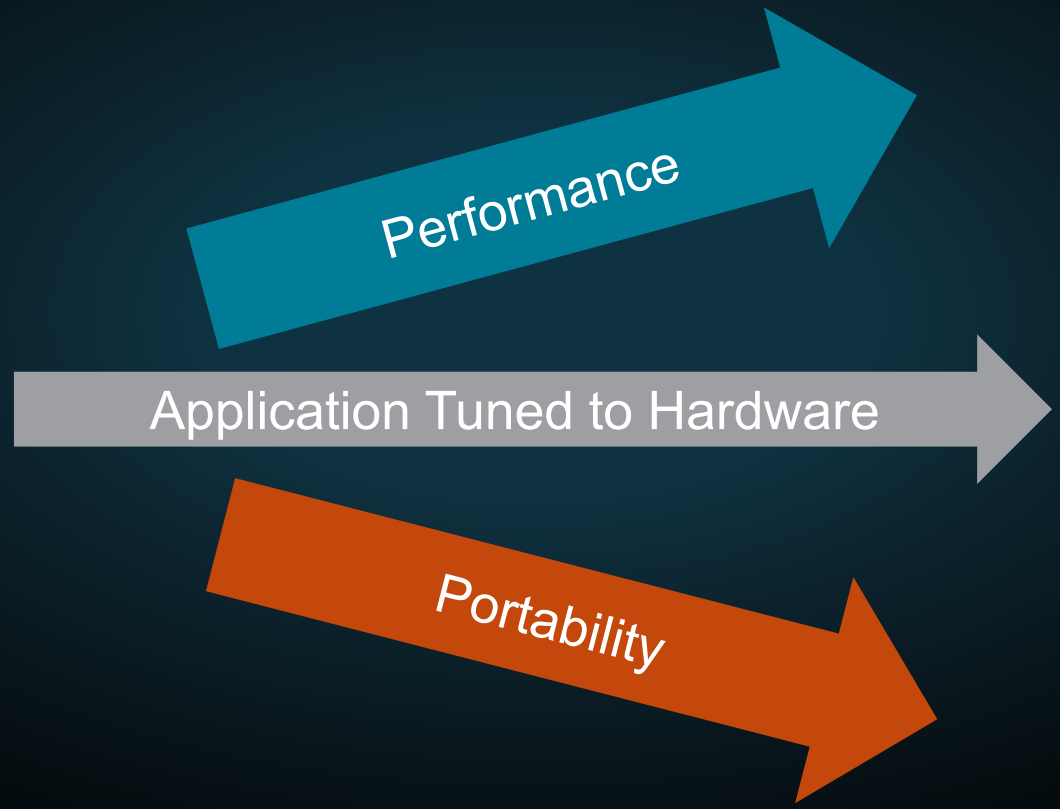
Productive

Optimize time to solution for new research

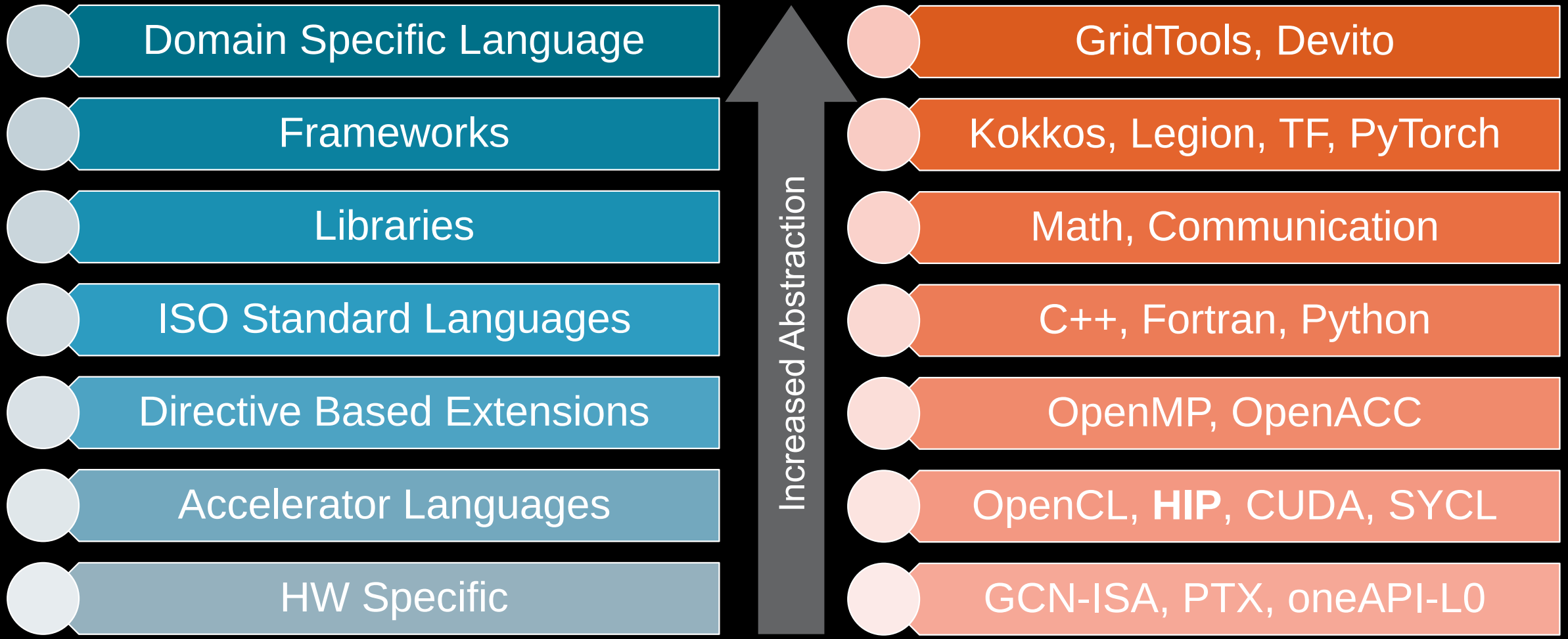
Abstract the computer science (code, data movement, scaling, etc)

PERFORMANCE VS PORTABILITY TRADEOFF

Portability drops as software is tuned for specific HW features



GPU PROGRAMMING IS DIFFICULT – BUT EASIER IF HW IS ABSTRACTED



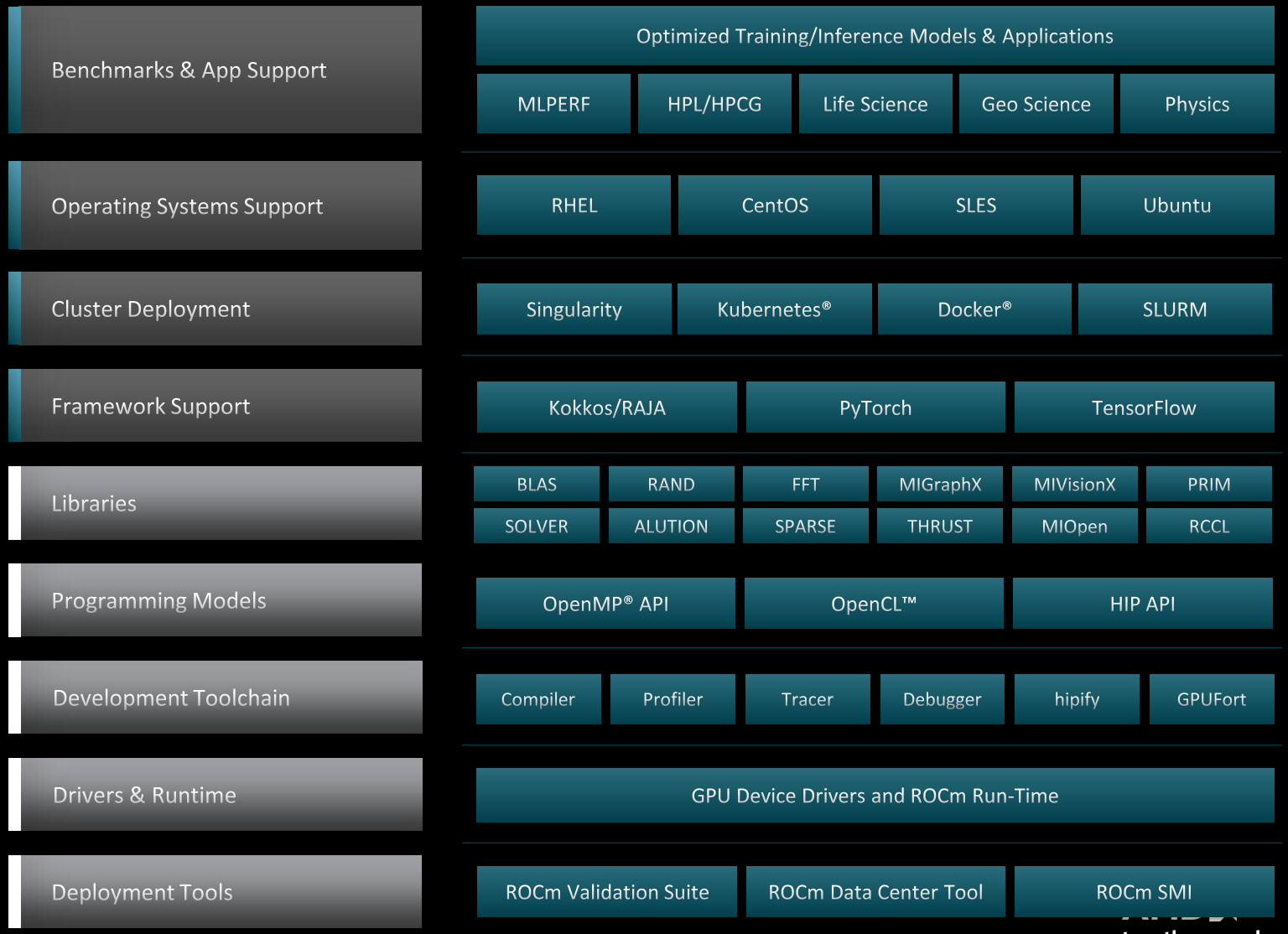
Agenda

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)

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



Compiler with OpenMP® support on AMD GPUs

	AOMP	CRAY	GNU (OG-12)
Stack	Packaged separately from ROCm Prototype new features for ROCmCC Same command line options as ROCmCC	Packaged with Cray Compiling Environment	Siemens' free GCC-based compiler Based on GCC-12 branch Support offloading to AMD CDNA™
Specifications	Complete 4.5 and partial 5.0 support	Complete 4.5 and partial 5.0 support	Complete 4.5 and partial 5.0 support
Enable OpenMP®	-fopenmp	-fopenmp or -homp	-fopenmp
Select GPU Targets (AOMP-15.0-2 or newer)	--offload-arch=<arch-name> Replaces -fopenmp-targets, -Xopenmp-target, and -march options	--offload-arch=<arch-name> Replaces -fopenmp-targets, -Xopenmp-target, and -march options	--foffload='-march=<arch-name>' Replaces -fopenmp-targets, -Xopenmp-target, and -march options
Optimizations	-O0 -O1 & above -famd-opt -ffast-math	-O0 -O1 & above -hfpN, -ffp=N	-O0 -O1 & above -famd-opt -ffast-math
Debug	-g	-g	-g

AMD development tools

ROC-profiler (rocprof)

Hardware Counters	Raw collection of GPU counters and traces	
	Counter collection with user input files	Counter results printed to a CSV
Traces and timelines	Trace collection support for	
	CPU copy	HIP API	HSA API	GPU Kernels
Visualisation	Traces visualized with Perfetto	

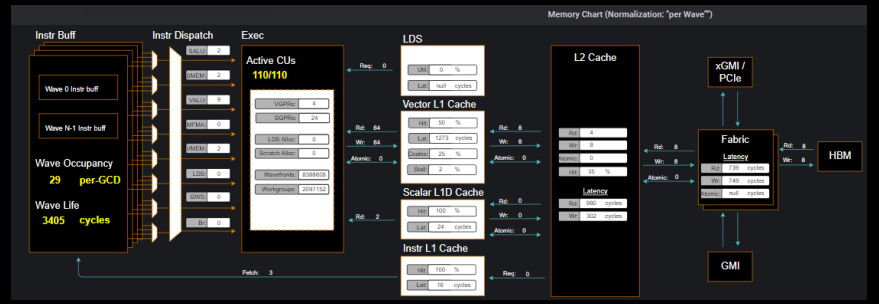
Omnitrace

Trace collection	Comprehensive trace collection				
	CPU	GPU			
Supports	CPU copy	HIP API	HSA API	GPU Kernels	
	OpenMP®	MPI	Kokkos	p-threads	multi-GPU
Visualisation	Traces visualized with Perfetto				

Omniperf

Performance Analysis	Automated collection of hardware counters			
	Analysis	Visualisation		
Supports	Speed of Light	Memory chart	Rooflines	Kernel comparison
Visualisation	With Grafana or standalone GUI			

	A	B	C	D	E
1	Name	Calls	TotalDura	AverageN	Percentage
2	hipMemcpyAsync	99	3.22E+10	3.25E+08	44.14872
3	hipEventSynchronize	330	2.42E+10	73394557	33.225
4	hipMemsetAsync	87	7.76E+09	89232696	10.64953
5	hipHostMalloc	9	5.41E+09	6.01E+08	7.415198
6	hipDeviceSynchronize	28	1.32E+09	47006288	1.805515
7	hipHostFree	17	1.05E+09	61534688	1.435014
8	hipMemcpy	41	8.11E+08	19791876	1.113161
9	hipLaunchKernel	1856	58082083	31294	0.079676
10	hipStreamCreate	2	46380834	23190417	0.063625
11	hipMemset	2	18847246	9423623	0.025854
12	hipStreamDestroy	2	15183338	7591669	0.020828
13	hipFree	38	8269713	217624	0.011344
14	hipEventRecord	330	2520035	7636	0.003457
15	hipMalloc	30	1484804	49493	0.002037
16	__hipPopCallConfigura	1856	229159	123	0.000314
17	__hipPushCallConfigur	1856	224177	120	0.000308
18	hipGetLastError	1494	100458	67	0.000138
19	hipEventCreate	330	76675	232	0.000105
20	hipEventDestroy	330	64671	195	8.87E-05
21	hipGetDevicePropertie	47	51808	1102	7.11E-05
22	hipGetDevice	64	11611	181	1.59E-05
23	hipSetDevice	1	401	401	5.50E-07
24	hipGetDeviceCount	1	220	220	3.02E-07



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

```
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

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

```
coe62819@cedar003:~/kernel/teamsdist> ./teamsdis
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from ./teamsdis2.f90:13
ACC:       allocate 'a(:)' (4000 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC: Execute kernel test_$ck_L13_1 blocks:1 threads:256 async(auto) from ./teamsdis2.f90:13
ACC: Wait async(auto) from ./teamsdis2.f90:17
ACC: Start transfer 1 items from ./teamsdis2.f90:17
ACC:       copy to host, free 'a(:)' (4000 bytes)
ACC: End transfer (to acc 0 bytes, to host 4000 bytes)
 A(1)= (2.,2.)
```

```
coe62819@cedar003:~/kernel/teamsdist> ./teamsdis
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from ./teamsdis2.f90:13
ACC:       allocate 'a(:)' (4000 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC: Execute kernel test_$ck_L13_1 blocks:2 threads:256 async(auto) from ./teamsdis2.f90:13
ACC: Wait async(auto) from ./teamsdis2.f90:17
ACC: Start transfer 1 items from ./teamsdis2.f90:17
ACC:       copy to host, free 'a(:)' (4000 bytes)
ACC: End transfer (to acc 0 bytes, to host 4000 bytes)
 A(1)= (2.,2.)
```

Basics of OpenMP® offloading

OMP TARGET	Defines a target region to be offloaded on device

```
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

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

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
```

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
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
```

LIBOMPTARGET_KERNEL_TRACE	Print useful statistics for device operations. Setting to 1 emits name of every kernel, number of teams, threads, and register usage. Setting to 2 prints timing and data transfer information.
LIBOMPTARGET_INFO	Value of 1 or higher to print information from device runtime. Setting to -1 will print all information
LIBOMPTARGET_DEBUG	Setting to 1 emits further detailed debugging information about data transfer operations and kernel launch.

LIBOMPTARGET_DEBUG = 1

```
Libomptarget --> Init target library!
Libomptarget --> OMPT: library_ompt_connect = libomp_ompt_connect
Libomptarget --> OMPT: library_ompt_connect = 0x7f16ea9b1bd0
Libomptarget --> OMPT: Exit ompt_init
Libomptarget -->  register_image_info image 0 of 1 offload-arch:gfx90a VERSION:1
Libomptarget --> Loading RTLs...
Libomptarget --> Loading library 'libomptarget.rtl.ppc64.so'...
Libomptarget --> Unable to load 'libomptarget.rtl.ppc64.so': libomptarget.rtl.ppc64.so: cannot open shared object file: No such file or directory!
Libomptarget --> Loading library 'libomptarget.rtl.x86_64.so'...
Libomptarget --> Successfully loaded library 'libomptarget.rtl.x86_64.so'!
Libomptarget --> Registering RTL libomptarget.rtl.x86_64.so supporting 4 devices!
Libomptarget --> Loading library 'libomptarget.rtl.cuda.so'...
Target CUDA RTL --> Start initializing CUDA
Target CUDA RTL --> Unable to load library 'libcuda.so': libcuda.so: cannot open shared object file: No such file or directory!
Target CUDA RTL --> Failed to load CUDA shared library
Libomptarget --> Successfully loaded library 'libomptarget.rtl.cuda.so'!
Libomptarget --> No devices supported in this RTL
Libomptarget --> Loading library 'libomptarget.rtl.aarch64.so'...
Libomptarget --> Unable to load 'libomptarget.rtl.aarch64.so': libomptarget.rtl.aarch64.so: cannot open shared object file: No such file or directory!
Libomptarget --> Loading library 'libomptarget.rtl.ve.so'...
Libomptarget --> Unable to load 'libomptarget.rtl.ve.so': libomptarget.rtl.ve.so: cannot open shared object file: No such file or directory!
Libomptarget --> Loading library 'libomptarget.rtl.amdgpu.so'...
Target AMDGPU RTL --> Start initializing AMDGPU
Target AMDGPU RTL --> There are 8 devices supporting HSA.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Alloc allowed in memory pool check failed: HSA_STATUS_ERROR: A generic error has occurred.
Target AMDGPU RTL --> Device 0: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> Device 1: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> Device 2: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> Device 3: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> Device 4: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> Device 5: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> Device 6: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> Device 7: Initial groupsPerDevice 128 & ThreadsPerGroup 256
Target AMDGPU RTL --> OMPT: Entering ompt_init
Target AMDGPU RTL --> OMPT: library_ompt_connect = libomptarget_ompt_connect
Target AMDGPU RTL --> OMPT: library_ompt_connect = 0x7f16ea889f80
Libomptarget --> OMPT: Enter libomptarget_ompt_connect
Libomptarget --> OMPT: Leave libomptarget_ompt_connect
Target AMDGPU RTL --> OMPT: Exiting ompt_init
Libomptarget --> Successfully loaded library 'libomptarget.rtl.amdgpu.so'!
Libomptarget --> Registering RTL libomptarget.rtl.amdgpu.so supporting 8 devices!
Libomptarget --> Loading library 'libomptarget.rtl.rpc.so'...
Libomptarget --> Unable to load 'libomptarget.rtl.rpc.so': libomptarget.rtl.rpc.so: cannot open shared object file: No such file or directory!
Libomptarget --> RTLs loaded!
Libomptarget --> Image 0x0000563eb018fb40 is NOT compatible with RTL libomptarget.rtl.x86_64.so!
Libomptarget --> Image 0x0000563eb018fb40 is compatible with RTL libomptarget.rtl.amdgpu.so!
Libomptarget --> RTL 0x0000563eb18eda40 has index 0!
Libomptarget --> Registering image 0x0000563eb018fb40 with RTL libomptarget.rtl.amdgpu.so!
Libomptarget --> Done registering entries!
 M=   (2.000000,2.000000)
Target AMDGPU RTL --> Finalizing the AMDGPU DeviceInfo.
Libomptarget --> Unloading target library!
Libomptarget --> Image 0x0000563eb018fb40 is compatible with RTL 0x0000563eb18eda40!
Libomptarget --> Unregistered image 0x0000563eb018fb40 from RTL 0x0000563eb18eda40!
Libomptarget --> Done unregistering images!
Libomptarget --> Removing translation table for descriptor 0x0000563eb01931f8
Libomptarget --> Done unregistering library!
Libomptarget --> Deinit target library!
```

AMD
together we advance_

Debugging with Cray compiler: CRAY_ACC_DEBUG

```
 1 program target_example
 2         complex :: M,N
 3         N=(2,2)
 4         M=(0,0)
 5
 6         !$omp target map(from:M) map(to:N)
 7         M=N
 8         !$omp end target
 9
10         write(*,*) "M= ", M
11
12 end program target_example
```

CRAY_ACC_DEBUG=1

```
ACC: Transfer 2 items (to acc 8 bytes, to host 0 bytes) from ./teamsdis3.f90:6
ACC: Execute kernel target_example_$ck_L6_1 async(auto) from ./teamsdis3.f90:6
ACC: Wait async(auto) from ./teamsdis3.f90:8
ACC: Transfer 2 items (to acc 0 bytes, to host 8 bytes) from ./teamsdis3.f90:8
 M=  (2.,2.)
```

CRAY_ACC_DEBUG=2

```
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 2 items from ./teamsdis3.f90:6
ACC:       allocate 'm' (8 bytes)
ACC:       allocate, copy to acc 'n' (8 bytes)
ACC: End transfer (to acc 8 bytes, to host 0 bytes)
ACC: Execute kernel target_example_$ck_L6_1 blocks:1 threads:1 async(auto) from ./teamsdis3.f90:6
ACC: Wait async(auto) from ./teamsdis3.f90:8
ACC: Start transfer 2 items from ./teamsdis3.f90:8
ACC:       copy to host, free 'm' (8 bytes)
ACC:       free 'n' (8 bytes)
ACC: End transfer (to acc 0 bytes, to host 8 bytes)
 M=  (2.,2.)
```

CRAY_ACC_DEBUG=3

```
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Compute level 9.0
ACC: Device Name:
ACC: Number of cus 120
ACC: Device name
ACC: AMD GCN arch name: gfx908:sramecc+:xnack-
ACC: Max shared memory 65536
ACC: Max thread blocks per cu 8
ACC: Max concurrent kernels 8
ACC: Async table size 8
ACC: Set Thread Context
ACC: Establish link bewteen libcrayacc and libcraymp
ACC:   libcrayacc interface v5
ACC:    libcraymp interface v5
ACC: Start transfer 2 items from ./teamsdis3.f90:6
ACC:   flags:
ACC:
ACC:   Trans 1
ACC:       Simple transfer of 'm' (8 bytes)
ACC:            host ptr 4053c0
ACC:            acc  ptr 0
ACC:            flags: ALLOCATE ACQ_PRESENT REG_PRESENT
ACC:            memory not found in present table
ACC:            allocate (8 bytes)
ACC:              get new reusable memory, added entry
ACC:            new allocated ptr (7f4d67608000)
ACC:            add to present table index 0: host 4053c0 to 4053c8, acc 7f4d67608000
ACC:            new acc ptr 7f4d67608000
ACC:
ACC:   Trans 2
ACC:       Simple transfer of 'n' (8 bytes)
ACC:            host ptr 4053c8
ACC:            acc  ptr 0
ACC:            flags: ALLOCATE COPY_HOST_TO_ACC ACQ_PRESENT REG_PRESENT
ACC:            memory not found in present table
ACC:            allocate (8 bytes)
ACC:              get new reusable memory, added entry
ACC:            new allocated ptr (7f4d67609000)
ACC:            add to present table index 1: host 4053c8 to 4053d0, acc 7f4d67609000
ACC:            copy host to acc (4053c8 to 7f4d67609000)
ACC:                internal copy host to acc (host 4053c8 to acc 7f4d67609000) size = 8
ACC:            new acc ptr 7f4d67609000
ACC:
ACC: End transfer (to acc 8 bytes, to host 0 bytes)
ACC:
ACC: Start kernel target_example_$ck_L6_1 async(auto) from ./teamsdis3.f90:6
ACC:         flags: CACHE_MOD CACHE_FUNC AUTO_ASYNC
ACC:     mod cache:  0x405640
ACC: kernel cache:  0x405440
ACC:    async info:  0x7f4d7b0918d0
ACC:     arguments: GPU argument info
ACC:             param size:  16
ACC:           param pointer:  0x7ffcd9b8ffc0
```

Debugging with Cray compiler: –hlist=aimd

teamdis.f90

```
1  program test
2          integer :: i
3          complex, pointer :: A(:)
4
5          allocate(A(500))
6
7          do i=1, 500
8          A(i) = (0,0)
9          enddo
10
11         !$omp target teams distribute parallel do simd map(from:A)
12         do i=1, 500
13         A(i)= (2,2)
14         enddo
15         !$omp end target teams distribute parallel do simd
16
17         write(*,*) "A(1)= ", A(1)
18
19 end program test
```

*.lst

```
    1.           program test
    2.                   integer :: i
    3.                   complex, pointer :: A(:)
    4.
    5.                   allocate(A(500))
    6.
    7.    A-----<        do i=1, 500
ftn-6202 ftn: VECTOR TEST, File = teamsdis.f90, Line = 7
  A loop starting at line 7 was replaced by a library call.

    8.    A              A(i) = (0,0)
    9.    A----->        enddo
   10.
   11.  + MG----<        !$omp target teams distribute parallel do simd map(from:A)
ftn-6405 ftn: ACCEL TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was placed on the accelerator.

ftn-6823 ftn: THREAD TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was multi-threaded.

ftn-6420 ftn: ACCEL TEST, File = teamsdis.f90, Line = 11
  If not already present: allocate memory for user shaped variable "a" on accelerator, copy back at line 15 (acc_cop

ftn-6823 ftn: THREAD TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was multi-threaded.

ftn-6823 ftn: THREAD TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was multi-threaded.

ftn-7256 ftn: WARNING TEST, File = teamsdis.f90, Line = 11
  An OpenMP parallel construct in a target region is limited to a single thread.

   12.    MG g--<        do i=1, 500
ftn-6430 ftn: ACCEL TEST, File = teamsdis.f90, Line = 12
  A loop starting at line 12 was partitioned across the threadblocks and the 256 threads within a threadblock.

   13.    MG g           A(i)= (2,2)
   14.    MG g-->        enddo
   15.    MG---->        !$omp end target teams distribute parallel do simd
   16.
   17.                   write(*,*) "A(1)= ", A(1)
   18.
   19.           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:

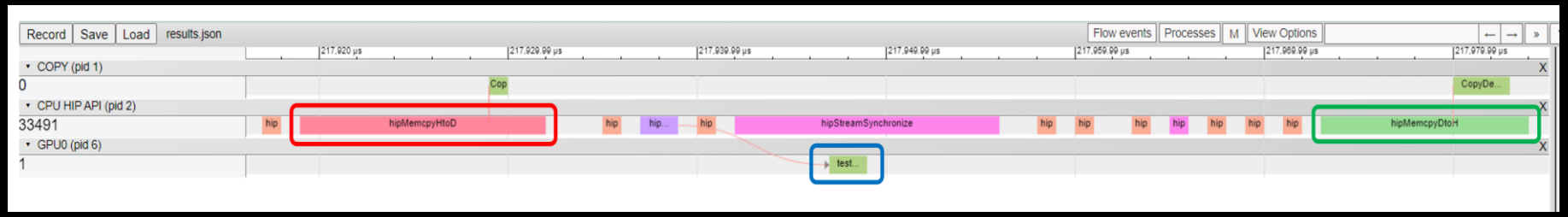
```
$ftn -hnoacc -fopenmp -homp -o ./test ./test.f90
```

Profile and collect HIP trace:

```
$rocprof –hip-trace ./test
```

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

```
program test
      integer:: i,j
      real, pointer:: A(:)

      allocate(A(100))

      A=0

      write(*,*) "A(1)= ", A(1)

      do j=1, 10
!$omp target enter data map(to:A)

!$omp target parallel do
      do i=1, 100
      A(i)=1
      enddo
!$omp end target parallel do

!$omp target update from(A
!$omp target exit data map(delete:A)

      enddo
      write(*,*) "A(1)= ", A(1)
      end program test
```



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

HIP & OpenMP® – Saxpy example

Basic profiling with rocprof:

```
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);
          }
}
```

```
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:

```
 __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 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);
}
```

Translation unit 1

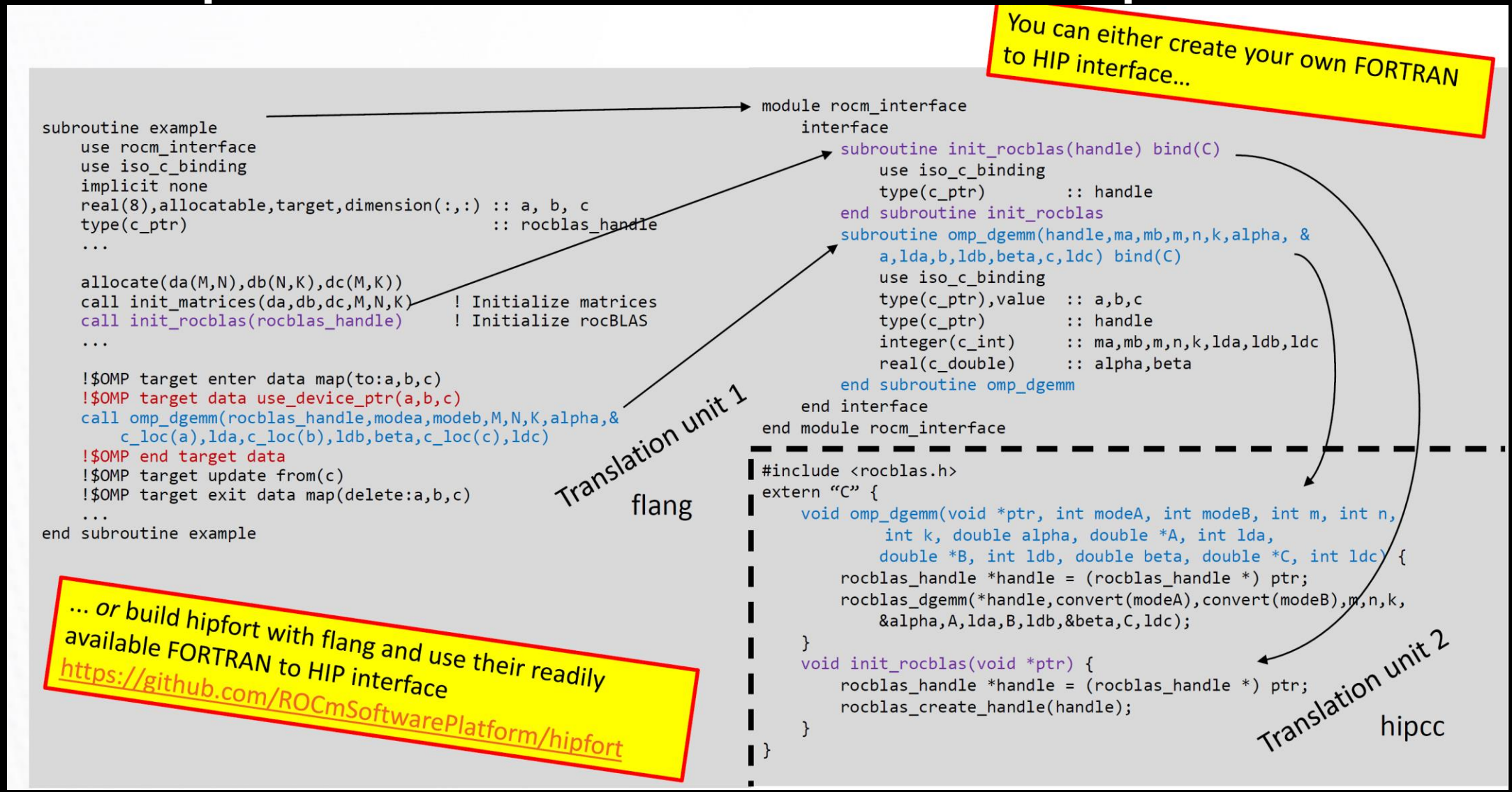
hipcc

```
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);
}
```

Translation unit 2

clang

HIP & OpenMP® – Fortran and DGEMM example



Courtesy: Justin Chang et al. (2022)

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

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

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

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

```
#ifdef _OFFLOAD
#define DOOFF
#define DOOMP                   !!
#else
#define DOOFF                   !!
#define DOOMP
#endif
```

Used when VASP is compiled with OpenACC

```
!$ACC PARALLEL LOOP PRESENT(CH,CW,DATAKE,WDES1) PRIVATE(MM)
DOOMP NOACC !$OMP PARALLEL DO SHARED(WDES1,CH,ISPINOR,DATAKE,EVALUE) PRIVATE(M,MM)
DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD PRIVATE(M,MM)
        DO M=1,WDES1%NGVECTOR
           MM=M+ISPINOR*WDES1%NGVECTOR
           CH(MM)=CH(MM)+CW(MM)*(WDES1%DATAKE(M,ISPINOR+1)-EVALUE)
        ENDDO
```

Used when OpenMP offloading is enabled

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

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

```
      DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO COLLAPSE(2) REDUCTION(+:EKIN) PRIVATE(MM,CPT) IF(OMP_EXEC_ON)
      DO ISPINOR=0,WDES1%NRSPINORS-1
         DO M=1,WDES1%NGVECTOR
            MM=M+ISPINOR*WDES1%NGVECTOR
            CPT=W1%CW(MM)
            EKIN =EKIN+ REAL( CPT*CONJG(CPT) ,KIND=q) * WDES1%DATAKE(M,ISPINOR+1)
         ENDDO
      ENDDO
      DOOFF !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO
```

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

```
#include "symbol.inc"
      MODULE moffload_struct_def
#ifdef _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

```
      CALL OFF_ZGEMM('N', 'N', m_ WDES1%NPL_RED , NSIM_, NSIM_*ITER, one, &
      &              WOPT%CW_RED(1,1), m_ WDES%NRPLWV_RED, CEIG(1,1), NSUBD,  &
      &              zero, WA%CW_RED(1,NPOS_RED+1), m_ WDES%NRPLWV_RED)
```

```
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
CHARACTER(1)     :: TRANSA,TRANSB
COMPLEX(q)       :: A(LDA,COLNUM(TRANSA,K,M)),B(LDB,COLNUM(TRANSB,N,K)),C(LDC,N)
COMPLEX(q)       :: ALPHA,BETA

DOOFF !$OMP TARGET DATA USE_DEVICE_PTR(A,B,C)
   CALL HIP_ZGEMM(ROCBLAS_HANDLE,CHAR_TO_OP(TRANSA),CHAR_TO_OP(TRANSB),M,N,K, &
          ALPHA,C_LOC(A),LDA,C_LOC(B),LDB,BETA,C_LOC(C),LDC)
DOOFF !$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

```
void hip_zgemm(void *ptr, int modeA, int modeB, int m, int n, int k, double _Complex alpha, double _Complex *A, int lda,
        double _Complex *B, int ldb, double _Complex beta, double _Complex *C, int ldc) {
    rocblas_handle *handle = (rocblas_handle *) ptr;
    rocblas_double_complex *A2 = reinterpret_cast<rocblas_double_complex*>(A);
    rocblas_double_complex *B2 = reinterpret_cast<rocblas_double_complex*>(B);
    rocblas_double_complex *C2 = reinterpret_cast<rocblas_double_complex*>(C);
    rocblas_double_complex *alpha2 = reinterpret_cast<rocblas_double_complex*>(&alpha);
    rocblas_double_complex *beta2 = reinterpret_cast<rocblas_double_complex*>(&beta);
    rocblas_zgemm(*handle,findop(modeA),findop(modeB),m,n,k,alpha2,A2,lda,B2,ldb,beta2,C2,ldc);
}
```

Exponential of Complex Variables

Original Code:

```
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
```

```
coe62819@cedar004:~/kernel/exp>
coe62819@cedar004:~/kernel/exp> ftn -hnoacc -fopenmp -homp -o ./exp ./exp.f90
   Error message      ::  Unimplemented: Complex lib call for AMDGCN
   Error detected     ::  File 'pdgcs/llvm-call-expr.c', line 283
   Optimizer built    ::  2022-02-11 (production)

   File               ::  ./exp.f90
   Function           ::  loop_test_
   at or near line    ::  11

   File path          ::  /home/users/coe62819/kernel/exp/exp.f90
   Compiler hash      ::  47886aea6358792836950db7cc33c2061a1a9d36
   Target             ::  Heterogeneous
     CPU              ::  x86-rome
     ACCEL            ::  amdgcn-gfx908

Creating internal compiler error backtrace (please wait):
[0x0000000110d873] linux_backtrace ??:?
[0x0000000110e752] pdgcs_internal_error(char const*, char const*, int) ??:?
[0x000000012d263c] llvm_cg::gen_llvm_complex_lib_call_expr(EXP_INFO, std::pair<llvm::Value*, llvm::Value*>*) ??:?
[0x0000000139ee4d] llvm_cg::gen_llvm_unary_expr(EXP_INFO) ??:?
[0x0000000134f9a9] llvm_cg::gen_llvm_expr(EXP_INFO, bool) ??:?
[0x0000000131b524] llvm_cg::gen_llvm_expr_stmt(int) llvm-stmt.c:?
[0x0000000131bcc6] llvm_cg::gen_llvm_stmt(int) ??:?
[0x000000011e0b17] llvm_cg::llvm_function_with_body() llvm-pdgcs.c:?
[0x000000011e448c] internal_llvm_function(int, int, int, int, llvm_cg::FunctionTranslateMode) llvm-pdgcs.c:?
[0x000000011e6a52] llvm_function(int) ??:?
[0x00000000718dca] PDGCS_do_proc ??:?
[0x0000000067e7f4] cvrt_proc_to_pdg m_cvrt.c:?
[0x0000000067f2a8] m_cvrt_to_pdg ??:?
[0x000000006abd64] process_scp m_i_control.c:?
[0x000000006ad57e] m_start_ipa ??:?
[0x000000005ab3b9] main ??:?
[0x007f56974eb34c] ?? ??:0
[0x00000000639e3d] _start /home/abuild/rpmbuild/BUILD/glibc-2.19/csu/../sysdeps/x86_64/start.S:122

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

ftn-7991 ftn: INTERNAL LOOP_TEST, File = ./exp.f90, Line = 11
  INTERNAL COMPILER ERROR:  "Unimplemented: Complex lib call for AMDGCN" (pdgcs/llvm-call-expr.c, line 283, version 47886aea6358792836950db7cc33c2061a1a9d36)
coe62819@cedar004:~/kernel/exp>
coe62819@cedar004:~/kernel/exp>
coe62819@cedar004:~/kernel/exp> vi exp.f90
coe62819@cedar004:~/kernel/exp> cp exp.f90 exp2.f90
coe62819@cedar004:~/kernel/exp> vi exp2.f90
coe62819@cedar004:~/kernel/exp>
coe62819@cedar004:~/kernel/exp>
coe62819@cedar004:~/kernel/exp> ftn -hnoacc -fopenmp -homp -o ./exp ./exp2.f90
   Error message      ::  Unimplemented: Complex lib call for AMDGCN
   Error detected     ::  File 'pdgcs/llvm-call-expr.c', line 283
   Optimizer built    ::  2022-02-11 (production)

   File               ::  ./exp2.f90
   Function           ::  loop_test_
   at or near line    ::  10

   File path          ::  /home/users/coe62819/kernel/exp/exp2.f90
   Compiler hash      ::  47886aea6358792836950db7cc33c2061a1a9d36
   Target             ::  Heterogeneous
     CPU              ::  x86-rome
     ACCEL            ::  amdgcn-gfx908

Creating internal compiler error backtrace (please wait):
[0x0000000110d873] linux_backtrace ??:?
[0x0000000110e752] pdgcs_internal_error(char const*, char const*, int) ??:?
[0x000000012d263c] llvm_cg::gen_llvm_complex_lib_call_expr(EXP_INFO, std::pair<llvm::Value*, llvm::Value*>*) ??:?
```

Exponential of Complex Variables

Workaround:

```
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
```

```
$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:

```
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
```

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

Workaround:

```
program test
      real(8),target :: CE
      CE=0

      !$OMP TARGET MAP(FROM:CE)
      CE=1
      !$OMP END TARGET

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

end program test
```

```
$ftn -hnoacc -fopenmp -homp -o
./enter_scalar ./enter_scalar_workaround.f90
$./enter_scalar
CE=  1.
```

Pointer aliasing

```
 1 MODULE wave_struct_def
 2 TYPE wavedes
 3         INTEGER,POINTER :: LMMAXX(:)
 4         END TYPE wavedes
 5
 6 TYPE wavedes1
 7         INTEGER,POINTER :: LMMAXX(:) => NULL()
 8         END TYPE wavedes1
 9 END MODULE wave_struct_def
10
11
12 program test
13       use wave_struct_def
14       integer :: i, j, k, N
15       TYPE (wavedes)  WDES
16       TYPE (wavedes1)  WDES1
17       INTEGER,POINTER :: OUTPUT(:)
18       N=10
19
20       ALLOCATE(WDES%LMMAXX(N))
21       ALLOCATE(OUTPUT(N))
22
23       do i=1, N
24       WDES%LMMAXX(i) = 1
25       OUTPUT(i) = 0
26       enddo
27       !$OMP TARGET ENTER DATA MAP(TO:WDES)
28       !$OMP TARGET ENTER DATA MAP(TO:WDES%LMMAXX)
29
30       ! use WDES / WDES%LMMAXX in different loops/directives
31
32       WDES1%LMMAXX => WDES%LMMAXX
33
34       !$OMP TARGET ENTER DATA MAP(TO:WDES1)
35       !$OMP TARGET ENTER DATA MAP(TO:WDES1%LMMAXX)
36
37       !$OMP TARGET TEAMS DISTRIBUTE MAP(FROM:OUTPUT)
38       do i=1, N
39       OUTPUT(i) = WDES1%LMMAXX(i)
40       enddo
41       !$OMP END TARGET TEAMS DISTRIBUTE
42
43       do i=1, N
44       write(*,*) "OUTPUT(", i, ")=", OUTPUT(i)
45       enddo
46       !$OMP TARGET EXIT DATA MAP(DELETE:WDES1%LMMAXX)
47       !$OMP TARGET EXIT DATA MAP(DELETE:WDES1)
48
49       !$OMP TARGET EXIT DATA MAP(DELETE:WDES%LMMAXX)
50       !$OMP TARGET EXIT DATA MAP(DELETE:WDES)
51
52  end program test
```

- 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

```
coe62819@cedar004:~/kernel/ticket5/crayticket> ./map_aliased
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from ./map_aliased_orig.f90:27
ACC:       allocate, copy to acc 'wdes' (72 bytes)
ACC: End transfer (to acc 72 bytes, to host 0 bytes)
ACC: Start transfer 3 items from ./map_aliased_orig.f90:28
ACC:       allocate, copy to acc 'wdes%lmmaxx(:)' (40 bytes)
ACC:       present 'wdes' (72 bytes)
ACC:       attach pointer 'wdes%lmmaxx' (72 bytes)
ACC: End transfer (to acc 40 bytes, to host 0 bytes)
ACC: Start transfer 1 items from ./map_aliased_orig.f90:34
ACC:       allocate, copy to acc 'wdes1' (72 bytes)
ACC: End transfer (to acc 72 bytes, to host 0 bytes)
ACC: Start transfer 3 items from ./map_aliased_orig.f90:35
ACC:       present 'wdes1%lmmaxx(:)' (40 bytes)
ACC:       present 'wdes1' (72 bytes)
ACC:       no attach pointer 'wdes1%lmmaxx' (72 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC: Start transfer 2 items from ./map_aliased_orig.f90:37
ACC:       allocate 'output(:)' (40 bytes)
ACC:       present 'wdes1' (72 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC: Execute kernel test_$ck_L37_1 blocks:1 threads:256 async(auto) from ./map_aliased_orig.f90:3
:0:rocdevice.cpp           :2615: 15286944398 us: 9126 : [tid:0x7fc99b2af700] Device::callbackQu
: 0x2b
Aborted
```

Pointer aliasing (alternative methods)

```
1  MODULE wave_struct_def
2  TYPE wavedes
3          INTEGER,POINTER :: LMMAXX(:)
4          END TYPE wavedes
5
6  TYPE wavedes1
7          INTEGER,POINTER :: LMMAXX(:) => NULL()
8          END TYPE wavedes1
9  END MODULE wave_struct_def
10
11
12 program test
13       use wave_struct_def
14       integer :: i, j, k, N
15       TYPE (wavedes)  WDES
16       TYPE (wavedes1)  WDES1
17       INTEGER,POINTER :: OUTPUT(:)
18       N=10
19
20       ALLOCATE(WDES%LMMAXX(N))
21       ALLOCATE(OUTPUT(N))
22
23       do i=1, N
24       WDES%LMMAXX(i) = 1
25       OUTPUT(i) = 0
26       enddo
27       !$OMP TARGET ENTER DATA MAP(TO:WDES)
28       !$OMP TARGET ENTER DATA MAP(TO:WDES%LMMAXX)
29
30       ! use WDES / WDES%LMMAXX in different loops/directives
31
32       !$OMP TARGET
33       WDES1%LMMAXX => WDES%LMMAXX
34       !$OMP END TARGET
35
36 !       !$OMP TARGET ENTER DATA MAP(TO:WDES1)
37 !       !$OMP TARGET ENTER DATA MAP(TO:WDES1%LMMAXX)
38
39       !$OMP TARGET TEAMS DISTRIBUTE MAP(FROM:OUTPUT)
40       do i=1, N
41       OUTPUT(i) = WDES1%LMMAXX(i)
42       enddo
43       !$OMP END TARGET TEAMS DISTRIBUTE
44
45       do i=1, N
46       write(*,*) "OUTPUT(", i, ")=", OUTPUT(i)
47       enddo
48       !$OMP TARGET EXIT DATA MAP(DELETE:WDES1%LMMAXX)
49       !$OMP TARGET EXIT DATA MAP(DELETE:WDES1)
50
51       !$OMP TARGET EXIT DATA MAP(DELETE:WDES%LMMAXX)
52       !$OMP TARGET EXIT DATA MAP(DELETE:WDES)
```

→ Launch a kernel

```
MODULE wave_struct_def
TYPE wavedes
        REAL,POINTER :: LMMAXX(:)
        END TYPE wavedes

TYPE wavedes1
        REAL,POINTER :: LMMAXX(:) => NULL()
        END TYPE wavedes1
END MODULE wave_struct_def


program test
      use wave_struct_def
      !!$omp requires unified_shared_memory
      integer :: i, j, k, N,q
      TYPE (wavedes)  WDES
      TYPE (wavedes1)  WDES1
      REAL,POINTER :: OUTPUT(:)
      N=10

      !do q=1, 1000000
      ALLOCATE(WDES%LMMAXX(N))
      ALLOCATE(OUTPUT(N))

      do i=1, N
      WDES%LMMAXX(i) = 1
      OUTPUT(i) = 0
      enddo
      !$OMP TARGET ENTER DATA MAP(TO:WDES)
      !$OMP TARGET ENTER DATA MAP(TO:WDES%LMMAXX)

      !use WDES / WDES%LMMAXX in different loops/directives

      !$OMP TARGET DATA USE_DEVICE_PTR(WDES)
      WDES1%LMMAXX => WDES%LMMAXX
      !$OMP END TARGET DATA

      !$OMP TARGET TEAMS DISTRIBUTE MAP(FROM:OUTPUT)
      do i=1, N
      OUTPUT(i) = WDES1%LMMAXX(i)
      enddo
      !$OMP END TARGET TEAMS DISTRIBUTE

      do i=1, N
      write(*,*) "OUTPUT(", i, ")=", OUTPUT(i)
      enddo

      !$OMP TARGET EXIT DATA MAP(DELETE:WDES%LMMAXX)
      !$OMP TARGET EXIT DATA MAP(DELETE:WDES)

      deallocate(OUTPUT)
      deallocate(WDES%LMMAXX)
      !enddo

      end program test
```

→ Using target data construct

Pointer mismatch in subroutine calls

```
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()
          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%CPROJ(100,100))
      NSIM=10

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

      DO NP=1, NSIM
      DO I=1, 10
      CALL SETWAV(W,W1(NP),I)

      CALL ECCP(W1(NP))

      ENDDO
      ENDDO

      DO NP=1, NSIM
      CALL DELWAV_R(W1(NP))
      ENDDO
      end program
```

```
SUBROUTINE NEWWAV_R(W1)
   use wave_struct_def
   TYPE (wavefun1), INTENT(INOUT) :: W1
   INTEGER MPLWV

   MPLWV=100

   ALLOCATE(W1%CR(MPLWV))
   !$OMP TARGET ENTER DATA MAP(ALLOC:W1%CR)
   !$OMP TARGET
   W1%CR=(1,1)
   !$OMP END TARGET
END SUBROUTINE

SUBROUTINE DELWAV_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
        TYPE (wavefun1) :: W1
        INTEGER MM
        COMPLEX(8), TARGET :: CE

        CE=0

        !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD REDUCTION(+:CE)
         DO MM =1, 100
         CE=CE+W1%CR(MM)
         ENDDO
         !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD

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

SUBROUTINE SETWAV(W,W1,I)
        use wave_struct_def
        TYPE (wavespin), INTENT(IN) :: W
        TYPE (wavefun1), INTENT(INOUT) :: W1
        INTEGER I,J,NP

        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPTWFP)
        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPROJ)

        W1%CPTWFP=>W%CPTWFP(:,I)
        W1%CPROJ =>W%CPROJ(:,I)

        !$OMP TARGET ENTER DATA MAP(TO:W1%CPTWFP,W1%CPROJ)
END SUBROUTINE
```

```
$./aliasing
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
 ce= (100.,100.)
:0:rocdevice.cpp          :2660: 1637590862517 us:
86531: [tid:0x7fbe82217700]
Device::callbackQueue aborting with error :
HSA_STATUS_ERROR_MEMORY_APERTURE_V
IOLATION: 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()
          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%CPROJ(100,100))
      NSIM=10

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

      DO NP=1, NSIM
      DO I=1, 10
      CALL SETWAV(W,W1(NP),I)

      CALL ECCP(W1,NP)

      ENDDO
      ENDDO

      DO NP=1, NSIM
      CALL DELWAV_R(W1(NP))
      ENDDO
      end program
```

```
SUBROUTINE NEWWAV_R(W1)
   use wave_struct_def
   TYPE (wavefun1), INTENT(INOUT) :: W1
   INTEGER MPLWV

   MPLWV=100

   ALLOCATE(W1%CR(MPLWV))
   !$OMP TARGET ENTER DATA MAP(ALLOC:W1%CR)
   !$OMP TARGET
   W1%CR=(1,1)
   !$OMP END TARGET
END SUBROUTINE

SUBROUTINE DELWAV_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,NP)
        use wave_struct_def
        TYPE (wavefun1) :: W1(10)
        INTEGER MM
        COMPLEX(8), TARGET :: CE

        CE=0

        !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD REDUCTION(+:CE)
         DO MM =1, 100
         CE=CE+W1(NP)%CR(MM)
         ENDDO
         !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD

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

SUBROUTINE SETWAV(W,W1,I)
        use wave_struct_def
        TYPE (wavespin), INTENT(IN) :: W
        TYPE (wavefun1), INTENT(INOUT) :: W1
        INTEGER I,J,NP

        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPTWFP)
        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPROJ)

        W1%CPTWFP=>W%CPTWFP(:,I)
        W1%CPROJ =>W%CPROJ(:,I)

        !$OMP TARGET ENTER DATA MAP(TO:W1%CPTWFP,W1%CPROJ)
END SUBROUTINE
```

Atomic update for complex(8)

Original code

```
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)

!$omp target teams distribute map(tofrom:B) private(k,k2)
      do i=1, M
!$omp parallel do
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
!$omp atomic update
      B(k,k2)%re=B(k,k2)%re+REAL(X)
!$omp atomic update
      B(k,k2)%im=B(k,k2)%im+AIMAG(X)
      enddo
!$omp end parallel do
      enddo
!$omp end target teams distribute

      write(*,*) "B(1,1)%im= ", B(1,1)%im
      do i=1, M
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
      C(k,k2)=C(k,k2)+(1,1)
      enddo
      enddo

      do i=1,51
      do j=1, 41
      if(B(i,j)/=C(i,j)) then
            write(*,*) "error at index (", i, j, ") B= ", B(i,j), "C= ", C(i,j)
      endif
      enddo
      enddo

      end program test
```

Alternative

```
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)

!$omp target teams distribute map(tofrom:B) private(k,k2)
      do i=1, M
!$omp parallel do
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
      call SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(B(k,k2),X)
      enddo
!$omp end parallel do
      enddo
!$omp end target teams distribute

      do i=1, M
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
      C(k,k2)=C(k,k2)+(1,1)
      enddo
      enddo

      do i=1,51
      do j=1,41
      if(B(i,j)/=C(i,j)) then
      write(*,*) "error at index (", i, j, ") B= ", B(i,j), "C= ", C(i,j)
      endif
      enddo
      enddo

      end program test
```

```
SUBROUTINE SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(SPLIT_CMPLX,TO_ADD)
    REAL(8),DIMENSION(2) :: SPLIT_CMPLX
    COMPLEX(8) :: TO_ADD
!$OMP ATOMIC UPDATE
    SPLIT_CMPLX(1)=SPLIT_CMPLX(1)+REAL(TO_ADD) ! real part
!$OMP ATOMIC UPDATE
    SPLIT_CMPLX(2)=SPLIT_CMPLX(2)+AIMAG(TO_ADD) ! imaginary part
  END SUBROUTINE SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX
```

The overhead of subroutine call assuming there is no need for atomic update

```
DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD PRIVATE(ISPIRAL,NI,NP,NT,LMMAXC,INDMAX,LMBASE,NLIIND,IBLOC
DO ITER=0,COUNTER-1
       ISPINOR=TOT_ITER(ITER*3+1)
       NI=TOT_ITER(ITER*3+2)
       NP=TOT_ITER(ITER*3+3)
            NT=NONLR_S%ITYP(NI)

            LMMAXC=NONLR_S%LMMAX(NT)

            INDMAX=NONLR_S%NLIMAX(NI __NOACC_omp_arg(i))

            LMBASE=NONLR_S%LMBASE(NI)+ISPINOR*NONLR_S%LMBASE(NONLR_S%NIONS+1)
            NLIIND=NONLR_S%NLIBASE(NI __NOACC_omp_arg(i))
            ISPIRAL=1; IF (NONLR_S%LSPIRAL) ISPIRAL=ISPINOR+1

            DO IBLOCK=0,INDMAX/BLOCKSIZE
            DO IND=IBLOCK*BLOCKSIZE+1,MIN((IBLOCK+1)*BLOCKSIZE,INDMAX)

               CTMP=0
               DO L=1,LMMAXC
                  CTMP=CTMP+CPROJ(L+LMBASE,NP)*NONLR_S%RPROJ(IND+(L-1)*INDMAX+NLIIND __NOACC_omp_arg(i))
               ENDDO
#ifndef gammareal
               CTMP=CTMP*CONJG(NONLR_S%CRREXP(IND,NI,ISPIRAL __NOACC_omp_arg(i)))
#endif
               IP=NONLR_S%NLI(IND,NT __NOACC_omp_arg(i))+ISPINOR*MPLWV_TMP
               CALL SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(CRACC(IP,NP),CTMP*WDES1%RINPL)
            ENDDO
            ENDDO
         ENDDO
DOOFF !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD
```

Kernel time= 80 ms

```
DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD PRIVATE(ISPIRAL,NI,NP,NT,LMMAXC,INDMAX,LMBASE,NLIIND,IBLOC
DO ITER=0,COUNTER-1
       ISPINOR=TOT_ITER(ITER*3+1)
       NI=TOT_ITER(ITER*3+2)
       NP=TOT_ITER(ITER*3+3)
            NT=NONLR_S%ITYP(NI)

            LMMAXC=NONLR_S%LMMAX(NT)

            INDMAX=NONLR_S%NLIMAX(NI __NOACC_omp_arg(i))

            LMBASE=NONLR_S%LMBASE(NI)+ISPINOR*NONLR_S%LMBASE(NONLR_S%NIONS+1)
            NLIIND=NONLR_S%NLIBASE(NI __NOACC_omp_arg(i))
            ISPIRAL=1; IF (NONLR_S%LSPIRAL) ISPIRAL=ISPINOR+1

            DO IBLOCK=0,INDMAX/BLOCKSIZE
            DO IND=IBLOCK*BLOCKSIZE+1,MIN((IBLOCK+1)*BLOCKSIZE,INDMAX)

               CTMP=0
               DO L=1,LMMAXC
                  CTMP=CTMP+CPROJ(L+LMBASE,NP)*NONLR_S%RPROJ(IND+(L-1)*INDMAX+NLIIND __NOACC_omp_arg(i))
               ENDDO
#ifndef gammareal
               CTMP=CTMP*CONJG(NONLR_S%CRREXP(IND,NI,ISPIRAL __NOACC_omp_arg(i)))
#endif
               IP=NONLR_S%NLI(IND,NT __NOACC_omp_arg(i))+ISPINOR*MPLWV_TMP
               CRACC1(IP,NP)=CRACC1(IP,NP)+CTMP*WDES1%RINPL
               CRACC2(IP,NP)=CRACC2(IP,NP)+CTMP*WDES1%RINPL
            ENDDO
            ENDDO
         ENDDO
DOOFF !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD
```

Kernel time= 22 ms

Declare target

```
PROGRAM reproducer

    IMPLICIT NONE

    INTEGER, PARAMETER :: DP = selected_real_kind(14, 200)
    COMPLEX(DP), ALLOCATABLE :: psi(:,:), ew(:)
    INTEGER :: n, notcnv, nbn, npwx, npol, nvecx, ierr, nbase, npw
    REAL(DP), EXTERNAL :: MYDDOT_VECTOR_GPU

    nbase = 1
    n = 10
    nbn = 2
    notcnv = 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)


    !$omp target teams distribute private(nbn)
    DO n = 1, notcnv
      nbn = nbase + n
      ew(n) = ew(n)  + MYDDOT_VECTOR_GPU( 2*npw, psi(npwx+1,nbn), psi(npwx+1,nbn) )
    END DO
    !$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
ftn -fopenmp -c myddot.f90 -o myddot.o

       !$omp parallel do simd reduction(+:RES)
ftn-7212 ftn: WARNING MYDDOT_VECTOR_GPU, File = myddot.f90, Line = 7
  Variable "res" is used before it is defined.
ftn-7256 ftn: 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_088e5928c3724749216ddb6b2fbbcd2152ed2bb8)
Cray Fortran : Thu Jan 05, 2023  15:58:21
Cray Fortran : Compile time:  0.0472 seconds
Cray Fortran : 13 source lines
Cray Fortran : 0 errors, 2 warnings, 0 other messages, 0 ansi
Cray Fortran : "explain ftn-message number" gives more information about each message.
ftn -fopenmp -c reproducer.f90 -o reproducer.o
ftn -fopenmp myddot.o reproducer.o -o reproducer.x
error: reproducer.f90:28:0: in function reproducer_$ck_L25_1 void (i64, i64, i64, i64, i64, i64): unsupported call to variadic function myddot_vector_gpu_

make: *** [Makefile:8: reproducer] Error 1
```

Declare target (alternative method)

```
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

PROGRAM reproducer

    IMPLICIT NONE

    INTEGER, PARAMETER :: DP = selected_real_kind(14, 200)
    COMPLEX(DP), ALLOCATABLE :: psi(:,:), ew(:)
    INTEGER :: n, notcnv, nbn, npwx, npol, nvecx, ierr, nbase, npw
    REAL(DP), EXTERNAL :: MYDDOT_VECTOR_GPU

    nbase = 1
    n = 10
    nbn = 2
    notcnv = 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)


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


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

END PROGRAM
```

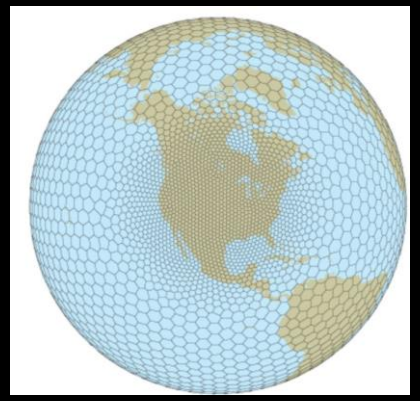
- To get around the error, we can define function in the same file as function call
  - It would be challenging to apply his workaround in the applications with many function/subroutine calls

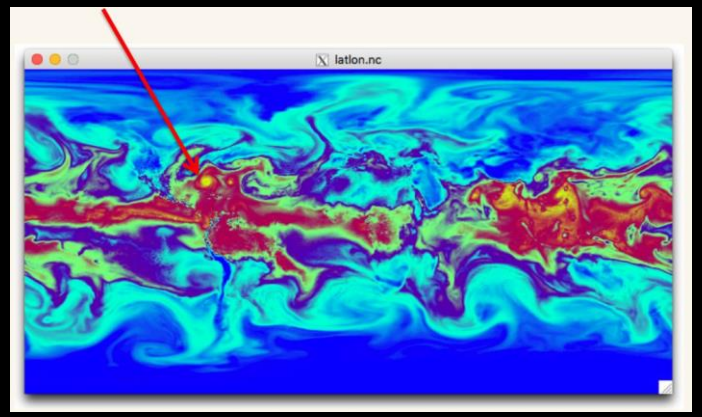
Case study 2 - MPAS

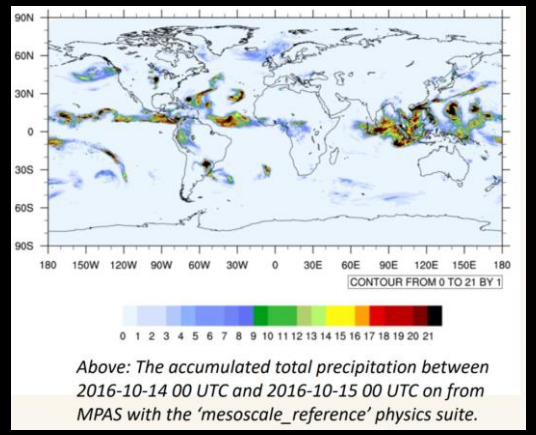
The Model for Prediction Across Scales (MPAS) is a collaborative project for developing atmosphere, ocean and other earth-system simulation components for use in climate, regional climate, and weather studies.

- 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

```
program mpas
    use mpas_subdriver
    use mpas_derived_types, only : core_type, domain_type
    implicit none
    type (core_type), pointer :: corelist => null()
    type (domain_type), pointer :: domain => null()
    call mpas_init(corelist,domain) ! Allocate domain and host arrays
    call mpas_run(domain)
    call mpas_finalize(corelist, domain)
end program mpas
```

```
subroutine mpas_pool_get_array_2d_real_gpu(inPool, key, array, timeLevel)!{{{
    implicit none
    type (mpas_pool_type), intent(in) :: inPool
    character (len=*), intent(in) :: key
    real (kind=RKIND), dimension(:,:), pointer :: array
    integer, intent(in), optional :: timeLevel
    type (field2DReal), pointer :: field
    type (mpas_coupler_type), pointer :: coupler
    nullify(field)
    call mpas_pool_get_field_2d_real(inPool, key, field, timeLevel)
    nullify(array)
    if (associated(field)) then
        coupler => field%block%domain%mpas_cpl
        array => field % array
#ifdef CORE_ATMOSPHERE
        if (coupler % pole_includes(ROLE_INTEGRATE)) then
            !$acc enter data copyin(field%array)
        end if
#endif
    end if
end subroutine mpas_pool_get_array_2d_real_gpu!}}}
```

- 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.

```
subroutine mpas_run(domain)
    ! extract information from domain ..

    do itimestep=1,Ntimestep
        ! Allocate GPU arrays
        call mpas_pool_get_array_gpu(state,'theta_m',gpu_theta_m_1,1)
        call mpas_pool_get_array_gpu(state,'scalars',gpu_scalars_1,1)
        call mpas_pool_get_array_gpu(diag,'pressure_p',gpu_pressure_p)
        call mpas_pool_get_array_gpu(diag,'rtheta_p',gpu_rtheta_p)
        call mpas_pool_get_array_gpu(state, 'w', gpu_w_2, 2)
        call mpas_pool_get_array_gpu(state, 'u',gpu_u_2, 2)
        ! and more...

        ! Compute...

        call mpas_update_gpu_data_on_host(domain)
    enddo
end subroutine
```

```
subroutine mpas_update_gpu_data_on_host(domain)
    ! extract information from domain ...
    call mpas_pool_get_array_gpu(state, 'w', w, 2)
    call mpas_pool_get_array_gpu(state, 'u', u, 2)
    !$acc update host(w, u)

end subroutine mpas_update_gpu_data_on_host
```

Example #1: OPENACC code

```
!$acc parallel vector_length(32)
!$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
!DIR$ IVDEP
!$acc loop vector
        do k=1,nVertLevels
            pgrad = ((rtheta_pp(k,cell2)-rtheta_pp(k,cell1))*invDcEdge(iEdge) )/(.5*(zz(k,cell2)+zz(k,cell1)))
            pgrad = cqu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
            pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
            ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)
        end do
        ! accumulate ru_p for use later in scalar transport
!DIR$ IVDEP
!$acc loop vector
        do k=1,nVertLevels
            ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
        end do
    end if ! end test for block-owned cells
end do ! end loop over edges
!$acc end parallel
```

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

```
1  !$omp target teams distribute
2  !$acc parallel vector_length(32)
3  !$acc loop gang
4  do iEdge=edgeStart,edgeEnd
5      cell1 = cellsOnEdge(1,iEdge)
6      cell2 = cellsOnEdge(2,iEdge)
7      ! update edges for block-owned cells
8      if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
9  !$omp parallel do simd
10 !DIR$ IVDEP
11 !$acc loop vector
12         do k=1,nVertLevels
13             pgrad = ((rtheta_pp(k,cell2)-rtheta_pp(k,cell1))*invDcEdge(iEdge) )/(.5*(zz(k,cell2)+zz(k,cell1)))
14             pgrad = cqu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
15             pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
16             ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)
17         end do
18 !$omp end parallel do simd
19         ! accumulate ru_p for use later in scalar transport
20 !$omp parallel do simd
21 !DIR$ IVDEP
22 !$acc loop vector
23         do k=1,nVertLevels
24             ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
25         end do
26 !$omp end parallel do simd
27     end if ! end test for block-owned cells
28 end do ! end loop over edges
29 !$acc end parallel
30 !$omp end target teams distribute
```

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

```
1  !$omp target teams distribute thread_limit(64) <----------------------------
2  !$acc parallel vector_length(32)
3  !$acc loop gang
4  do iEdge=edgeStart,edgeEnd
5      cell1 = cellsOnEdge(1,iEdge)
6      cell2 = cellsOnEdge(2,iEdge)
7      ! update edges for block-owned cells
8      if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
9  !$omp parallel do simd
10 !DIR$ IVDEP
11 !$acc loop vector
12         do k=1,nVertLevels
13             pgrad = ((rtheta_pp(k,cell2)-rtheta_pp(k,cell1))*invDcEdge(iEdge) )/(.5*(zz(k,cell2)+zz(k,cell1)))
14             pgrad = cqu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
15             pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
16             ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)
17         end do
18 !$omp end parallel do simd
19         ! accumulate ru_p for use later in scalar transport
20 !$omp parallel do simd
21 !DIR$ IVDEP
22 !$acc loop vector
23         do k=1,nVertLevels
24             ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
25         end do
26 !$omp end parallel do simd
27     end if ! end test for block-owned cells
28 end do ! end loop over edges
29 !$acc end parallel
30 !$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

```
1  !$omp target teams distribute collapse(2)
2  !$acc parallel vector_length(32)
3  !$acc loop gang
4  do iEdge=edgeStart,edgeEnd
5  GPUOMP do k=1,nVertLevels
6      cell1 = cellsOnEdge(1,iEdge)
7      cell2 = cellsOnEdge(2,iEdge)
8      ! update edges for block-owned cells
9      if (cell1 <= nCellsSolve .or. cell2 <= nCellsSolve ) then
10 !$omp parallel do simd
11 !DIR$ IVDEP
12 !$acc loop vector
13 GPUACC  do k=1,nVertLevels
14             pgrad = ((rtheta_pp(k,cell2)-rtheta_pp(k,cell1))*invDcEdge(iEdge) )/(.5*(zz(k,cell2)+zz(k,cell1)))
15             pgrad = cqu(k,iEdge)*0.5*c2*(exner(k,cell1)+exner(k,cell2))*pgrad
16             pgrad = pgrad + 0.5*zxu(k,iEdge)*gravity*(rho_pp(k,cell1)+rho_pp(k,cell2))
17             ru_p(k,iEdge) = ru_p(k,iEdge) + dts*(tend_ru(k,iEdge) - (1.0_RKIND - specZoneMaskEdge(iEdge))*pgrad)
18 GPUOMP      ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
19 GPUACC  end do
20 !$omp end parallel do simd
21         ! accumulate ru_p for use later in scalar transport
22 !$omp parallel do simd
23 !DIR$ IVDEP
24 !$acc loop vector
25 GPUACC  do k=1,nVertLevels
26 GPUACC      ruAvg(k,iEdge) = ruAvg(k,iEdge) + ru_p(k,iEdge)
27 GPUACC  end do
28 !$omp end parallel do simd
29     end if ! end test for block-owned cells
30 GPUOMP end do
31 end do ! end loop over edges
32 !$acc end parallel
33 !$omp end target teams distribute
```

Can now use default number of threads (256)

```
1  #ifdef _OPENACC
2  #define GPUACC
3  #define GPUOMP !
4  #else
5  #define GPUACC !
6  #define GPUOMP
7  #endif
```

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

Example #2: OpenACC code

```
1   !$acc parallel vector_length(64)
2   !$acc loop gang private(wduz, tend_wk, eoe_w, we_w)
3   do iEdge=edgeSolveStart,edgeSolveEnd
4   !$acc cache(tend_wk,wduz,eoe_w,we_w)
5       cell1 = cellsOnEdge(1,iEdge)
6       cell2 = cellsOnEdge(2,iEdge)
7   !DIR$ IVDEP
8   !$acc loop vector
9       do k=1,nVertLevels
10          ! compute ...
11      end do
12      ! Compute ...
13  !$acc loop vector shortloop
14      do j = 1,nEdgesOnEdge(iEdge)
15          eoe_w(j) = edgesOnEdge(j,iEdge)
16          we_w(j)  = weightsOnEdge(j,iEdge)
17      end do
```

Caches local arrays into shared memory
No OpenMP equivalent for !$acc cache

```
18  !DIR$ IVDEP
19  !$acc loop vector
20      do k=1,nVertLevels
21          q1 = pv_edge(k,iEdge)
22          q2 = 0.0
23  !$acc loop seq
24          do j = 1,nEdgesOnEdge(iEdge)
25              eoe = eoe_w(j)
26              workpv = 0.5 * (q1 + pv_edge(k,eoe))
27              q2 = q2 + we_w(j) * u(k,eoe) * workpv
28          end do
29          t_w = - rdzw(k)*(wduz(k+1)-wduz(k))
30          tend_u(k,iEdge) = t_w + rho_edge(k,iEdge) * &
31                                  (q2 - (ke(k,cell2) - ke(k,cell1)) *  &
32                                  invDcEdge(iEdge)) - tend_wk(k) * 0.5 * &
33                                  (h_divergence(k,cell1)+h_divergence(k,cell2))
34          end do
35      end do
36  !$acc end parallel
```

Collapsing inner loops not always possible

Example #2: OpenMP® initial port

```
1  !$omp target teams distribute parallel do thread_limit(64)
2  !$acc parallel vector_length(64)
3  !$acc loop gang private(wduz, tend_wk, eoe_w, we_w)
4  do iEdge=edgeSolveStart,edgeSolveEnd
5  !$acc cache(tend_wk,wduz,eoe_w,we_w)
6      cell1 = cellsOnEdge(1,iEdge)
7      cell2 = cellsOnEdge(2,iEdge)
8  !$omp parallel do simd
9  !DIR$ IVDEP
10 !$acc loop vector
11     do k=1,nVertLevels
12         ! compute ...
13     end do
14 !$omp end parallel do simd
15     ! Compute ...
16 !$omp parallel do simd
17 !$acc loop vector shortloop
18     do j = 1,nEdgesOnEdge(iEdge)
19         eoe_w(j) = edgesOnEdge(j,iEdge)
20         we_w(j)  = weightsOnEdge(j,iEdge)
21     end do
22 !$omp end parallel do simd
```

```
23 !DIR$ IVDEP
24 !$omp parallel do simd
25 !$acc loop vector
26     do k=1,nVertLevels
27         q1 = pv_edge(k,iEdge)
28         q2 = 0.0
29 !$acc loop seq
30         do j = 1,nEdgesOnEdge(iEdge)
31             eoe = eoe_w(j)
32             workpv = 0.5 * (q1 + pv_edge(k,eoe))
33             q2 = q2 + we_w(j) * u(k,eoe) * workpv
34         end do
35         t_w = - rdzw(k)*(wduz(k+1)-wduz(k))
36         tend_u(k,iEdge) = t_w + rho_edge(k,iEdge) * &
37                           (q2 - (ke(k,cell2) - ke(k,cell1)) *  &
38                           invDcEdge(iEdge)) - tend_wk(k) * 0.5 * &
39                           (h_divergence(k,cell1)+h_divergence(k,cell2))
40     end do
41 !$omp end parallel do simd
42 end do
43 !$acc end parallel
44 !$omp end target teams distribute
```

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?

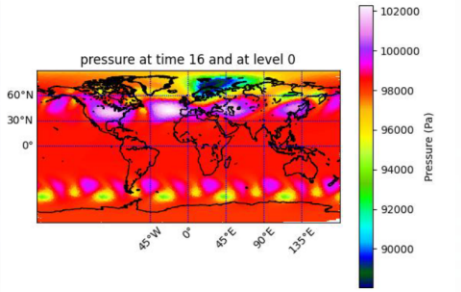
Example #2: OpenMP® optimization – rearranging and splitting loops

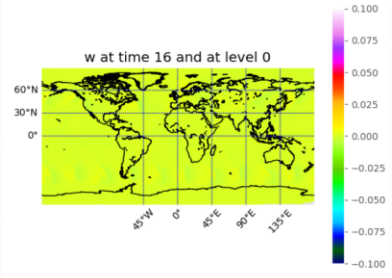
```
1  !$omp target teams distribute parallel do thread_limit(64)
2  !$acc parallel vector_length(64)
3  !$acc loop gang private(wduz, tend_wk, eoe_w, we_w)
4  do iEdge=edgeSolveStart,edgeSolveEnd
5  !$acc cache(tend_wk,wduz,eoe_w,we_w)
6      cell1 = cellsOnEdge(1,iEdge)
7      cell2 = cellsOnEdge(2,iEdge)
8  !$omp parallel do simd
9  !DIR$ IVDEP
10 !$acc loop vector
11     do k=1,nVertLevels
12         ! compute ...
13     end do
14 !$omp end parallel do simd
15     ! Compute ...
16 !$omp parallel do simd
17 !$acc loop vector shortloop
18     do j = 1,nEdgesOnEdge(iEdge)
19         eoe_w(j) = edgesOnEdge(j,iEdge)
20         we_w(j)  = weightsOnEdge(j,iEdge)
21     end do
22 !$omp end parallel do simd
```

```
23 #ifdef _OPENACC
24 ! ...
25 #else
26 !$omp parallel do simd
27     do k=1,nVertLevels
28         t_w = - rdzw(k)*(wduz(k+1)-wduz(k))
29         tend_u(k,iEdge) = t_w + rho_edge(k,iEdge) * &
30                           (-(ke(k,cell2) - ke(k,cell1))* invDcEdge(iEdge)) &
31                           - tend_wk(k) * 0.5 * (h_divergence(k,cell1) &
32                           + h_divergence(k,cell2))
33     end do
34 !$omp end parallel do simd
35     do j = 1,nEdgesOnEdge(iEdge)
36         eoe = eoe_w(j)
37 !$omp parallel do simd
38         do k=1,nVertLevels
39             tend_u(k,iEdge) = tend_u(k,iEdge) + rho_edge(k,iEdge) * we_w(j) &
40                               * u(k,eoe) * 0.5 * (pv_edge(k,iEdge) + pv_edge(k,eoe))
41         end do
42 !$omp end parallel do simd
43     end do
44 #endif
45 end do
46 !$acc end parallel
47 !$omp end target teams distribute
```

Rearrange order of the sequential and parallel do loops. Multiple parallel loops to minimize global memory reads/writes

JW Baroclynic wave – Initial performance





Events	Initial port (secs)	Initial optimization (secs)	Ratio
time integration	**334.99**	**92.94**	**3.60**
atm_rk_integration_setup	1.95	0.54	3.61
atm_compute_moist_coefficients	20.28	0.63	32.19
physics_get_tend	0.57	0.4	1.41
atm_compute_vert_imp_coefs	4.03	1.68	2.40
atm_compute_dyn_tend	128.90	20.68	6.23
small_step_prep	2.83	1.86	1.52
atm_advance_acoustic_step	113.55	31.23	3.64
atm_divergence_damping_3d	8.97	3.1	2.89
atm_recover_large_step_variables	8.68	4.76	1.82
atm_compute_solve_diagnostics	13.66	6.53	2.09
atm_rk_dynamics_substep_finish	0.66	0.49	1.35
atm_rk_reconstruct	2.18	0.86	2.54
atm_rk_summary	2.78	2.77	1.00
mpas update GPU data on host	6.46	6.33	1.02

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

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)

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.

https://www.kernel.org/doc/html/v5.0/vm/hmm.html

OpenMP® programming on systems with HMM

A simple test example:

```
 #pragma omp requires unified_shared_memory
int main(){
     double * X, * Y, *Z;
     size_t N = ( size_t ) 1024*1024* sizeof (
     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

```
DEVID: 0 SGN:2 ConstWGSize:1024 args: 5 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16716B sgpr_count:60 vgpr_count:58 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l50
Call __tgt_rtl_run_target_team_region_async:    10us        0 (        0, 0x00000203cf40, 0x00000203bd20, 0x00000203bd70,        5,       128,       1024,      131072, 0x7ffd4b4b40d0)
Call          __tgt_rtl_synchronize:    2207us        0 (        0, 0x7ffd4b4b40d0)
GPU par : dot loop time = 0.00224113 [s] effective read BW = 0.871489 [GB/s]

DEVID: 0 SGN:2 ConstWGSize:1024 args: 4 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16452B sgpr_count:24 vgpr_count:44 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l57
Call __tgt_rtl_run_target_team_region_async:     7us        0 (        0, 0x00000203d1b0, 0x00000203d170, 0x00000203a940,        4,       128,       1024,      131072, 0x7ffd4b4b40d0)
Call          __tgt_rtl_synchronize:      18us        0 (        0, 0x7ffd4b4b40d0)
GPU par : read loop time = 4.1008e-05 [s] effective read BW = 47.6279 [GB/s]
```

NO UNIFIED MEMORY

```
Call          __tgt_rtl_data_alloc:       0us 0x7ff450408000 (        0,        8, 0x7ffe2056e990)
Call       __tgt_rtl_data_submit_async:    42us        0 (        0, 0x7ff450408000, 0x7ffe2056e990,        8, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_alloc:       0us 0x7ff450409000 (        0,  1048576, 0x0000025f7660)
Call       __tgt_rtl_data_submit_async:    62us        0 (        0, 0x7ff450409000, 0x0000025f7660,  1048576, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_alloc:       1us 0x7ff440200000 (        0,  1048576, 0x0000026f7670)
Call       __tgt_rtl_data_submit_async:    84us        0 (        0, 0x7ff440200000, 0x0000026f7670,  1048576, 0x7ffe2056e8c0)
DEVID: 0 SGN:2 ConstWGSize:1024 args: 5 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16716B sgpr_count:60 vgpr_count:58 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l50
Call __tgt_rtl_run_target_team_region_async:    13us        0 (        0, 0x00000285bf10, 0x0000028816c0, 0x000002881710,        5,       128,       1024,      131072, 0x7ffe2056e8c0)
Call       __tgt_rtl_data_retrieve_async:   517us        0 (        0, 0x7ffe2056e990, 0x7ff450408000,        8, 0x7ffe2056e8c0)
Call          __tgt_rtl_synchronize:   501us        0 (        0, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_delete:     1us        0 (        0, 0x7ff440200000)
Call          __tgt_rtl_data_delete:     0us        0 (        0, 0x7ff450409000)
Call          __tgt_rtl_data_delete:     0us        0 (        0, 0x7ff450408000)
GPU par : dot loop time = 0.00127697 [s] effective read BW = 1.5295 [GB/s]
Call          __tgt_rtl_data_alloc:       0us 0x7ff450408000 (        0,  1048576, 0x0000025f7660)
Call       __tgt_rtl_data_submit_async:    39us        0 (        0, 0x7ff450408000, 0x0000025f7660,  1048576, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_alloc:       0us 0x7ff440200000 (        0,  1048576, 0x0000026f7670)
Call       __tgt_rtl_data_submit_async:    38us        0 (        0, 0x7ff440200000, 0x0000026f7670,  1048576, 0x7ffe2056e8c0)
DEVID: 0 SGN:2 ConstWGSize:1024 args: 4 teamsXthrds:( 128X1024) reqd:( 128X1024) lds_usage:16452B sgpr_count:24 vgpr_count:44 sgpr_spill_count:0 vgpr_spill_count:0 tripcount:131072 rpc:0
n:__omp_offloading_10304_ac24ca_main_l57
Call __tgt_rtl_run_target_team_region_async:     8us        0 (        0, 0x00000285c180, 0x00000285ad20, 0x00000285ca60,        4,       128,       1024,      131072, 0x7ffe2056e8c0)
Call          __tgt_rtl_synchronize:   400us        0 (        0, 0x7ffe2056e8c0)
Call          __tgt_rtl_data_delete:     1us        0 (        0, 0x7ff440200000)
Call          __tgt_rtl_data_delete:     0us        0 (        0, 0x7ff450408000)
GPU par : read loop time = 0.00051403 [s] effective read BW = 3.79963 [GB/s]
```

OpenMP® offloading with HMM – OpenFOAM® case-study

```
template<class T>
Foam::List<T>::List(const Foam::one, const T& val)
:
    UList<T>(new T[1], 1)
{
    this->v_[0] = val;
}
```

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.

```
void Foam::List<T>::doResize(const label len)
{
    if (len == this->size_)
    {
        return;
    }
    if (len > 0)
    {
        // With sign-check to avoid spurious -Walloc-size-larger-than

        size_t alignement = 16;
        size_t bytes_needed = sizeof(T)*len;
        if (bytes_needed > 2*128){ //LG1 AMD
           alignement = 512;       //LG1 AMD
        }
        T* nv = new (std::align_val_t( alignement)) T[len]; //LG1 AMD
        //if (bytes_needed > 2*128){   //LG1 AMD
        //   #pragma omp target enter data map(to:nv[0:len]) //LG1 AMD
       // } //LG1 AMD
```

OpenFOAM® initial porting with HMM - choosing the executing device and expressing parallelism

```
//LG using OpenMP offloading and HMM
#include <omp.h>
#pragma omp requires unified_shared_memory
```

Notify OpenMP run-time about intent to use unified virtual memory. Not required in openmp spec 5.2

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

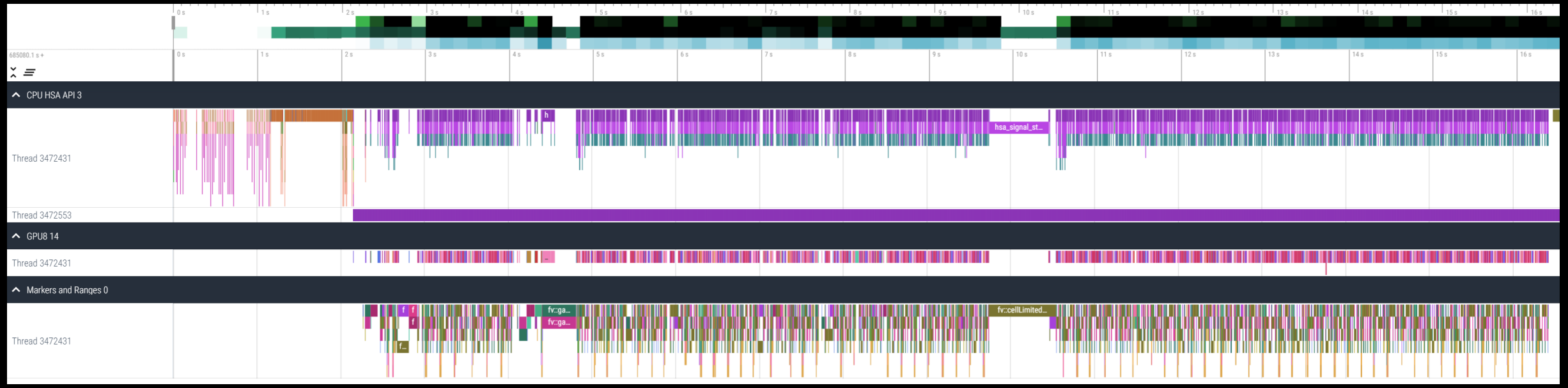
Compiler will produce a Host and a Device code, apply parallelization and offload computation to the Device

Note – no need to allocate memory on the device and copy data between the Host and the Device

```
#pragma omp target teams distribute parallel for //map(tofrom:ApsiPtr[0:nCells])
for (label face=0; face<nFaces; face++)
{
    #pragma omp atomic hint(AMD_fast_fp_atomics)
    ApsiPtr[uPtr[face]] += lowerPtr[face]*psiPtr[lPtr[face]];
    #pragma omp atomic hint(AMD_fast_fp_atomics)
    ApsiPtr[lPtr[face]] += upperPtr[face]*psiPtr[uPtr[face]];
}
```

By default, all memory is fine-grained, hence atomics are system scope atomics. Our GPUs can not handle system scope atomics in a performing way …
by applying the “map” clause we can force memory to be coarse grained (==additional expense)

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.


AMD