Developing Fortran Applications: HIPFort, OpenMP®, and OpenACC

Bob Robey and Brian Cornille
EuroCC National Compentence Centre Sweden (ENCCS)
Nov 29th, 2022

AMD
together we advance_

Authors and Contributors

Brian Cornille

Bob Robey

Mahdieh Ghazimirsaeed

Justin Chang

Thanks to all the AMD contributors for their work on creating these materials.

Agenda

1. Fortran Pathways
   a. Hipify – Fortran with separate CUDA routines
   b. HIPFort – a native HIP solution
   c. Using OpenMP® offloading: a directive-based approach
   d. OpenACC: alternative, but more limited option

Hipify

Hipify

- In this case, we have CUDA code that is called from a Fortran code.
    - Difficulties with calling C routines from Fortran have already been taken care of

- Hipfiy and hipify-clang can be used on separate CUDA C/C++ files
    - This process has already been covered in the HIP and hipify talks
- Compile resulting HIP code with hipcc
- Compile Fortran code with Fortran compiler
- Link with hipcc
    - Standard issues with cross-language links

HIPFort

HIPFort

- A native GPU language solution is desired for cases with
  - CUDA Fortran conversion
  - Pure Fortran code
- HIP functions are callable from C, using `extern C`, so they can be called directly from Fortran
- The strategy here is:
  - **Manually port** CUDA Fortran code to HIP kernels in C-like syntax
  - Wrap the kernel launch in a C function
  - Call the C function from Fortran through Fortran’s ISO_C_binding.
  - Fortran 2003 is required. An improved interface is available with Fortran 2008.
  - With HIP, resulting code can run on both AMD and Nvidia GPUs
  - ROCm™ interfaces will only run on AMD GPUs

HIPFort -- installation

- HIPFort is part of the ROCm™ software package
  - HIPFort is installed as part of the meta-packages starting with ROCM-5.4.0
  - Check to see if it is installed with your ROCm packages – check for /opt/rocm<-version>/bin/hipfc
  - May need to be specifically installed with a package install command before 5.4.0
  - PATH should include /opt/rocm<-version>/bin/hipfc
  - INCLUDE_PATH should include /opt/rocm<-version>/include/hipfort
  - LD_LIBRARY_PATH should include /opt/rocm<-version>/libexe/hipfort
  - Sample Makefile.hipfort at /opt/rocm<-version>/share/hipfort/Makefile.hipfort
- If need to do a user install
  - git clone https://github.com/ROCmSoftwarePlatform/hipfort
  - Add the hipfort/bin location to your path

CUDA Fortran -> Fortran + HIP C/C++ (I)

- There is no HIP equivalent to CUDA Fortran
- But HIP functions are callable from C, using `extern C`, so they can be called directly from Fortran
- The strategy here is:
  - **Manually port** CUDA Fortran code to HIP kernels in C-like syntax
  - Wrap the kernel launch in a C function
  - Call the C function from Fortran through Fortran's ISO_C_binding. It requires either Fortran 2003 or a simpler version with Fortran 2008.
- This strategy should be usable by Fortran users since it is standard conforming Fortran
- ROCm™ has an interface layer for libraires, hipFort, which provides the wrapped bindings for use in Fortran
  - https://github.com/ROCmSoftwarePlatform/hipfort

More explanation -- example of hipLaunchKernelGGL wrapper

```
extern "C" {
  void launch(double **dout, double **da, double **db, int N) {
    hipLaunchKernelGGL((vector_add), dim3(320), dim3(256), 0, 0, *dout, *da,
  *db, N);
  }
}
  interface
     subroutine launch(out,a,b,N) bind(c)
       use iso_c_binding
       implicit none
       type(c_ptr) :: a, b, out
       integer, value :: N
     end subroutine
  end interface
```

Example

Install HIPFort

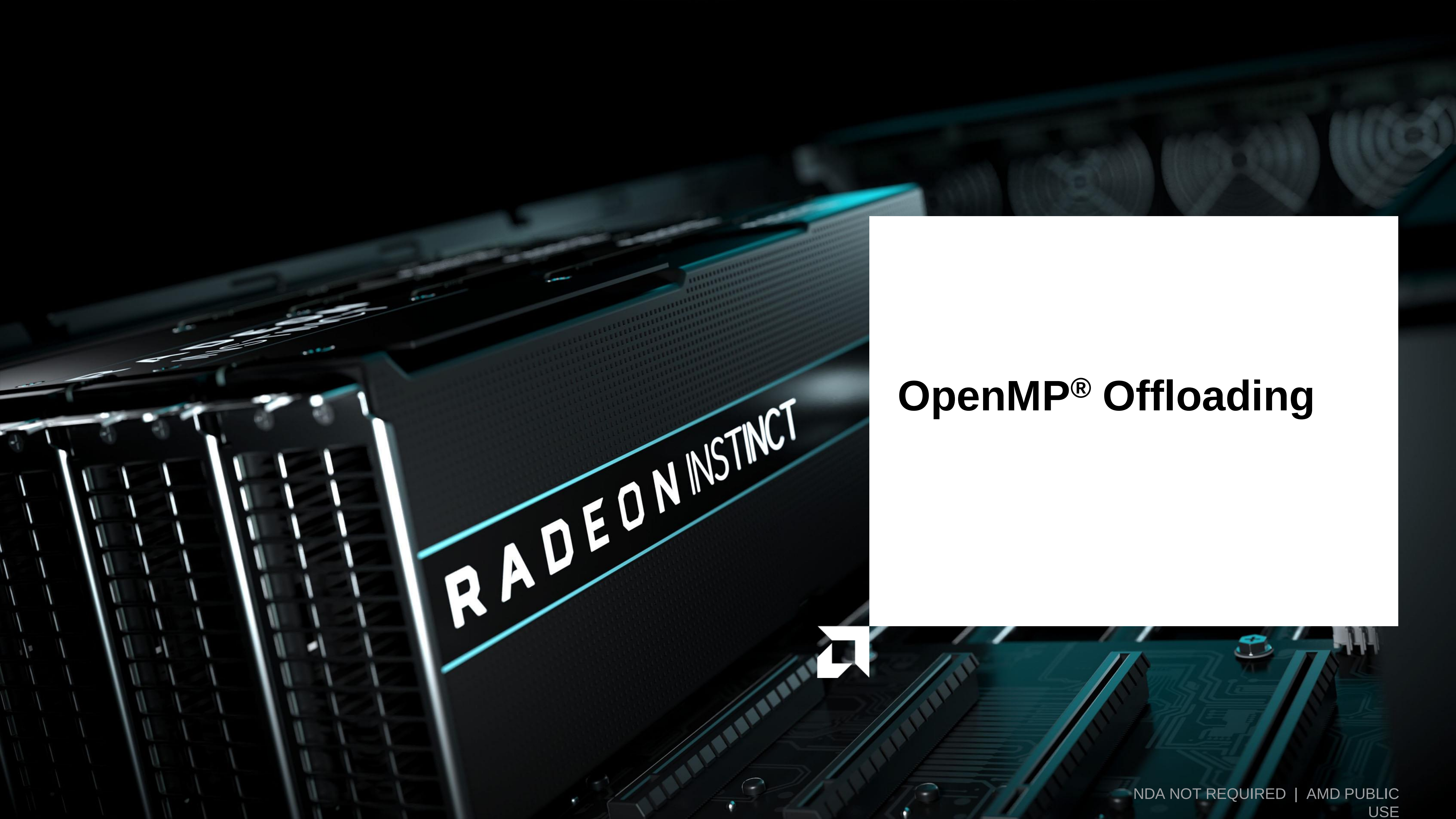
- `export HIPFORT_INSTALL_DIR=`pwd`/hipfort`
- `git clone https://github.com/ROCmSoftwarePlatform/hipfort hipfort-source`
- `mkdir hipfort-build; cd hipfort-build`
- `cmake -DHIPFORT_INSTALL_DIR=${HIPFORT_INSTALL_DIR} ../hipfort-source`
- `make install`
- `export PATH=${HIPFORT_INSTALL_DIR}/bin:$PATH`

Try a test problem

- ``ROCM_GPU=`rocminfo |grep -m 1 -E gfx[^0]{1} | sed -e 's/ *Name: *//'` ``
- `cd ../hipfort-source/test/f2003/vecadd`
- `hipfc -v --offload-arch=${ROCM_GPU} hip_implementation.cpp main.f03`
- `./a.out`
- `cd ../../f2008/vecadd`
- `hipfc -v --offload-arch=${ROCM_GPU} hip_implementation.cpp main.f03`
- `./a.out`

Other Resources

- Github repository -- https://github.com/ROCmSoftwarePlatform/hipfort
- Lunch & Learn: Joe Schoonover: Porting multi-GPU SELF Fluids code to HIPFort
  - Part of the AMD “Lunch & Learn” series
  - https://www.youtube.com/watch?v=RGDmu29T4ik
- FortranCon2021: HIPFort: Present and Future Directions for Portable GPU Programming in Fortran
  - Alessandro Fanfarillo, AMD staff
  - https://www.youtube.com/watch?v=tunH_GUeiPg



OpenMP® Offloading

OpenMP® Offload GPU Support

- ROCm™ and AOMP
  - ROCm supports both HIP and OpenMP
  - AOMP: the AMD OpenMP research compiler, it is used to prototype the new OpenMP features for ROCm
    - Released version of AOMP is at /opt/rocm<-version>/llvm/bin in clang and flang compiler.

- Pre-release version of AOMP is at https://github.com/ROCm-Developer-Tools/aomp. This version, which is undergoing testing for inclusion in ROCm, may have more features, but may also have some bugs.

- GNU compilers:
  - Provide OpenMP and OpenACC offloading support for AMD GPUs
  - GCC 11: Supports AMD GCN gfx908 (MI100)
  - GCC 13: Supports AMD GCN gfx90a (MI200 series)

OpenMP® Offload GPU Support (continued)

- Siemens® Compilers (Sourcery CodeBench Lite – C/C++/Fortran)
  - Siemen’s free GCC-based compilers
  - Supports all GCC 11 features, enriched by OpenMP features from GCC’s development branch and AMD GCN improvements such as support for offloading debugging.
  - https://sourcery.sw.siemens.com/GNUToolchain/release3586
  - wget https://sourcery.sw.siemens.com/GNUToolchain/package16406/public/x86_64-none-linux-gnu/sourceryg++-2022.09-6-x86_64-none-linux-gnu-x86_64-linux-gnu.bin
  - The changes introduced in the Siemen's compiler are being upstreamed into GCC.

List of OpenMP Compilers & Tools : https://www.openmp.org/resources/openmp-compilers-tools/

Compilers for AMD/HPE GPU Programming

- If you are on an AMD/HPE HPC system, there are additional options

- Cray Compilers (HPE compilers)
  - Provide offloading support to AMD GPUs (OpenMP®, HIP, OpenACC)

- Note that the Cray Fortran has their original OpenMP® and OpenACC implementations
- C/C++ is based on LLVM™ and has support for OpenMP® and OpenACC through LLVM

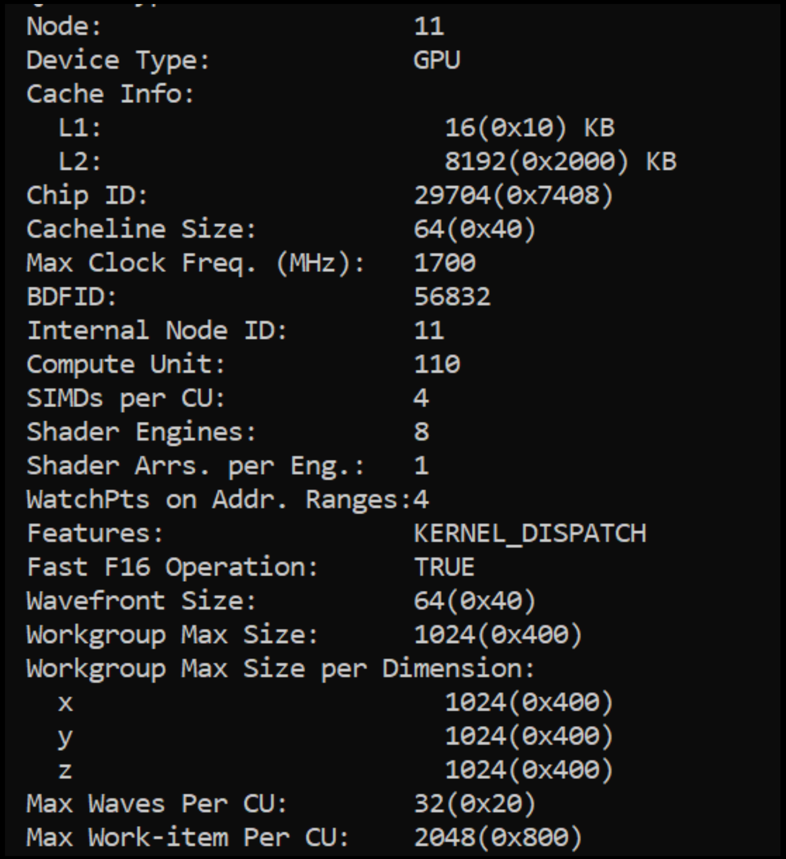
Understanding the hardware options

- ***rocminfo***
  - 110 CUs
  - Wavefront of size 64
  - 4 SIMDs per CU

Options for !omp teams target

- num_teams(220): Multiple number of workgroups with regards the compute units
- thread_limit(256): Threads per workgroup

- Thread limit is multiple of 64
- Teams * thread_limit should be multiple or a divisor of the trip count of a loop



```
Node:                    11
Device Type:             GPU
Cache Info:
  L1:                      16(0x10) KB
  L2:                      8192(0x2000) KB
Chip ID:                 29704(0x7408)
Cacheline Size:          64(0x40)
Max Clock Freq. (MHz):   1700
BDFID:                   56832
Internal Node ID:        11
Compute Unit:            110
SIMDs per CU:            4
Shader Engines:          8
Shader Arrs. per Eng.:   1
WatchPts on Addr. Ranges:4
Features:                KERNEL_DISPATCH
Fast F16 Operation:      TRUE
Wavefront Size:          64(0x40)
Workgroup Max Size:      1024(0x400)
Workgroup Max Size per Dimension:
  x                        1024(0x400)
  y                        1024(0x400)
  z                        1024(0x400)
Max Waves Per CU:        32(0x20)
Max Work-item Per CU:    2048(0x800)
```

Examples -- Fortran vecadd with OpenMP®

```
program main
    integer :: i, n = 100000
    real(8),dimension(:),allocatable :: a, b, c
    real(8) :: sum
    allocate(a(n), b(n), c(n))
    do i=1,n
        a(i) = sin(dble(i)*1.0d0)*sin(dble(i)*1.0d0)
        b(i) = cos(dble(i)*1.0d0)*cos(dble(i)*1.0d0)
    enddo
    !$omp target teams distribute parallel do simd map(to: a(1:n),b(1:n)) map(from: c(1:n))
    do i=1,n
        c(i) = a(i) + b(i)
    enddo
    sum = 0.0d0
    do i=1,n
        sum = sum +  c(i)
    enddo
    sum = sum/dble(n)
    write(*,'("Final result: ",f10.6)') sum
    deallocate(a, b, c)
end program
```

Examples -- Fortran vecadd with OpenMP® -- environment

```
module load aomp
export FC=${AOMP}/bin/flang
```

The makefile uses the ${FC} environment variable so that different Fortran compilers can be used

The ROCm™ module may need to be loaded for the calculation to be able to run on the GPU.

If there is no module, this is what is necessary to set.

Note that there is a version of AOMP installed at /opt/rocm<-version>/llvm/bin

```
export AOMP=<path_to_aomp install>
export PATH=${AOMP}/bin:${PATH}
export FC=${AOMP}/bin/flang
```

For more verbose debugging output during run

```
export LIBOMPTARGET_KERNEL_TRACE=1
export LIBOMPTARGET_INFO=$((0x20 | 0x02 | 0x01 | 0x10))
```

Examples -- Fortran vecadd with OpenMP® -- Makefile

```
default: vecadd
all: vecadd

ROCM_GPU ?= $(strip $(shell rocminfo |grep -m 1 -E gfx[^0]{1} | sed -e 's/ *Name: *//'))

ifeq ($(notdir $(FC)), flang)
  OPENMP_FLAGS = -fopenmp --offload-arch=$(ROCM_GPU)
  FREE_FORM_FLAG = -Mfreeform
else ifeq ($(notdir $(FC)), amdflang)
  OPENMP_FLAGS = -fopenmp --offload-arch=$(ROCM_GPU)
  FREE_FORM_FLAG = -Mfreeform
else ifeq ($(notdir $(FC)), ftn)
  OPENMP_FLAGS = -homp #the craype-accel-amd-gfx* module sets the architecture
  FREE_FORM_FLAG = -ffree
else
  OPENMP_FLAGS = -fopenmp -foffload=-march=${ROCM_GPU} -fopt-info-optimized-omp
  FREE_FORM_FLAG = -ffree-form
endif

FFLAGS = -g -O3 ${FREE_FORM_FLAG} ${OPENMP_FLAGS}
LDFLAGS = ${OPENMP_FLAGS}

vecadd: vecadd.o
    $(FC) $(LDFLAGS) $^ -o $@

clean:
    rm -f *.o vecadd *.mod
```

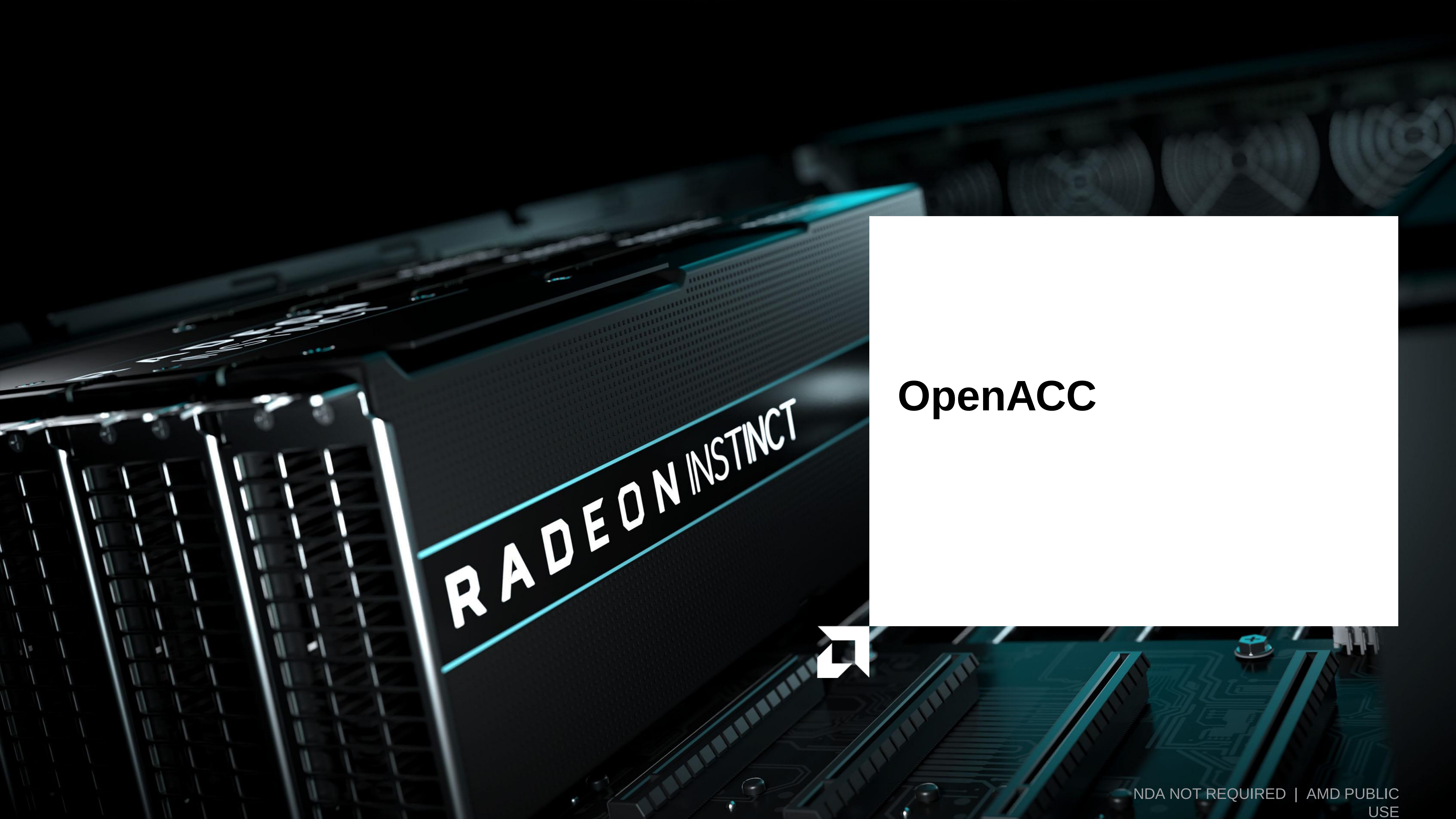
Summary of OpenMP® offloading across AMD compilers

- For AOMP LLVM™ compiler:
  - Compile succeeded, ran on the GPU
- For GCC compiler:
  - Compile succeeded, did not run on the GPU
- For Siemens® GCC compiler:
  - Compile succeeded, ran on the GPU
- For HPE compiler:
  - Compile succeeded, ran on the GPU

Note that the GCC compiler is not built to run the calculations on the AMD GPU and just ran on the CPU. The other three compilers successfully compiled and ran the calculation on the AMD GPU.

Exercises:

- Try modifying the program to put the initialization of the arrays on the GPU
- Test your own OpenMP Fortran application and report any issues with any of these compilers



OpenACC

OpenACC compilers

- OpenMP is the primary directive-based language for AMD
- But compilers based on GCC can be set up with OpenACC support

- Siemen's® sourcery compiler is one option

- Cray Fortran compilers have support for OpenACC version 2.6 + a little???

- LLVM™ based compilers are focusing on OpenMP but have said they will support an OpenACC to OpenMP® translation

Examples -- Fortran vecadd with OpenACC

```
program main
    integer :: i, n = 100000
    real(8),dimension(:),allocatable :: a, b, c
    real(8) :: sum
    allocate(a(n), b(n), c(n))
    do i=1,n
        a(i) = sin(dble(i)*1.0d0)*sin(dble(i)*1.0d0)
        b(i) = cos(dble(i)*1.0d0)*cos(dble(i)*1.0d0)
    enddo
    !$acc parallel loop copyin(a(1:n),b(1:n)), copyout(c(1:n))
    do i=1,n
        c(i) = a(i) + b(i)
    enddo
    sum = 0.0d0
    do i=1,n
        sum = sum +  c(i)
    enddo
    sum = sum/dble(n)
    write(*,'("Final result: ",f10.6)') sum
    deallocate(a, b, c)
end program
```

Only change from OpenMP version

Examples -- Fortran vecadd with OpenACC -- environment

```
module load rocm sourceryg++
export FC=<path-to-siemens>/bin/x86_64-none-linux-gnu-gfortran
```

The makefile uses the ${FC} environment variable so that different Fortran compilers can be used

The ROCm™ module may need to be loaded for the calculation to be able to run on the GPU.

If there is no module, this is what is necessary to set.

```
export PATH=<path-to-siemens>/bin:${PATH}
export INCLUDE=<path-to-siemens>/include:${INCLUDE}
export LD_LIBRARY_PATH=<path-to-siemens>/lib64:/opt/rocm<-version>/lib:${LD_LIBRARY_PATH}
export MANPATH=<path-to-siemens>/bin:${MANPATH}
export FC=<path-to-siemens>/bin/x86_64-none-linux-gnu-gfortran
```

Yes, that is really the compiler name. We've soft linked it to srcy-gfortran for ease of use.

For more verbose debugging output during run

```
export GCN_SUPPRESS_HOST_FALLBACK=true
export GCN_DEBUG=1
```

Examples -- Fortran vecadd with OpenACC -- Makefile

```
default: vecadd
all: vecadd

ROCM_GPU ?= $(strip $(shell rocminfo |grep -m 1 -E gfx[^0]{1} | sed -e 's/ *Name: *//'))
UNAMEP = $(shell uname -p)
ROCM_CPUTARGET = $(UNAMEP)-pc-linux-gnu
ROCM_GPUTARGET ?= amdgcn-amd-amdhsa

ifeq ($(notdir $(FC)), ftn)
  OPENMP_FLAGS = -hacc #the craype-accel-amd-gfx* module sets the architecture
  FREE_FORM_FLAG = -ffree
else
  OPENACC_FLAGS = -fopenacc -foffload=-march=${ROCM_GPU} -fopt-info-optimized-omp
  FREE_FORM_FLAG = -Mfreeform
endif

FFLAGS = -g -O3 ${FREE_FORM_FLAG}  ${OPENACC_FLAGS}
LDFLAGS = ${OPENACC_FLAGS}

vecadd: vecadd.o
    $(FC) $(LDFLAGS) $^ -o $@

clean:
    rm -f *.o vecadd *.mod
```

Summary of OpenACC across AMD compilers

- For Siemens® GCC compiler:
  - Compile succeeded, ran on the GPU
- For HPE compiler:
  - Compile succeeded, ran on the GPU

Only the Siemens® GCC and HPE compilers work for the OpenACC code for AMD GPUs

Using CRAY_ACC_DEBUG=[1,2,3] can help expose what is happening with the application while running

- –hlist=aimd and –hmsgs will give more detail during the compilation

Exercises:

- Try modifying the program to put the initialization of the arrays on the GPU
- Test your own OpenACC Fortran application and report any issues with any of these compilers

Summary

OpenMP® offloading and OpenACC

- Many features are still being added to Fortran compilers
- Use the latest compiler version
- Expect features to be added with every release
- HPE Fortran compilers are more mature and may be the best choice if they are available, especially in the short term
- OpenMP is getting stronger development support
- May want to transition from OpenACC to OpenMP in the longer term
- Please report any compiler issues so that they can continue to be improved

Some common error reports

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

$\searrow$

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 source.f90
```

$\searrow$

Data is mapped to device but is not deleted/released!

Thank you!

Disclaimer

AMD