



# HIP/OpenMP<sup>®</sup> Interoperability

Presenter: Johanna Potyka



# HIP and OpenMP<sup>®</sup> Interoperability

OpenMP supports the following interactions:

- Calling low-level HIP kernels from OpenMP application code

- Calling HIP/ROCm libraries (rocBLAS, rocFFT, etc.) from OpenMP application code

- Calling OpenMP kernels from low-level HIP application code (not shown here)

# Why use both HIP and OpenMP®?

- OpenMP is a higher-level and more portable language
- Reduces code maintenance
- Fully compatible with Fortran (without iso C bindings)

## Best practice:

- Use OpenMP for straight-forward loops
- Use libraries where possible
- Use HIP in more complex coding and where performance matters

# OpenMP® Calling HIP Routine: SAXPY Example

compute\_1  
&  
compute\_2  
run **on CPU**  
and modify x  
and y!

```
int main(int argc, char* argv[])
{
    int n = 1024;    // use 1024 for our example
    float a = 2.0f; // use 2.0f for our example
    float *x = new float[n];
    float *y = new float[n];

    // allocate the device memory
    #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]) // update x and y on the target
        saxpy(n, a, x, y); // compute a * x[i] + y[i] in parallel
    }
    compute_3(n, y);
}
```

Allocate device memory for x  
and y, and specify directions  
of data transfers

Let's assume that we want to  
implement the saxpy() kernel in  
a low-level language.

# OpenMP® Calling HIP Routine: SAXPY Example

```

int main(int argc, char* argv[])
{
    int n = 1024;    // use 1024 for our example
    float a = 2.0f; // use 2.0f for our example
    float *x = new float[n];
    float *y = new float[n];

    // allocate the device memory
    #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])
        saxpy(n, a, x, y); // compute a * x[i]
    }
    compute_3(n, y);
}

```

OpenMP version should be translated to HIP!

```

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

```

# OpenMP® Calling HIP: HIP Kernel for SAXPY

A HIP version of the SAXPY kernel:

```
__global__ void saxpy_kernel(int n, float a, float * x, float * y) {  
    int i = threadIdx.x + blockIdx.x * blockDim.x;  
    y[i] = a * x[i] + y[i];  
}  
void saxpy_hip(int n, float a, float * x, float * y) {  
    assert(n % 256 == 0);  
    saxpy_kernel<<<n/256,256,0,NULL>>>(n, a, x, y);  
    int ret=hipDeviceSynchronize();  
}
```

These are device pointers!

We need a way to translate the host pointer that was mapped by OpenMP directives and retrieve the associated device pointer.

# OpenMP® Calling HIP: Putting it Together

```

__global__ void saxpy_kernel(int n, float a, float * x, float * y) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    y[i] = a * x[i] + y[i];
    // for debug
}
void saxpy_hip(int n, float a, float * x, float * y) {
    assert(n % 256 == 0);
    saxpy_kernel<<<n/256,256,0,NULL>>>(n, a, x, y);
    int ret=hipDeviceSynchronize();
}

```

Translation unit 1

hipcc

```

int main(int argc, char* argv[])
{

```

```

    int n = 1024;    // use 1024 for our example
    float a = 2.0f; // use 2.0f for our example
    float *x = new float[n];
    float *y = new float[n];

```

Creating mapping table with host  
pointer and device pointer -->  
Uses host pointer

```

// allocate the device memory
#pragma omp target data map(to:x[0:count]) map(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)

```

Translation unit 2

clang

```

    {
        saxpy_hip(n, a, x, y); // x[i] + y[i] parallel // mapping table: x:[0xabcd,0xef12], x = 0xef12
    }
}
compute_3(n, y);
}

```

Queries mapping table and finds device  
pointer associated with host array

# Makefile example

```
EXECUTABLE = ./saxpy
all: $(EXECUTABLE) test
```

```
.PHONY: test
```

```
OBJECTS = saxpy_openmp.o saxpy_hip.o
```

```
CXXFLAGS = -g -O2 -fPIC -D__HOST_CODE__
HIPCC_FLAGS = -g -O2 -DNDEBUG -fPIC -I${ROCM_PATH}/include
```

```
HIPCC ?= hipcc
```

```
HIPCC_FLAGS += -munsafe-fp-atomics -D__DEVICE_CODE__
LDLFLAGS = -L${ROCM_PATH}/lib -lamdhip64
```

```
saxpy_hip.o: saxpy_hip.hip
$(HIPCC) $(HIPCC_FLAGS) -c $^ -o $@
```

```
$(EXECUTABLE): $(OBJECTS)
$(CXX) $(OBJECTS) $(LDLFLAGS) -o $@
```

HIP and OpenMP® code parts in **different files** (compilation units) compiled with **different compilers**

# Separate compilation files

- It is important to have HIP code in a **separate file** from OpenMP<sup>®</sup> code.
  - Compile hip code with hipcc
  - Compile OpenMP code with amdclang
- Trick to combine HIP and OpenMP<sup>®</sup> code in a single file
  - use preprocessor directives and copy source into a separate file before compiling
  - example:

```
git clone https://github.com/amd/HPCTrainingExamples
cd HPCTrainingExamples/HIP-OpenMP/CXX/daxpy
module load rocm
module load amdclang
export HSA_XNACK=1
make
```

# Other examples

- Checkout git clone <https://github.com/amd/HPCTrainingExamples>
- cd HPCTrainingExamples/HIP-OpenMP/CXX
- saxpy\_openmp\_offload – **original saxpy openmp offload code**

```
cd saxpy_openmp_offload
module load amdclang
make
```

  - Works with most C++ compilers supporting openmp offload. Try others
- saxpy\_openmp\_hip – **example with HIP kernel in separate file**

```
cd saxpy_openmp_hip
module load rocml
module load amdclang
export HSA_XNACK=1
make
```

# OpenMP® Calling HIP: APU

```

int main(int argc, char* argv[])
{
    int n = 1024;    // use 1024 for our example
    float a = 2.0f; // use 2.0f for our example
    float *x = new float[n];
    float *y = new float[n];

    // allocate the device memory
    #pragma omp target data map(to:x[0:count]) map(tofrom:y[0:count])
    {
        compute_1(n, x); // Just a single memory pointer 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); // x[i] + y[i] // Just a single pointer x = 0xabcd
        }
    }
    compute_3(n, y);
}

```

No longer need any of the memory handling directives

# APU Example

```
module load rocm
module load amdclang
export HSA_XNACK=1
git clone https://github.com/amd/HPCTrainingExamples
cd HPCTrainingExamples/HIP-OpenMP/CXX/saxpy_APU
make
```

- This example is written for the MI300A. Does it run on other MI200 and MI300 series GPUs? Because they support the APU programming model, it should run on them as well, but performance will be different.

# OpenMP® Calling HIP: Fortran and DGEMM

**FORTRAN to HIP interface**

```

subroutine example
  use rocm_interface
  use iso_c_binding
  implicit none
  real(8),allocatable,target,dimension(:,:) :: a, b, c
  type(c_ptr) :: rocblas_handle
  ...

  allocate(da(M,N),db(N,K),dc(M,K))
  call init_matrices(da,db,dc,M,N,K) ! Initialize matrices
  call init_rocblas(rocblas_handle) ! Initialize rocBLAS
  ...

  !$OMP target enter data map(to:a,b,c)
  !$OMP target data use_device_addr(a,b,c)
  call omp_dgemm(rocblas_handle,modea,modeb,M,N,K,alpha,&
    c_loc(a),lda,c_loc(b),ldb,beta,c_loc(c),ldc)
  !$OMP end target data
  !$OMP target update from(c)
  !$OMP target exit data map(delete:a,b,c)
  ...
end subroutine example

```

```

module rocm_interface
interface
  subroutine init_rocblas(handle) bind(C)
    use iso_c_binding
    type(c_ptr) :: handle
  end subroutine init_rocblas
  subroutine omp_dgemm(handle,ma,mb,m,n,k,alpha, &
    a,lda,b,ldb,beta,c,ldc) bind(C)
    use iso_c_binding
    type(c_ptr),value :: a,b,c
    type(c_ptr) :: handle
    integer(c_int) :: ma,mb,m,n,k,lda,ldb,ldc
    real(c_double) :: alpha,beta
  end subroutine omp_dgemm
end interface
end module rocm_interface

```

```

#include <rocblas.h>
extern "C" {
  void omp_dgemm(void *ptr, int modeA, int modeB, int m, int n,
    int k, double alpha, double *A, int lda,
    double *B, int ldb, double beta, double *C, int ldc) {
    rocblas_handle *handle = (rocblas_handle *) ptr;
    rocblas_dgemm(*handle,convert(modeA),convert(modeB),m,n,k,
    &alpha,A,lda,B,ldb,&beta,C,ldc);
  }
  void init_rocblas(void *ptr) {
    rocblas_handle *handle = (rocblas_handle *) ptr;
    rocblas_create_handle(handle);
  }
}

```

Translation unit 1  
ftn or  
amdflang

Translation unit 2  
hipcc

**HIP**

**Note: OMP data directives not needed in APU code, but shown here for portability to discrete GPUs**

# OpenMP® Calling HIP: Fortran and DGEMM

```

subroutine example
  use rocm_interface
  use iso_c_binding
  implicit none
  real(8),allocatable,target,dimension(:,:) :: a, b, c
  type(c_ptr)                               :: rocbas_handle
  ...

  allocate(da(M,N),db(N,K),dc(M,K))
  call init_matrices(da,db,dc,M,N,K)      ! Initialize matrices
  call init_rocbas(rocbas_handle)        ! Initialize rocBLAS
  ...

  !$OMP target enter data map(to:a,b,c)
  !$OMP target data use_device_addr(a,b,c)
  call omp_dgemm(rocbas_handle,modea,modeb,M,N,K,alpha,&
    c_loc(a),lda,c_loc(b),ldb,beta,c_loc(c),ldc)
  !$OMP end target data
  !$OMP target update from(c)
  !$OMP target exit data map(delete:a,b,c)
  ...
end subroutine example

```

You still have to create interfaces for your own HIP kernels, but not for the rocm library calls

... or build hipfort and use their readily available FORTRAN to HIP interface for rocm libraries  
<https://github.com/ROCmSoftwarePlatform/hipfort>

If you write your own HIP kernels for Fortran code, hipfort has many examples to see how it may be accomplished to call HIP from Fortran.

# Recommendation for CPU to APU porting

- Use OpenMP to port loops (see OpenMP presentation yesterday)
- Use rocm libraries where appropriate (see how in this presentation)
- Optimize OpenMP code (some hints in yesterday's presentation)

If you are a true performance enthusiast:

- Use HIP where optimization with OpenMP does not provide the expected performance (see tools presentation on last day)
- Optimize HIP (see next presentation)

Depending on which systems you want to run on:

- Make code portable to discrete GPUs (see OpenMP presentation yesterday and this presentation)

# Summary

- OpenMP® and HIP can be used in the same code
- C and HIP code has to be in different compilation units
  - Compile one with hipcc the other with amdclang++
- use\_device\_pointer or use\_device\_addr required on discrete GPUs or with HSA\_XNACK=0
- Calling HIP code with the APU programming model doesn't require memory movement
  
- Fortran requires C-binding interfaces to call HIP code
- Hipfort provides the interfaces for rocm library calls for Fortran

# Excercises

<https://github.com/amd/HPCTrainingExamples/tree/main/HIP-OpenMP>

The screenshot shows a GitHub commit page for the directory `HPCTrainingExamples / HIP-OpenMP`. The commit was made by `gcapodagAMD` with the message "Fixed typo" and commit hash `6b5c887` 2 months ago. A table below lists the files and folders in the commit:

Name	Last commit message	Last commit date
..		
CXX	Added README details for HIP-OpenMP examples	2 months ago
F/Calling_DGEMM	Fixed typo	2 months ago

# 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.

© 2025 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD ROCm, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries

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

**AMD** 