



HIP/OpenMP[®] Interoperability

Presenter: Giacomo Capodaglio
AMD @ CASTIEL HPC
Oct 28-30, 2025

AMD 
together we advance_

HIP and OpenMP[®] Interoperability

OpenMP[®] supports the following interactions:

- Calling low-level HIP kernels from OpenMP[®] application code
 - this includes HIP/ROCm libraries (rocBLAS, rocFFT, etc.)
- Calling OpenMP[®] kernels from low-level HIP application code
- Interaction with the underlying asynchronous stream mechanism

Why use both HIP and OpenMP®?

- OpenMP is a higher-level and more portable language
- Reduces code maintenance

Best practice:

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

To keep in mind:

- The HIP compiler `hipcc` cannot offload OpenMP® GPU kernels
- The AMD compilers used to offload OpenMP® GPU kernels cannot compile HIP kernels

OpenMP® application code with HIP kernel: OpenMP® app

```
void saxpy_hip(int n, float a, float * x, float * y);
```

wrapper declared here but defined on separate code compiled with 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];
```

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

```
    // allocate the device memory
```

```
    #pragma omp target data map(to:x[0:n]) map(tofrom:y[0:n])
```

```
{
```

```
    compute_1(n, x);
```

compute_1 and compute_2 run **on CPU** and modify x and y

```
    compute_2(n, y);
```

```
    #pragma omp target update to(x[0:n]) to(y[0:n]) // update x and y on the target
```

```
    #pragma omp target data use_device_ptr(x,y)
```

```
{
```

```
    saxpy_hip(n, a, x, y); // compute a * x[i] + y[i] in parallel
```

```
}
```

```
}
```

```
    compute_3(n, y);
```

This is a saxpy kernel in a low-level language (HIP).

```
}
```

OpenMP® application code with HIP kernel: HIP kernel

We need the HIP code to be in a separate file, since the compiler we use to do OpenMP® offloading to GPU cannot compile HIP kernels:

```
#include <stdio.h>
#include <hip/hip_runtime.h>

__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];
}
```

These are device pointers

```
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();
}
```

this is the wrapper that we use from the OpenMP code to call the HIP kernel

The device pointers needed by the HIP kernel can be supplied using `use_device_ptr` when calling the wrapper function

Full code at: https://github.com/amd/HPCTrainingExamples/blob/main/HIP-OpenMP/CXX/saxpy_openmp_hip/saxpy_hip.hip

OpenMP[®] application code with HIP kernel: 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];
}
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();
}

```

```

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:n]) map(tofrom:y[0:n])
    {
        compute_1(n, x); // mapping table: x:[0xabcd,0xef12], x = 0xabcd
        compute_2(n, y);
        #pragma omp target update to(x[0:n]) to(y[0:n]) // update x and y on the target
        #pragma omp target data use_device_ptr(x,y)
        {
            saxpy_hip(n, a, x, y); // mapping table: x:[0xabcd,0xef12], x = 0xef12
        }
    }
    compute_3(n, y);
}

```

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

Queries mapping table and finds device pointer associated with host array

Translation unit 1

hipcc

Translation unit 2

amdclang++

HIP application code with OpenMP[®] kernel: HIP app

```
void saxpy_openmp(int n, float a, float * x, float * y);
```

wrapper declared here but defined on separate code compiled with amdc1ang++

```
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 *h_x = new float[n];
    float *h_y = new float[n];
    float* d_x;
    float* d_y;

    // Size, in bytes, of each vector
    size_t bytes = n*sizeof(float);
    hipCheck( hipMalloc(&d_x, bytes) );
    hipCheck( hipMalloc(&d_y, bytes) );

    compute_1(n, h_x);
    compute_2(n, h_y);

    hipCheck( hipMemcpy(d_x, h_x, bytes, hipMemcpyHostToDevice) );
    hipCheck( hipMemcpy(d_y, h_y, bytes, hipMemcpyHostToDevice) );

    saxpy_openmp(n, a, d_x, d_y); // compute a * x[i] + y[i] in parallel

    hipCheck( hipMemcpy(h_x, d_x, bytes, hipMemcpyDeviceToHost) );
    hipCheck( hipMemcpy(h_y, d_y, bytes, hipMemcpyDeviceToHost) );

    compute_3(n, h_y);
}
```

Allocate device memory for x and y using the HIP API

compute_1 and compute_2 run **on CPU** and modify x and y

note the explicit memory copies from host to device and back

This is a saxpy with OpenMP[®] kernel

HIP application code with OpenMP[®] kernel: OpenMP[®] kernel

We need the code with the OpenMP[®] kernel to be in a separate file, since the hipcc compiler cannot do OpenMP[®] offloading:

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

Instructs the compiler to not do a presence check and avoids pointer translation

this is the wrapper that we use from the HIP code to call the OpenMP[®] kernel

Full code at: https://github.com/amd/HPCTrainingExamples/blob/main/HIP-OpenMP/CXX/saxpy_hip_openmp/saxpy_openmp.cc

Makefile example

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

OBJECTS = saxpy_openmp.o saxpy_hip.o

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

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

HIPCC ?= hipcc

HIPCC_FLAGS += -munsafe-fp-atomics
LDFLAGS = ${OPENMP_FLAGS} -L${ROCM_PATH}/lib -lamdhip64

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

$(EXECUTABLE): $(OBJECTS)
    $(CXX) $(OBJECTS) $(LDFLAGS) -o $@

clean:
    rm -f $(EXECUTABLE)
    rm -f $(OBJECTS)
```

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

Combine HIP kernels and OpenMP[®] kernels in single file

```

EXECUTABLE = ./daxpy
all: $(EXECUTABLE)

OBJECTS = daxpy.o daxpy_hip.o

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

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

HIPCC ?= hipcc

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

daxpy_hip.hip: daxpy.cc
    cp $^ $@

daxpy_hip.o: daxpy.cc
    $(HIPCC) $(HIPCC_FLAGS) -c $^ -o $@

$(EXECUTABLE): $(OBJECTS)
    $(CXX) $(OBJECTS) $(LDFLAGS) -o $@

clean:
    rm -f $(EXECUTABLE)
    rm -f $(OBJECTS) *.hip

```

- Trick to combine HIP and OpenMP[®] code in a single file
 - use **preprocessor directives**
 - **copy source into a separate file** before compiling

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_ptr(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

```

Translation unit 1
ftn or flang-new

```

#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);
  }
}

```

HIP

Translation unit 2
hipcc

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)                               :: 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_ptr(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

```

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>

Summary

- OpenMP® and HIP can be used in the same application code
- OpenMP® kernels and HIP kernels have to be in different compilation units
 - Additional step if using Fortran: C-binding interfaces to call HIP code
 - You can write them yourself or use hipfort
- The APU programming model doesn't require memory movement and therefore simplifies the interoperability of OpenMP® and HIP.

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 