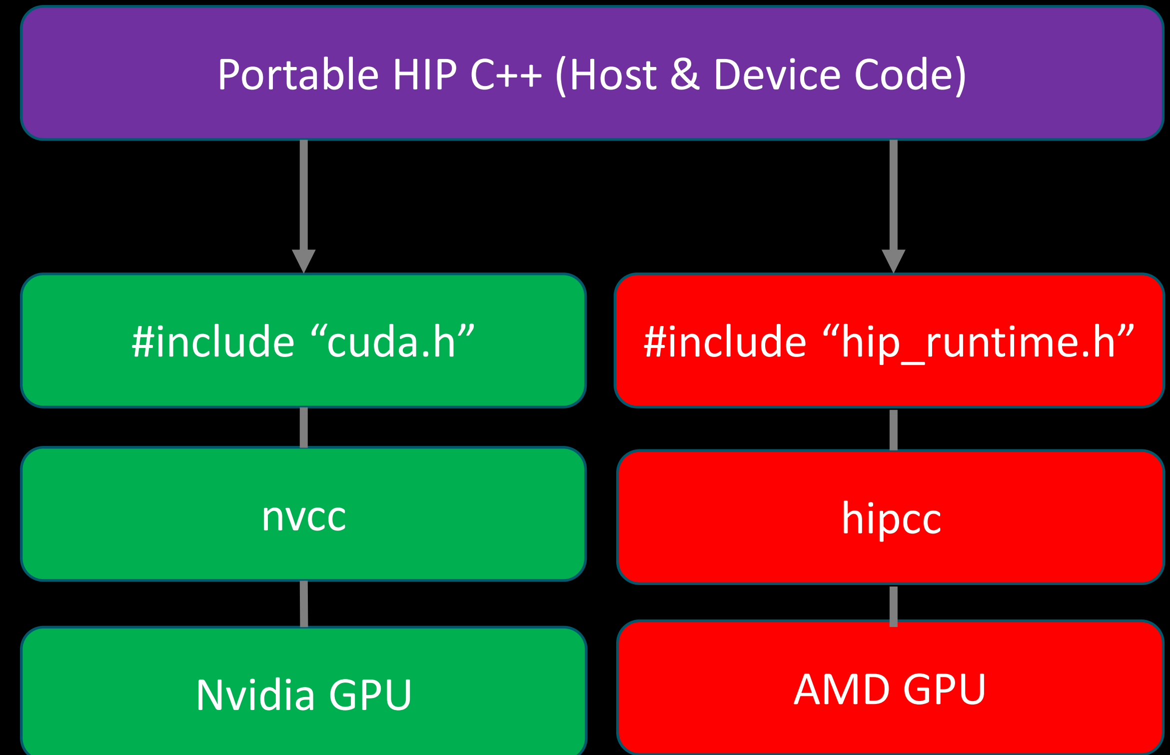
HIPFort: Present and Future Directions for Portable GPU Programming in Fortran

Alessandro Fanfarillo

September 2021

What is HIP?

AMD **H**eterogeneous-compute **I**nterface for **P**ortability, or **HIP**, is a C++ runtime API and kernel language that allows developers to create portable applications that can run on AMD accelerators as well as CUDA devices



HIP:

- Is open-source
- Provides an API for an application to leverage GPU acceleration for both AMD and CUDA devices
- Syntactically similar to CUDA - most CUDA API calls can be converted in place: cuda -> hip
- Supports a strong subset of CUDA runtime functionality

AMD

Kernels

A simple embarrassingly parallel loop

```
for (int i=0;i<N;i++) {
  h_a[i] *= 2.0;
}
```

Can be translated into a GPU kernel:

```
__global__ void myKernel(int N, double *d_a) {
  int i = threadIdx.x + blockIdx.x*blockDim.x;
  if (i<N) {
    d_a[i] *= 2.0;
  }
}
```

- A device function that will be launched from the host program is called a kernel and is declared with the __global__ attribute
- Kernels should be declared void
- All pointers passed to kernels must point to memory on the device
- All threads execute the kernel's body "simultaneously"
- Each thread uses its unique thread and block IDs to compute a global ID
- There could be more than N threads in the grid

AMD

[Public]

Kernels

Kernels are launched from the host:

```
dim3 threads(256,1,1);                                   //3D dimensions of a block of threads
dim3 blocks((N+256-1)/256,1,1);                          //3D dimensions the grid of blocks
hipMalloc(&d_a, N*sizeof(double));                       //Device memory allocation
hipMemcpy(d_a,h_a,N*sizeof(double),hipMemcpyHostToDevice); //Memory copy H->D
hipLaunchKernelGGL(myKernel,          //Kernel name (__global__ void function)
                        blocks,           //Grid dimensions
                        threads,          //Block dimensions
                        0,                //Bytes of dynamic LDS space
                        0,                //Stream (0=NULL stream)
                        N, d_a);            //Kernel arguments

hipFree(d_a);
```

Analogous to CUDA kernel launch syntax (supported by hipcc):

```
myKernel<<<blocks, threads, 0, 0>>>(N, d_a);
```

Threads in a block are executed in **64-wide** chunks called “wavefronts”

AMD

ROCm GPU Libraries

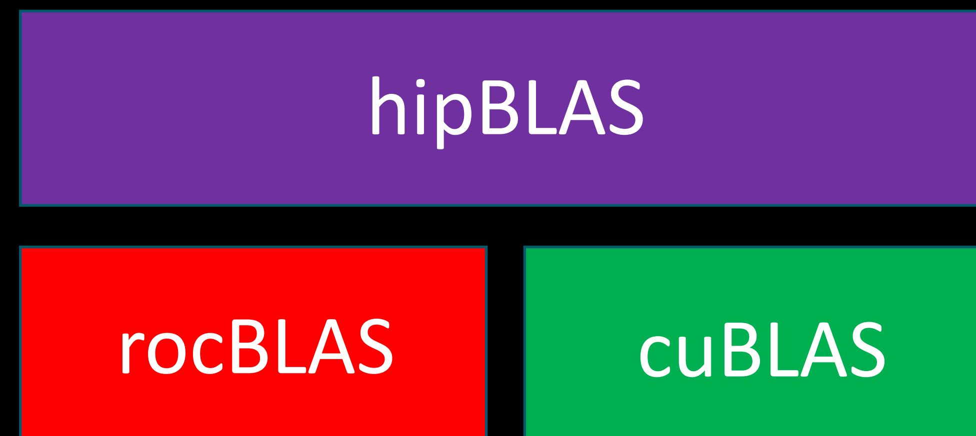
ROCm provides several GPU math libraries

- Typically, two versions:
  - roc* -> AMD GPU library, usually written in HIP
  - hip* -> Thin interface between roc* and Nvidia cu* library

When developing an application meant to target both CUDA and AMD devices, use the hip* libraries (portability)

When developing an application meant to target only AMD devices, may prefer the roc* library API (performance).

- Some roc* libraries perform **better** by using addition APIs not available in the cu* equivalents



AMD

[Public]

Fortran + HIP C/C++

- 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** write HIP kernels in C++
  - Wrap the kernel launch in a C function
  - Call the C function from Fortran through Fortran's ISO_C_binding
- This strategy should be usable by Fortran users since it is standard conforming Fortran
- ROCm has an interface layer, hipFort, which provides the wrapped bindings for use in Fortran
  - https://github.com/ROCmSoftwarePlatform/hipfort
- HIPFort provides interfaces for several hip* and roc* libraries

AMD

Hipfort installation and testing

```
git clone https://github.com/ROCmSoftwarePlatform/hipfort

cd hipfort; mkdir build ; cd build

cmake -DHIPFORT_INSTALL_DIR=/tmp/hipfort ..

make install

export PATH=/tmp/hipfort/bin:$PATH

cd ../test/f2003/vecadd

hipfc -v hip_implementation.cpp main.f03

./a.out
```


AMD

Vector add (f2003)

```fortran
interface
    subroutine launch(out,a,b,N) bind(c)
      use iso_c_binding
      implicit none
      type(c_ptr), value :: a, b, out
      integer, value :: N
    end subroutine
  end interface
  Nbytes = N*c_sizeof(c_float)
  type(c_ptr) :: da = c_null_ptr          ! Same for db and dout
  call hipCheck ( hipMalloc(da, Nbytes) ) ! Same for db and dout
  call hipCheck(hipMemcpy(da, c_loc(a), Nbytes, hipMemcpyHostToDevice))
  call launch(dout,da,db,N)
  call hipCheck(hipMemcpy(c_loc(out), dout, Nbytes, hipMemcpyDeviceToHost))
```

```cpp
__global__ void vector_add(float *out, float *a, float *b, int n)
{
  size_t index = blockIdx.x * blockDim.x + threadIdx.x;
  size_t stride = blockDim.x * gridDim.x;
  for (size_t i = index; i < n; i += stride)
    out[i] = a[i] + b[i];
}

extern "C"
{
  void launch(float *dout, float *da, float *db, int N)
  {
      hipLaunchKernelGGL((vector_add), dim3(320), dim3(256), 0, 0, dout, da, db, N);
  }
}
```


AMD

Vector add (f2008)

```
interface
    subroutine launch(grid,block,shmem,stream,out,a,b,N) bind(c)
      use iso_c_binding
      use hipfort_types
      implicit none
      type(c_ptr),value :: a, b, out
      integer(c_int), value :: N, shmem
      type(dim3) :: grid, block
      type(c_ptr),value :: stream
    end subroutine
end interface

Real(8), allocatable, dimension(:) ::  a, b, out
Real(8), pointer, dimension(:) :: da, db, dout

Allocate(a(N), b(N), out(N)

call hipCheck( hipMalloc( da, N ) ) !same for db

Call hipCheck( hipMalloc( db, source=b ) )

Type(dim3) :: grid = dim3(320,1,1)

Type(dim3) :: block = dim3(256,1,1)

Call hipCheck ( hipMemcpy(da, a, N, hipMemcopyHostToDevice) ) !same for db

Call launch( grid, block, 0, c_null_ptr, c_loc(dout), c_loc(da), c_loc(db), N )

call hipCheck ( hipMemcpy(out,dout,N,hipMemcpyDevicetoHost) )
```

```
Interface hipMalloc

  function hipMalloc_b ( ptr, mySize ) bind (c, name=hipMalloc)
    integer( kind(hipSuccess) ) :: hipMalloc_b
    type(c_ptr) :: ptr
    integer( kind(c_size_t) ) :: mySize
  end function

  module procedure hipMalloc_l_0_source, hipMalloc_l_1_source, hipMalloc_l_1_c_int, &
      hipMalloc_l_1_c_size_t, hipMalloc_l_1_2_source, ….
!for int4, int8, real4, real8, complex4, complex8, for all possible ranks (in this case 8).

end interface

function hipMalloc_i4_1_c_size_t(ptr,length1)
     use iso_c_binding
     use hipfort_enums
     use hipfort_types
     implicit none
     integer(4),pointer,dimension(:), intent(inout) :: ptr
     integer(c_size_t) :: length1
     type(c_ptr) :: cptr
     integer(kind(hipSuccess)) :: hipMalloc_i4_1_c_size_t

     hipMalloc_i4_1_c_size_t = hipMalloc_b(cptr,length1*4_8)
     call C_F_POINTER(cptr,ptr,SHAPE=[length1])
end function
```


AMD

[Public]

Vector add (f2008)

```
interface
   subroutine launch(grid,block,shmem,stream,out,a,b,N) bind(c)
     use iso_c_binding
     use hipfort_types
     implicit none
     type(c_ptr),value :: a, b, out
     integer(c_int), value :: N, shmem
     type(dim3) :: grid, block
     type(c_ptr),value :: stream
   end subroutine
end interface

Real(8), allocatable, dimension(:) ::  a, b, out
Real(8), pointer, dimension(:) :: da, db, dout

Allocate(a(N), b(N), out(N))

call hipCheck( hipMalloc( da, N ) ) !same for db

Call hipCheck( hipMalloc( db, source=b ) )

Type(dim3) :: grid = dim3(320,1,1)

Type(dim3) :: block = dim3(256,1,1)

Call hipCheck ( hipMemcpy(da, a, N, hipMemcopyHostToDevice) ) !same for db

Call launch( grid, block, 0, c_null_ptr, c_loc(dout), c_loc(da), c_loc(db), N )

call hipCheck ( hipMemcpy(out,dout,N,hipMemcpyDevicetoHost) )
```

```
Interface hipMemcpy

function hipMemcpy_b(dest,src,sizeBytes,myKind) bind(c, name="hipMemcpy")
     use iso_c_binding
     use hipfort_enums
     use hipfort_types
     implicit none
     integer(kind(hipSuccess)) :: hipMemcpy_b
     type(c_ptr),value :: dest
     type(c_ptr),value :: src
     integer(c_size_t),value :: sizeBytes
     integer(kind(hipMemcpyHostToHost)),value :: myKind
   end function

  module procedure hipMemcpy_l_0,  hipMemcpy_l_0_c_int,  hipMemcpy_l_0_c_size_t,
     hipMemcpy_l_1,  hipMemcpy_l_1_c_int,& ....

 function hipMemcpy_l_2_c_size_t(dest,src,length,myKind)
     use iso_c_binding
     use hipfort_enums
     use hipfort_types
     implicit none
     logical,target,dimension(:,:),intent(inout) :: dest
     logical,target,dimension(:,:),intent(in)    :: src
     integer(c_size_t),intent(in) :: length
     integer(kind(hipMemcpyHostToHost)) :: myKind
integer(kind(cudaSuccess)) :: hipMemcpy_l_2_c_size_t
integer(kind(hipSuccess)) :: hipMemcpy_l_2_c_size_t

     hipMemcpy_l_2_c_size_t = hipMemcpy_b(c_loc(dest),c_loc(src),length*1_8,myKind)
end function
```


AMD

Fortran 2018 low-level C Interop.

Fortran 2018 provides **optional**, **assumed-type** dummy arguments, and **assumed-rank** :

```
Interface
  function hipMemcpy_b(dest,source,N,myKind) bind(c,name='hipmemcpy_')
       use iso_c_binding, only: c_ptr, c_size_t
       integer(kind(hipSuccess)) :: hipMemcpy_b
       type(c_ptr),value :: dest
       type(c_ptr),value :: source
       integer(c_size_t),value :: N
       integer(kind(hipMemcpyHostToHost)),value :: myKind
     end function hipMemcpyDH_b
End interface

 Function hipMemcpy (device, host, hipMemcpyKind)
    integer(kind(hipSuccess) :: res
    type(*), target :: ptr (..)
    type(*), target :: source(..)
    integer(kind(hipMemcpyHostToHost)) :: hipMemcpyKind

    res = hipMemcpy_b( c_loc(dest), c_loc(source), N, hipMemcpyKind)
  end function
```

In C:

```
int hipmemcpy_(void *dest, void *source, size_t N, int myKind)
{
    return hipMemcpy(dest,source,N,myKind);
}
```


AMD

Fortran 2018 C Descriptor

From ISO_Fortran_binding.h in gfortran:

```
typedef struct CFI_dim_t
 {
   CFI_index_t lower_bound;
   CFI_index_t extent;
   CFI_index_t sm;
 }
CFI_dim_t;

typedef struct CFI_cdesc_t
 {
   void *base_addr;
   size_t elem_len;
   int version;
   CFI_rank_t rank;
   CFI_attribute_t attribute;
   CFI_type_t type
   CFI_dim_t dim[];
 }
CFI_cdesc_t;
```

```
Void your_function(CFI_cdesc_t *desc)
{
  size_t new_n = 1;
  for(int r = 0; r < desc->rank; r++)
     new_n *= desc->dim[r].extent;
}
```

It might be tempting to do something like this:

```
Void myHipMalloc(CFI_cdesc_t *desc, size_t N)
{
  hipMalloc(&desc->base_addr,N);
}
```

THIS IS NOT GOOD. Changes to the cdesc must be done via the CFI_* routines exposed by ISO_Fortran_binding.h

AMD

[Public]

Main HIPFort Contributors

Thanks to:

Dominic Charrier

Greg Rodgers

Paul Bauman

AMD

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.

© 2021 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ROCm, Radeon, Radeon Instinct and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

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

AMD