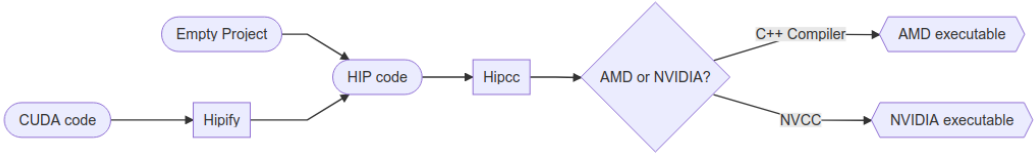
AMD HIP porting tools and methods

SURF Open Innovation Lab and University of Amsterdam
In partnership with NIKHEF

What is HIP

- Open source C++ runtime API and kernel language
- Part of AMD ROCm
- Similar programming model to CUDA.
- Platform independent (targets NVIDIA and AMD cards)
- Feature compariable with CUDA // (in spirit).

What is HIP


Empty Project
CUDA code
Hipify
HIP code
Hipcc
AMD or NVIDIA?
C++ Compiler
AMD executable
NVCC
NVIDIA executable

HIP code example

Term	CUDA	HIP
Event	cudaEvent_t	hipEvent_t
Thread-index	threadIdx.x	hipThreadIdx_x
Device kernel	__global__	__global__
Group Memory	__shared__	__shared__
Constant	__constant__	__constant__

Table: Term translation table

HIP code example: vector add

```
__global__
void vectorAddKernel(float* deviceA, float* deviceB, float* deviceC){
    // unsigned index = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned index = hipBlockIdx_x * hipBlockDim_x + hipThreadIdx_x;
    deviceC[index] = deviceA[index] + deviceB[index];
}
```

HIP code example: memory copy

```
// cudaMalloc((void **) &deviceA, n*sizeof(float));
// cudaMemcpy(deviceA, hostA, n*sizeof(float), cudaMemcpyHostToDevice);
hipMalloc((void **) &deviceA, n*sizeof(float));
hipMemcpy(deviceA, hostA, n*sizeof(float), hipMemcpyHostToDevice);
```

HIP code example: kernel launch

```
// vectorAddKernel<<<n/threadBlockSize, threadBlockSize>>>
//                  (deviceA, deviceB, deviceC);
hipLaunchKernelGGL((vectorAddKernel),
                   dim3(n/threadBlockSize), dim3(threadBlockSize),
                   0 /*Shared*/, 0/*stream*/,
                   deviceA, deviceB, deviceC);
```

Hipify

- Translates a project from CUDA to HIP.
- Auto-translates most of the calls and kernels.
- Reports what it could not change.
- Support for most well known CUDA libraries

Libraries

CUDA	HIP	
cuBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuFFT	rocFFT	Fast Fourier Transfer Library
cuSPARSE	rocSPARSE	Sparse BLAS + SPMV
cuSolver	rocSolver	Lapack library
AMG-X	rocALUTION	Sparse iterative solvers and preconditioners
Thrust	hipThrust	C++ parallel algorithms library
CUB	rocPRIM	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	Deep learning Solver Library
cuRAND	rocRAND	Random Number Generator Library
EIGEN	EIGEN (Ported)	C++ template library for linear algebra
NCCL	RCCL	Communications Primitives Library based on the MPI

Table: Cuda compatible libraries

Hipify tools

- **hipexamine**: Scans source directory and reports which files contain CUDA code and how much of the code can automatically be ported.
- **hipify**: "(Hipifies)" a single file.
- **hipconvertinplace**: Ports in-place a full directory from CUDA to HIP.
- **hipify-cmakefile**: Translates cmake files to use HIP.

Porthing methods

- Hipify-perl
- Hipify-clang
- Custom defines to map cuda calls to hip

Hipify-perl

- Convert provided files to HIP
- Intelligent search replace
- Might not work for more complex codes

Hipify-clang

- Clang based source parsing
- Can replace the compiler to convert a large complex problem
- Works better on complex (c++) inputs

Using Defines

- Minimal changes needed, only ifdefs for the kernel launches
- No need for a fork of the repo while porting continues.
- Easier to integrate in large projects

HIP code example: Porting Defines

```
#define cudaMalloc hipMalloc
#define cudaMallocHost hipHostMalloc
#define cudaMemcpy hipMemcpy
#define cudaMemcpyAsync hipMemcpyAsync
#define cudaMemset hipMemset
#define cudaMemsetAsync hipMemsetAsync
#define cudaPeekAtLastError hipPeekAtLastError
#define cudaEventCreate hipEventCreate
#define cudaEventCreateWithFlags hipEventCreateWithFlags
```

HIP code example: Porting Defines

```
#if defined(__HIPCC__)
hipLaunchKernelGGL((vectorAddKernel),
                   dim3(n/threadBlockSize), dim3(threadBlockSize),
                   0 /*Shared*/, 0/*stream*/,
                   deviceA, deviceB, deviceC);
#else
vectorAddKernel<<<n/threadBlockSize, threadBlockSize>>>
                (deviceA, deviceB, deviceC);
#endif
```

Porting examples

- Porting of Nvidia sample codes
- Error checking removed on slides
- CuBlAS, FFT and hostcode examples

Porting examples

```
cuhipblasCreate(&handle);
cuhipblasSetVector(n2, sizeof(h_A[0]), h_A, 1, d_A, 1);
cuhipblasSetVector(n2, sizeof(h_A[0]), h_A, 1, d_A, 1);
cuhipblasSetVector(n2, sizeof(h_C[0]), h_C, 1, d_C, 1);
cuhipblasSgemm(handle, CUHIPBLAS_OP_N, CUHIPBLAS_OP_N, N,
                    N, N, &alpha, d_A,N, d_B, N, &beta, d_C, N);
cuhipblasGetVector(n2, sizeof(h_C[0]), d_C, 1, h_C, 1);
cuhipblasDestroy(handle);
```

Porting examples

```
cuhipfftPlan1d(&plan, new_size, CUHIPFFT_C2C, 1);
cuhipfftExecC2C(plan, cuhipfftComplex *)d_signal,
                  (cuhipfftComplex *)d_signal, CUHIPFFT_FORWARD);
cuhipfftExecC2C(plan, (cuhipfftComplex *)d_filter_kernel,
                  (cuhipfftComplex *)d_filter_kernel, CUHIPFFT_FORWARD);
printf("Launching ComplexPointwiseMulAndScale<<< >>>\n");
cuhipLaunchKernelGGL(ComplexPointwiseMulAndScale, dim3(32), dim3(256),
                       0, 0, d_signal, d_filter_kernel,
                       new_size, 1.0f / new_size);
cuhipfftExecC2C(plan, (cuhipfftComplex *)d_signal,
                (cuhipfftComplex *)d_signal, CUHIPFFT_BACKWARD);
```

Hipify-perl output

```
info: converted 3 CUDA->HIP refs ( ... )
warn:1 LOC:274 in './simplefft.cpp'
hipMemcpy 3
hipFree 2
HIPFFT_FORWARD 2
hipMemcpyHostToDevice 2
hipMalloc 2
hipDeviceReset 1
hipLaunchKernelGGL 1
HIPFFT_C2C 1
hipMemcpyDeviceToHost 1
HIPFFT_BACKWARD 1
hip_runtime 1
```

Hipify-clang output

```
[HIPIFY] info: file './simplefft.cpp.old' statistics:
CONVERTED refs count: 30
UNCONVERTED refs count: 0
CONVERSION %: 100
REPLACED bytes: 456
TOTAL bytes: 9346
CHANGED lines of code: 20
TOTAL lines of code: 274
CODE CHANGED (in bytes) %: 5
CODE CHANGED (in lines) %: 7
TIME ELAPSED s: 0.17
[HIPIFY] info: CONVERTED refs by type:
[HIPIFY] info: CONVERTED refs by API:
```

Pitfalls

- `hipcc` is a lot stricter compared to nvcc
- All functions that are called from the device need to be marked as such
- Low level optimizations for cuda cards need to be redone

Conclusion

- HIP enables the development of portable GPU codes
- Most CUDA code can be ported to HIP
- Tooling is still improving
- Performance depends on the kernels