HIP for CUDA Programmers

Getting you up to speed on converting your CUDA code to HIP

Subil Abraham

HPC Engineer

What We'll Cover Today

(Does require a basic familiarity with CUDA)

- Get you familiar with hipify tools
  - Demonstrate usage through several examples
- Show things to watch out for with hipify and compiling with HIP
- AMD talk – Alessandro Fanfarillo – Experiences with CAAR apps
- Exercises for you to practice hipify

Brief Overview of HIP

- AMD's API for GPU programming.
- Usable with both ROCm backend (for AMD GPUs) and CUDA backend (for Nvidia GPUs).
- Almost 1 to 1 replacement of CUDA (cudaAbcCall -> hipAbcCall)
  - Some CUDA calls not supported, because they are deprecated or not yet implemented for HIP
  - Documentation: docs.amd.com
  - HIP-CUDA support table https://github.com/ROCm-Developer-Tools/HIPIFY#cuda-apis

Why use HIP?

- Well, you want to run on Frontier, don't you?
- (Mostly) Identical to CUDA, so almost no learning curve.
  - cudaMalloc -> hipMalloc
  - cudaDeviceSynchronize -> hipDeviceSynchronize
  - mykernel<<<blocks, grid>>>(args) -> hipLaunchKernelGGL(args)**
- Can be used for AMD, Nvidia and (soon*) Intel GPUs
- Existing tools for converting your CUDA code to HIP

*Ongoing ECP project
**mykernel<<<>>> syntax may be supported in HIP now

Converting CUDA to HIP

- A couple of tools available
  - hipify-perl – regex find and replace
  - hipify-clang – think of it as a source to source compiler. Walks the AST, works for more complicated constructs where regex might fail.
- For most cases, they should work the same.
  - hipify-perl will warn if you have user defined calls with prefix ‘cuda’ (e.g. cudaErrorCheck macro)
  - Both will warn if it encounters unsupported (legitimate) Cuda API call (e.g. cublasZgemm3m has no HIP equivalent)
  - I’ve yet to encounter where I would need one over the other, but I’ve only done relatively simple cases. So keep your eyes open.

What's Available on Summit

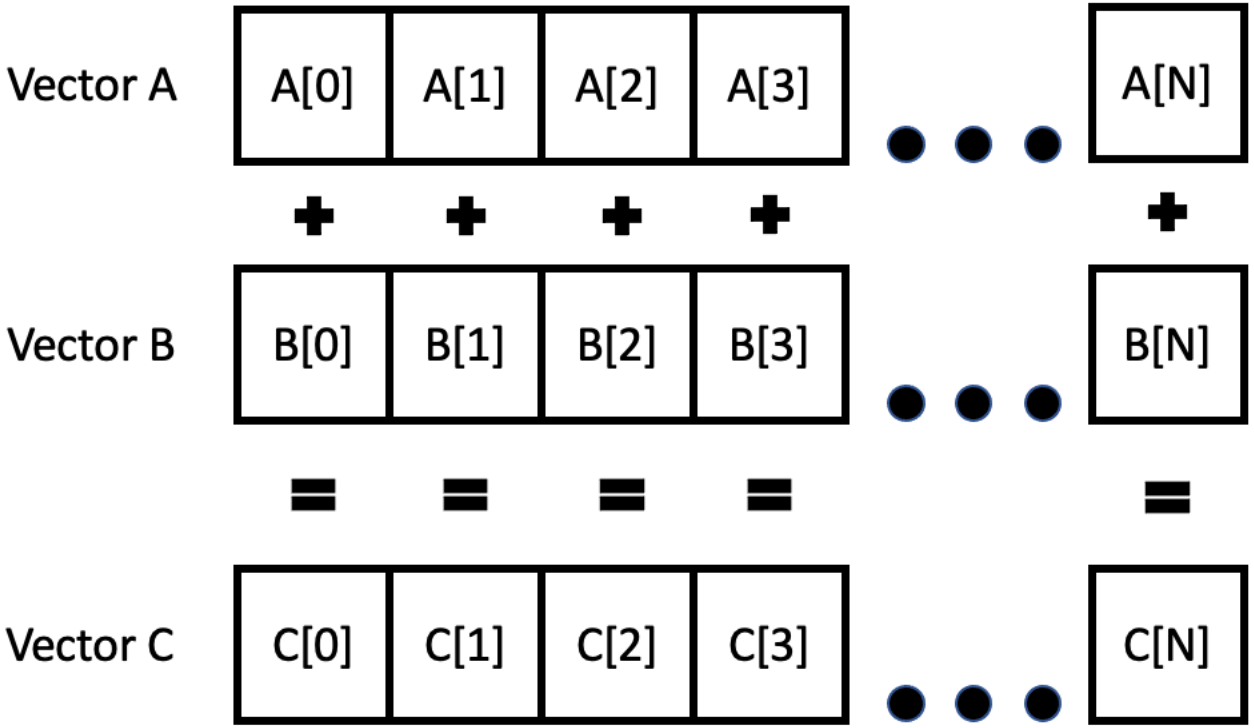
- `module load cuda/11.5.2 hip-cuda`
- Currently supported - HIP 5.1.0
- This module also includes the following libraries:
  - hipBLAS (we'll cover an example and exercise)
  - hipFFT
  - hipSolver
  - hipSparse
  - hipRand
- These are (mostly) equivalent to the corresponding CUDA libraries

Let’s Look At Some Examples

- git clone https://github.com/olcf/HIP_for_CUDA_programmers
- Follow along in your terminal
- Add #BSUB –U HIPforCUDA to your batch scripts to use today’s reservation

Vector Add

- Needs no introduction – parallel addition of two arrays



Vector add

- run `hipify-perl vector_addition_nohipifywarnings.cu > vector_addition_nohipifywarnings_hip.cpp`
- cudaXyz --> hipXyz translated in all cases, and work the same

```
kernel_name<<< blocks_per_grid,
threads_per_block,
shared_memory,
stream_id >>>(
       kernel_arg1,
       kernel_arg2, ...
)
```

```
hipLaunchKernelGGL(
       kernel_name,
       dim3(blocks_per_grid),
       dim3(threads_per_block),
       dynamic_shared_memory,
       stream_id,
       kernel_arg1,
       kernel_arg2, ...
)
```

Vector add

- Run `hipify-perl vector_addition.cu > vector_addition_hip.cpp`
- Look at all the warnings and fix them.

cpu_gpu_dgemm

- Matrix multiplication with double precision FP

This function performs the matrix-matrix multiplication
$C = \alpha \mathrm{op}(A)\mathrm{op}(B) + \beta C$ *where op(X)is one of op(X) = X, or op(X) =* X^T*, or op(X) =* X^H
where α and β are scalars, and A , B and C are matrices stored in column-major format with dimensions op(A) $m \times k$, op(B) $k \times n$ and C $m \times n$, respectively

`cublasStatus_t cublasDgemm(`
`cublasHandle_t handle,`
`cublasOperation_t transa,`
`cublasOperation_t transb,`
`int m, int n, int k,`
`const double *alpha,`
`const double *A, int lda,`
`const double *B, int ldb,`
`const double *beta,`
`double *C, int ldc`
`)`
	`hipblasStatus_t hipblasDgemm(`
	`hipblasHandle_t handle,`
	`hipblasOperation_t transa,`
	`hipblasOperation_t transb,`
	`int m, int n, int k,`
	`const double *alpha,`
	`const double *A, int lda,`
	`const double *B, int ldb,`
	`const double *beta,`
	`double *C, int ldc`
	`)`
		`void dgemm(`
		`char* transa,`
		`char* transb,`
		`int m, int n, int k,`
		`double *alpha,`
		`double *A, int lda,`
		`double *B, int ldb,`
		`double *beta,`
		`double *C, int ldc`
		`)`

cpu_gpu_zgemm

- Matrix multiplication with double precision FP for complex numbers

`cublasStatus_t cublasZgemm(`
`  cublasHandle_t handle,`
`  cublasOperation_t transa,`
`  cublasOperation_t transb,`
`  int m, int n, int k,`
`  const cuDoubleComplex *alpha,`
`  const cuDoubleComplex *A,`
`  int lda,`
`  const cuDoubleComplex *B,`
`  int ldb,`
`  const cuDoubleComplex *beta,`
`  cuDoubleComplex *C,`
`  int ldc`
`)`
	`hipblasStatus_t hipblasZgemm(`
	`  hipblasHandle_t handle,`
	`  hipblasOperation_t transa,`
	`  hipblasOperation_t transb,`
	`  int m, int n, int k,`
	`  const hipblasDoubleComplex *alpha,`
	`  const hipblasDoubleComplex *A,`
	`  int lda,`
	`  const hipblasDoubleComplex *B,`
	`  int ldb,`
	`  const hipblasDoubleComplex *beta,`
	`  hipblasDoubleComplex *C,`
	`int ldc`
	`)`
		`void zgemm(`
		`  char* transa,`
		`  char* transb,`
		`  int m, int n, int k,`
		`  complex *alpha,`
		`  complex *A, int lda,`
		`  complex *B, int ldb,`
		`  complex *beta,`
		`  complex *C, int ldc`
		`)`

Things to Note

- Since HIP uses CUDA backend on Summit, you can profile compiled code with Nvidia Nsight tools & debuggers.
- Try to use platform agnostic names e.g. gpuErrorCheck instead of cudaErrorCheck (or whichever naming scheme works best for your team and code).

Things to Note

- Pass the `-Xcompiler -x -Xcompiler c++` flags to hipcc when using `hipcc -ccbin xlc++_r` (see examples/redundant_MM/onefile/hipversion/Makefile.hipcc)
  - Not necessary when you're using gcc as your underlying compiler
- Compiling a HIP file with `OMPI_CXX=hipcc mpicxx` will fail because mpicxx automatically adds the `-pthread` flag which hipcc doesn't support. This is an issue with the mpi compiler wrapper (see examples/redundant_MM/onefile/hipversion/Makefile.mpicc).
  - Compile with hipcc directly and link in the MPI libraries instead if your HIP file mixes MPI and HIP code.
- hipcc does not support PGI compiler, `hipcc -ccbin pgc++` will error.
  - hipcc uses clang flags, which match gcc and xl flags so gcc and xl work for the most part as the underlying compiler.
- When using mpicc for linking, link both the CUDA and HIP libraries (see examples/redundant_MM/twofiles/hipversion/Makefile.mpicclink)

Conclusions

- HIP mostly supports CUDA API
- Hipify tools will convert supported CUDA calls to HIP, and warn if something not supported
- If anything is not supported:
  - Write to the help desk, we'll work with the vendors
  - Implement the kernel yourself, use an alternate HIP call, or use the CPU version
  - (On Summit) use an ifdef to use the CUDA call and link the CUDA libraries (see examples/cpu_gpu_zgemm/hipversion)
- Let us know if you run into any issues as you try things out