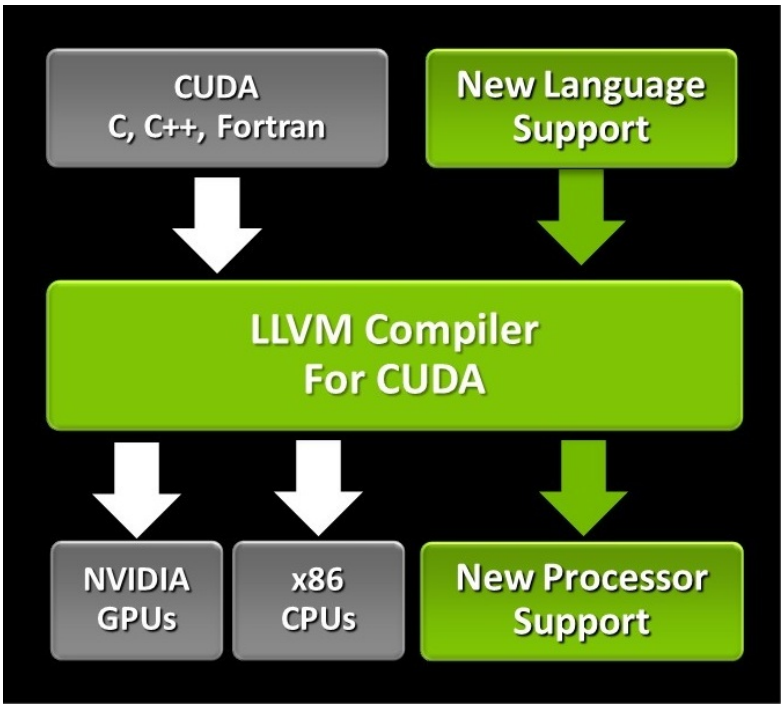
Viability and performance characteristics of using HIPify for CUDA projects

Why would that be needed?

CUDA is the most popular GPU computing platform. But it is also completely propietary, supporting only Nvidia GPUs. Even though CUDA is based on Open Source LLVM compiler infrastructure, and LLVM can compile CUDA even without NVCC that is based on it, it still needs propietary CUDA SDK.[^1]



CUDA LLVM structure

There has been many approaches to make CUDA code run on other GPUs, but I will focus on the most mature HIP (Heterogeneous Interface for Portability) and ROCm software stack, which are both fully opensource.

What is HIP?

HIP is a portable source code that can be compiled for both AMD and Nvidia GPUs using `HCC` or `NVCC` respectively. HIP code can be extremely similar to CUDA C++, with only CUDA function names being replaced with HIP ones.

HCC is based on LLVM same as NVCC, I assume not only because it is so powerful, but also because LLVM already had support for CUDA, so AMD could reuse a lot of that code. This means LLVM is now used for CUDA, while AMD and Intel use it for both graphics (shader) and compute compiler. This will hopefully mean more vendor compatibility in the future.

What is HIPify?

HIPify-clang is a tool to convert CUDA to Portable C++ Code. It is utilising LLVM's CUDA support and "It translates CUDA source into an abstract syntax tree, which is traversed by transformation matchers. After applying all the matchers, the output HIP source is produced."[1]

From the start this approach could mean some incompatibility, because even LLVM themselves are warning that NVCC is parsing code slightly differently than mainline LLVM.[2]

This approach supports all the CUDA versions LLVM does, so it is only a few months behind what Nvidia releases, and "even very complicated constructs will be parsed successfully".[3]

But I found out this approach brings a lot of its own incompatibilities with it. It is a standalone package that doesn't require the rest of ROCm stack, but it does require a new LLVM (the newer the higher CUDA support), as well as a full CUDA SDK installation.

That fact immediately breaks the idea of fully open source running of CUDA code. Also because of its license many Linux distributions can't package CUDA and include it in its repositories, including mine. I tried in many ways manually extracting the CUDA installer, to get the libraries and headers. But it turns out that HIPify still uses LLVM's full CUDA detection, which means LLVM checks all components of CUDA and if they were properly installed, including e.g. device libraries.[4]

After a lot of trying I had to use a different OS for just translation to HIP, I used Ubuntu 20.04 with CUDA SDK straight from Nvidia, and then I could use that translated code on my OS with fully build from source software stack.

```
~/Downloads/HIPIFY-rocm-5.4.2/dist/hipify-clang vectorAdd.cu -I ~/Downloads/cuda-samples-11.6/Common/ --experimental
```

Incompatibilities when converting using HIPify

Only now we can deal with actual conversion and its shortcomings. I have decided to use Nvidia's CUDA Samples[5] because they offer a wide range of features to be tested, as well as I expected they would be better supported because they are the official reference code. And yet I still encountered many problems with them

- You can't specify which c++ version it is using for converting, like you can when compiling CUDA, but you can when compiling. This meant I had to replace c++ 11's sprintf_s with something supported in previous version, even when HIPify should be using C++ 11 by default

```
/tmp/deviceQuery.cpp-040229.hip:308:3: error: use of undeclared identifier 'sprintf_s'
 sprintf_s(cTemp, 10, "%d.%d", driverVersion / 1000,
```

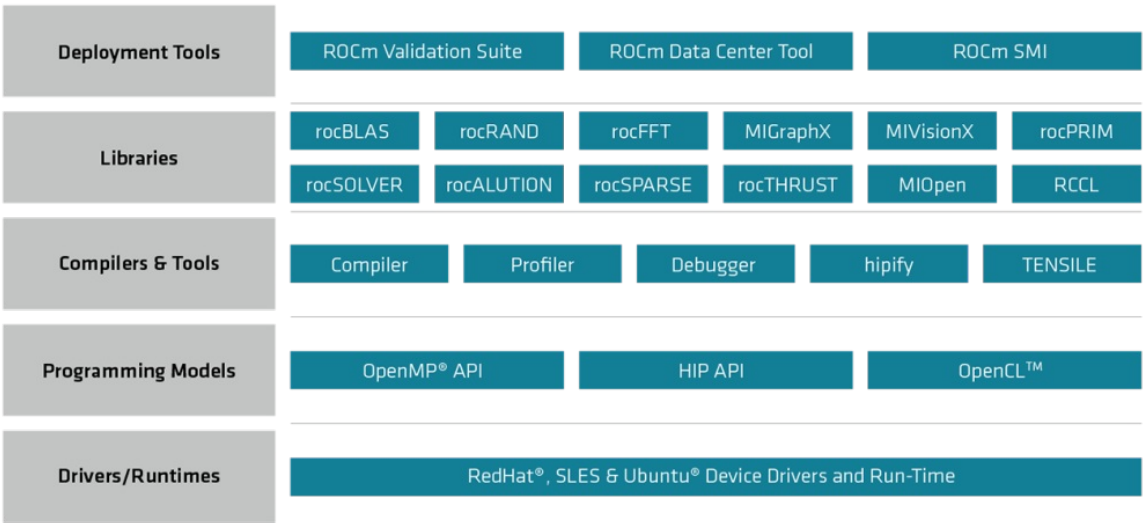
- some device properties from CUDA are not implemented in HIP's hipDeviceProp_t

```
deviceQuery.cpp.hip:180:20: error: no member named 'maxTexture1DLayered' in 'hipDeviceProp_t'; did you mean 'maxTexture1DLinear'?
      deviceProp.maxTexture1DLayered[0], deviceProp.maxTexture1DLayered[1]);
                 ^~~~~~~~~~~~~~~~~~~
                 maxTexture1DLinear
/usr/include/hip/hip_runtime_api.h:122:9: note: 'maxTexture1DLinear' declared here
    int maxTexture1DLinear;    ///< Maximum size for 1D textures bound to linear memory
        ^
deviceQuery.cpp.hip:180:39: error: subscripted value is not an array, pointer, or vector
        deviceProp.maxTexture1DLayered[0], deviceProp.maxTexture1DLayered[1]);
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~
deviceQuery.cpp.hip:180:55: error: no member named 'maxTexture1DLayered' in 'hipDeviceProp_t'; did you mean 'maxTexture1DLinear'?
        deviceProp.maxTexture1DLayered[0], deviceProp.maxTexture1DLayered[1]);
                                                      ^~~~~~~~~~~~~~~~~~~
                                                      maxTexture1DLinear
/usr/include/hip/hip_runtime_api.h:122:9: note: 'maxTexture1DLinear' declared here
    int maxTexture1DLinear;    ///< Maximum size for 1D textures bound to linear memory
        ^
deviceQuery.cpp.hip:180:74: error: subscripted value is not an array, pointer, or vector
        deviceProp.maxTexture1DLayered[0], deviceProp.maxTexture1DLayered[1]);
                                           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~
deviceQuery.cpp.hip:184:20: error: no member named 'maxTexture2DLayered' in 'hipDeviceProp_t'; did you mean 'maxTexture1DLinear'?
        deviceProp.maxTexture2DLayered[0], deviceProp.maxTexture2DLayered[1],
                   ^~~~~~~~~~~~~~~~~~~
                   maxTexture1DLinear
/usr/include/hip/hip_runtime_api.h:122:9: note: 'maxTexture1DLinear' declared here
    int maxTexture1DLinear;    ///< Maximum size for 1D textures bound to linear memory
        ^
deviceQuery.cpp.hip:184:39: error: subscripted value is not an array, pointer, or vector
        deviceProp.maxTexture2DLayered[0], deviceProp.maxTexture2DLayered[1],
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~
deviceQuery.cpp.hip:184:55: error: no member named 'maxTexture2DLayered' in 'hipDeviceProp_t'; did you mean 'maxTexture1DLinear'?
        deviceProp.maxTexture2DLayered[0], deviceProp.maxTexture2DLayered[1],
                                                      ^~~~~~~~~~~~~~~~~~~
                                                      maxTexture1DLinear
/usr/include/hip/hip_runtime_api.h:122:9: note: 'maxTexture1DLinear' declared here
    int maxTexture1DLinear;    ///< Maximum size for 1D textures bound to linear memory
        ^
deviceQuery.cpp.hip:184:74: error: subscripted value is not an array, pointer, or vector
        deviceProp.maxTexture2DLayered[0], deviceProp.maxTexture2DLayered[1],
                                           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~
deviceQuery.cpp.hip:185:20: error: no member named 'maxTexture2DLayered' in 'hipDeviceProp_t'; did you mean 'maxTexture1DLinear'?
        deviceProp.maxTexture2DLayered[2]);
                   ^~~~~~~~~~~~~~~~~~~
                   maxTexture1DLinear
/usr/include/hip/hip_runtime_api.h:122:9: note: 'maxTexture1DLinear' declared here
    int maxTexture1DLinear;    ///< Maximum size for 1D textures bound to linear memory
        ^
deviceQuery.cpp.hip:185:39: error: subscripted value is not an array, pointer, or vector
        deviceProp.maxTexture2DLayered[2]);
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~
deviceQuery.cpp.hip:192:23: error: no member named 'sharedMemPerMultiprocessor' in 'hipDeviceProp_t'; did you mean
'maxSharedMemoryPerMultiProcessor'?
          deviceProp.sharedMemPerMultiprocessor);
                     ^~~~~~~~~~~~~~~~~~~~~~~~~~~
                     maxSharedMemoryPerMultiProcessor
/usr/include/hip/hip_runtime_api.h:114:12: note: 'maxSharedMemoryPerMultiProcessor' declared here
    size_t maxSharedMemoryPerMultiProcessor;  ///< Maximum Shared Memory Per Multiprocessor.
           ^
deviceQuery.cpp.hip:214:21: error: no member named 'deviceOverlap' in 'hipDeviceProp_t'
        (deviceProp.deviceOverlap ? "Yes" : "No"), deviceProp.asyncEngineCount);
        ~~~~~~~~~~ ^
deviceQuery.cpp.hip:214:63: error: no member named 'asyncEngineCount' in 'hipDeviceProp_t'
        (deviceProp.deviceOverlap ? "Yes" : "No"), deviceProp.asyncEngineCount);
                                                   ~~~~~~~~~~ ^
deviceQuery.cpp.hip:222:23: error: no member named 'surfaceAlignment' in 'hipDeviceProp_t'
          deviceProp.surfaceAlignment ? "Yes" : "No");
          ~~~~~~~~~~ ^
deviceQuery.cpp.hip:231:23: error: no member named 'unifiedAddressing' in 'hipDeviceProp_t'
          deviceProp.unifiedAddressing ? "Yes" : "No");
```

```
          ~~~~~~~~~~ ^
deviceQuery.cpp.hip:235:23: error: no member named 'computePreemptionSupported' in 'hipDeviceProp_t'
          deviceProp.computePreemptionSupported ? "Yes" : "No");
          ~~~~~~~~~~ ^
```

- Converting helper_cuda.h was the hardest as HIPify can't handle `cuBLAS`, `cuRAND`, `cuFFT`, `cuPARSE`, even though those APIs should be supported[6]
- It ignores preprocesor statements, so for example when a file contains `#ifdefs` it will throw errors about e.g. redefinition
- Oddities, like `__DRIVER_TYPES_H__` never being defined by HIP, nor HIPify will it to `HIP_INCLUDE_HIP_DRIVER_TYPES_H`. This down the line makes compilation not work
- There is no easy way to convert makefiles, all have to be done manually

The ROCm stack



ROCm

The whole of the stack is Open Source, from kernel driver up, most released under MIT or GPL-2 license. The only propietary part is the GPU firmware.

You can of course download packages for supported OSes from AMD, but the whole differentiating factor from CUDA is it being open, so let's review how easy it is to compile it from source and package it for a distribution.

Building and packaging ROCm from source

https://github.com/JacekJagosz/rocm-Solus

Problems when compiling

```
hipcc vectorAdd.cu.hip -o vectorAddRX580 --rocm-device-lib-path=/usr/lib64/amdgcn/bitcode/ -DROCM_PATH=/usr -I ~/Pobrane/cuda-
        samples-11.6/Common/

hipcc vectorAdd.cu.hip -o vectorAddRX580v2 --rocm-device-lib-path=/usr/lib64/amdgcn/bitcode/ -DROCM_PATH=/usr -I
        ../../CommonConvertedToHIP_5.4.2/

hipcc commonKernels.cu.hip helperFunctions.cpp.hip matrixMultiplyPerf.cu.hip --rocm-device-lib-path=/usr/lib64/amdgcn/bitcode/ -
        DROCM_PATH=/usr -I ../CommonConvertedToHIP_5.4.2/
```

- Even odder things

```
 matrixMultiplyPerf.cu.hip:318:23: error: no matching function for call to 'hipHostGetDevicePointer'
       checkCudaErrors(hipHostGetDevicePointer(&dptrA, hptrA, 0));
```

- Can't find an identifier for some reason:

```
error: use of undeclared identifier 'findCudaDevice'
 int dev = findCudaDevice(argc, (const char **)argv);
```

Problems when running

- cudaProfilerStart and cudaProfilerStop are deprecated but exposed by torch.cuda.cudart(). HIP has corresponding functions stubbed out, hipProfilerStart and hipProfilerStop, but they return hipErrorNotSupported, which causes the program to stop. I think if they are

stub functions anyways they should return success. `code=801(hipErrorNotSupported) "hipProfilerStart()"`: https://github.com/pytorch/pytorch/pull/82778 It is CUDA's problem too, as even their own samples are using a deprecated API, which is still implemented in CUDA but AMD didn't bother.

Performance

First APU results, 37% performance of the RTX 4000, with only 27.4% of the theoretical TFLOPs, and about 10x smaller power consumption (0,4482,1802/7.119) matrixMul:

```
MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 216.03 GFlop/s, Time= 0.607 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS
```

deviceQuery:

```
Detected 1 CUDA Capable device(s)

Device 0: ""
  CUDA Driver Version / Runtime Version          50120.3 / 50120.3
  CUDA Capability Major/Minor version number:    9.0
  Total amount of global memory:                 4096 MBytes (4294967296 bytes)
MapSMtoCores for SM 9.0 is undefined.  Default to use 128 Cores/SM
MapSMtoCores for SM 9.0 is undefined.  Default to use 128 Cores/SM
  (007) Multiprocessors, (128) CUDA Cores/MP:    896 CUDADetected 1 CUDA Capable device(s)

Device 0: ""
  CUDA Driver Version / Runtime Version          50120.3 / 50120.3
  CUDA Capability Major/Minor version number:    9.0
  Total amount of global memory:                 4096 MBytes (4294967296 bytes)
MapSMtoCores for SM 9.0 is undefined.  Default to use 128 Cores/SM
MapSMtoCores for SM 9.0 is undefined.  Default to use 128 Cores/SM
  (007) Multiprocessors, (128) CUDA Cores/MP:    896 CUDA Cores
  GPU Max Clock rate:                            1900 MHz (1.90 GHz)
  Memory Clock rate:                             1933 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 1048576 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(16384), 2D=(16384, 16384), 3D=(16384, 16384, 8192)
  Total amount of constant memory:               4294967296 bytes
  Total amount of shared memory per block:       65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     64
  Maximum number of threads per multiprocessor:  2560
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 1024)
  Max dimension size of a grid size    (x,y,z): (2147483647, 2147483647, 2147483647)
  Maximum memory pitch:                          4294967296 bytes
  Texture alignment:                             256 bytes
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Device has ECC support:                        Disabled
  Device supports Managed Memory:                No
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 50120.3, CUDA Runtime Version = 50120.3, NumDevs = 1
Result = PASS Cores
  GPU Max Clock rate:                            1900 MHz (1.90 GHz)
  Memory Clock rate:                             1933 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 1048576 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(16384), 2D=(16384, 16384), 3D=(16384, 16384, 8192)
  Total amount of constant memory:               4294967296 bytes
  Total amount of shared memory per block:       65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     64
  Maximum number of threads per multiprocessor:  2560
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 1024)
  Max dimension size of a grid size    (x,y,z): (2147483647, 2147483647, 2147483647)
  Maximum memory pitch:                          4294967296 bytes
  Texture alignment:                             256 bytes
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Device has ECC support:                        Disabled
  Device supports Managed Memory:                No
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 50120.3, CUDA Runtime Version = 50120.3, NumDevs = 1
Result = PASS
```

Conclusions

HIPify is having a crazy high development pace, as seen by number of commits

- `/tmp/helper_cuda.h-978032.hip:63:3: warning: 'cuGetErrorName' is experimental in 'HIP'; to hipify it, use the '--experimental' option.`

Ideas for improvements:

- HIPify only changes function names, HIP is reimplementing CUDA functions anyways. It shouldn't need installation of CUDA SDK, it necesitates propietary software, limits OS compatibility and makes the whole process more complicated for no good reason. Al least only the headers should be necessary, not relying on LLVM and its CUDA detection
- ROCm needs to be steadily improved to help with packaging on all distros, not just by Debian and Arch maintainers, but AMD devs need to help too (more directly)
- hardware compatibility is a lot worse than with CUDA, especially support duration. It is a massive issue for desktop use when you want to have only one version for all users, not a lot of fragmentation. Separate kernels for all familis don't help either.

Sources: [^1]: https://developer.nvidia.com/cuda-llvm-compiler

---

1. https://github.com/ROCm-Developer-Tools/HIPIFY__
2. https://www.llvm.org/docs/CompileCudaWithLLVM.html#dialect-differences-between-clang-and-nvcc__
3. https://github.com/ROCm-Developer-Tools/HIPIFY__
4. https://github.com/llvm/llvm-project/blob/7e629e4e888262abd8f076512b252208acd72d62/clang/lib/Driver/ToolChains/Cuda.cpp#L123__
5. https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v11.6.tar.gz__
6. https://github.com/ROCm-Developer-Tools/HIPIFY#-supported-cuda-apis__