Lattice QCD GPU Inverters on ROCm Platform

Yujiang Bi[1,*], *Yi* Xiao[2], *WeiYi* Guo[3], *Ming* Gong[1,4], *Peng* Sun[5], *Shun* Xu[6], and *Yi-bo* Yang[7,**]

[1]Institute of High Energy Physics, Chinese Academy of Science, Beijing 100049, China

[2]The Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

[3]Department of Physics, University of Warwick, Coventry, CV4 7AL, United Kingdom

[4]School of Physics, University of Chinese Academy of Sciences, Beijing 100049, China

[5]Nanjing Normal University, Nanjing, Jiangsu, 210023, China

[6]Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China

[7]CAS Key Laboratory of Theoretical Physics, Institute of Theoretical Physics, Chinese Academy of Sciences, Beijing 100190, China

Abstract. The open source ROCm/HIP platform for GPU computing provides a uniform framework to support both the NVIDIA and AMD GPUs, and also the possibility to porting the CUDA code to the HIP-compatible one. We present the porting progress on the Overlap fermion inverter (GWU-code) and also the general Lattice QCD inverter package - QUDA. The manual of using QUDA on HIP and also the tips of porting general CUDA code into the HIP framework are also provided.

1 Introduction

The engineering and energy efficiency constraints push the modern supercomputer architecture to multi-level parallelism, and heterogeneous computing architectures such as CPU+GPU are widely used in top 500 supercomputers[1], including the most recent fastest Summit and Sierra. Most of the performance of them comes from the NVIDIA GPU V100, and efficient codes are essential to benefit various science computation like Lattice Chromodynamics (Lattice QCD) from those machines.

As we know that QCD is the dominant theory describing strong interaction, Lattice QCD[2, 3] is the discretized version of QCD based on Euclidean space-time as shown in Fig.1 instead of Minkowski space-time. Lattice QCD calculation requires massive computation resources, and increasingly relies on GPU acceleration.

The CUDA platform developed by NVIDIA is wildly adopted for accelerating computation in many fields as well as Lattice QCD. There are already quite a few packages that supports CUDA with good performance and also multi-GPU scaling, including QUDA (for most of

*e-mail: biyujiang@ihep.ac.cn
**e-mail: ybyang@itp.ac.cn

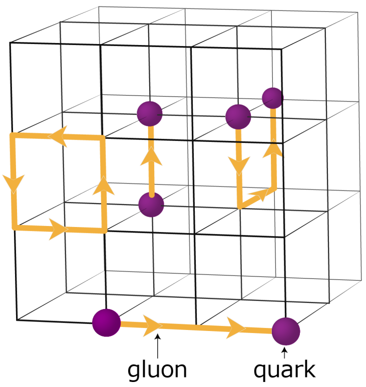

Figure 1. Lattice QCD discretization illustration. Quark fields lie on the grid points, and gluon are represented by links connecting two points.

the fermion actions)[4–6], GRID (for the domain wall fermion and etc.) [7], GWU-code (for the overlap and clover fermion)[8, 9], and so on.

On the other hand, few efficient codes support AMD GPU except some effort with OpenCL (e.g. CL2QCD[10]), while the peak performance of the AMD GPU have caught up and the E-flops supercomputer “Frontier" with AMD GPU will be built in US by 2021. An open-source programming platform ROCm(Radeon Open Compute), therefore was promoted by AMD in recent years to improving the experience of programming on its own CPU/GPUs.

ROCm[11] platform is "*the first open-source HPC/Hyperscale-class platform for GPU computing*", still young and under active development. It supports kinds of languages such as HIP[12], OpenCL and Python, and supports both AMD’s and NVIDIA’s GPUs, as shown in Fig. 2. One can write one set of codes using ROCm/HIP to support both AMD’s and NVIDIA’s hardwares, which is convenient, and can port CUDA codes into ROCm/HIP with the help of official porting tools like hipify-perl[13].

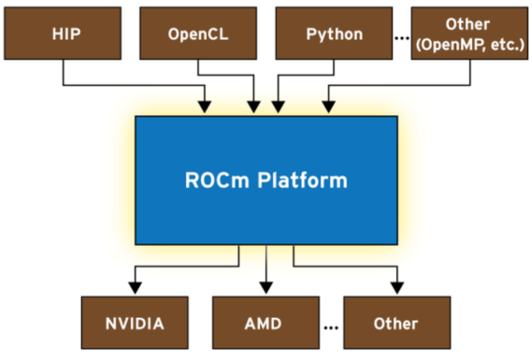

Figure 2. ROCm platform supports languages such as HIP, OpenCL and Python etc., and supports hardwares of AMD and of NVIDIA.

Based on this, writing codes from scratch and implementing hundreds of features needed by the lattice QCD calculation, could be avoided by porting the existed CUDA codes. And in this proceeding, we will present our finding on using the package GWU-code and QUDA on the AMD GPU through the ROCm/HIP platform, with a summary on the known issues.

2 Porting Prerequisites

Before we started porting QUDA and GPU-Code, we analyzed the external dependent libraries. The QUDA library is officially supported by NVIDIA and depends on CUDA and many third-party CUDA libraries such as cufft, curand and cub. Fortunately there is an HIP version for each library. Table.1 shows the correspondence between CUDA and ROCm/HIP. Note that the Eigen library is partially supporting ROCm/HIP, and we had to provide some de-

Table 1. CUDA modules used in QUDA and GWU-code and corresponding modules in ROCm, more details see ref. [14].

CUDA	cublas	cub	thrust	cufft	curand	cusparse	Eigen	···
ROCm/HIP	hipblas	rocprim	rocthrust	rocfft	hiprand	hipsparse	Eigen	···

vice functions needed by QUDA, most of which are no difference with host versions. GWU-Code however, depends on thrust only, which seems easier than QUDA to porting.

3 Porting Progress

In this section, we will discuss the porting progress and problems we encountered when we ported QUDA and GWU-Code. Note that ROCm/HIP is a young and active project, some problems may be solved by upgrading ROCm/HIP, while some additional problems may arise consequently.

3.1 Porting tips and compiling manual of QUDA

Generally, the porting can be separated into 3 stages: convert the code with hipify-perl, patch the codes manually to satisfy the requirement of compiler, replace the unsupported features to avoid the runtime crash. Let us take the porting of QUDA as example:

1. Convert the code with hipify-perl. The hipify-perl is a perl script to map the name of the CUDA functions to that of their HIP counterpart, and also the CUDA header files. If the script meets some unknown words starting with "cu", message “warning:... : unsupported device function" will be thrown out, while the conversion will continue.
2. Patch the codes manually to satisfy the requirement of compiler. Currently, QUDA is compiled by hip-clang, instead of HCC (Heterogeneous Compute Compiler). HC is a C++ AMP syntax language with HSA Extend[15]. HCC will translate HIP kernel syntax into C++ AMP syntax by using functional or macro grid launch, while certain QUDA device functions that use complicated class and template will cause syntax or runtime error. In the other hand, hip-clang is a hip kernel syntax supported LLVM frontend as shown in Fig.3. By setting the environment variable HIP_PLATFORM to clang, hip-clang will take over the compiling and compile CUDA-like syntax source code to LLVM IR directly[16], and then the AMD GPU backend of LLVM will compile the LLVM IR to binary. The patches we applied include:

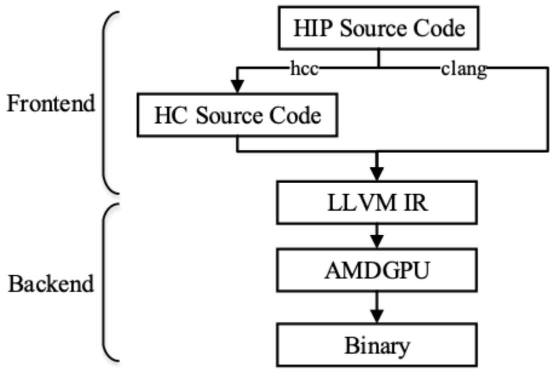

Figure 3. HIPCC compilation process illustration. The clang compiler skips the step to generate the HC source code.

   (a) Set CMAKE_CXX_SYSROOT_FLAG_CODE to add the .cu suffix to the CMAKE_CXX_SOURCE_FILE_EXTENSIONS, and then use hip-clang to compile both the .cpp and .cu files. Note that the flag “-g" should be avoid and “-_STRICT_ASNI_ -O3" is necessary to make the code works well. Compile clover_deriv_quda.cu and gauge_stout.cu will crash the compiler and add the flag "-fno-inline" would be a choice to avoid it.

   (b) Fix the functions with different prototype, likes hipGetErrorString, hipGetErrorName, hipPointerGetAttributes, hipMemcpyHtoDAsync and etc. At the same time, some of the functions likes blasCgetrfBatched and blasCgetriBatched are replaced with the CPU version.

   (c) Add the __host__ flags in front of the __device__ functions which used in the CPU code.

   (d) Rewrite the ptx code into the normal C++ codes.

   (e) Move the declare of the shared memory into the body of the functions, as it can not be located in the link stage.

3. Replace the unsupported features to avoid the runtime crash. The major changes include:

   (a) Suppose the memory Type is host if hipPointerGetAttributes return an error, and comment out checkCudaError() in the constructors.

   (b) Limited the maximum threads used in the tuning to 512 or even smaller number, if certain function crashes on the AMD GPU with more threads.

   (c) Use the global function to copy the constant array needed by multi_blas_kernel to the global memory, as the function hipmemcpyToSymbol doesn’t work correctly:

```
__global__ void set_Amatix(signed char *ref) {
        int idx = blockIdx.x * blockDim.x + threadIdx.x;
        if(idx>=MAX_MATRIX_SIZE)
                return;
        Amatrix_d[idx]=ref[idx];
}
```

Table 2. Summary of the porting progress in term of QUDA build options. All the parts needed for a Multigrid inverter of the Clover fermion have been done.

Module name	Ported	Tested
QUDA_DIRAC_WILSON	yes	yes
QUDA_DIRAC_CLOVER	yes	yes
QUDA_CONTRACT	yes	yes
QUDA_COVDEV	yes	yes
QUDA_DIRAC_STAGGERED	yes	no
QUDA_FORCE_GAUGE	yes	no
QUDA_DIRAC_DOMAIN_WALL	no	no
QUDA_DIRAC_TWISTED_MASS	no	no
QUDA_DIRAC_TWISTED_CLOVER	no	no
QUDA_DIRAC_CLOVER_HASENBUSCH	no	no
QUDA_DIRAC_NDEG_TWISTED_MASS	no	no
QUDA_LINK_ASQTAD	no	no
QUDA_LINK_HISQ	no	no
QUDA_FORCE_HISQ	no	no
QUDA_GAUGE_TOOLS	no	no
QUDA_GAUGE_ALG	no	no
QUDA_DYNAMIC_CLOVER	no	no

```
signed char *A_d;
hipMalloc(&A_d, MAX_MATRIX_SIZE);
hipMemcpy(A_d,A,MAX_MATRIX_SIZE,hipMemcpyHostToDevice);
set_Amatix<<<256,MAX_MATRIX_SIZE/256>>>(A_d);
hipDeviceSynchronize(); hipFree(A_d);
```

(d) In the function multiblas, multireduce and ComputeVUV, passing the parameters though the argument class of the global function like:

```
__global__ func(Arg arg){...}
```

can make the performance to be extremely low. Such a problem can be avoided by copying the argument class to the GPU memory first, and using its reference as the argument:

```
__global__ func(Arg &arg){...}
        ...
        Arg *arg_d;
        hipMalloc(&arg_d, sizeof(Arg));
        hipMemcpy(arg_d,&arg, sizeof(Arg),hipMemcpyHostToDevice);
        func(*arg_d);
        hipDeviceSynchronize();
        hipFree(arg_d);
```

Other minor changes can be found in the present branch on the github:

https://github.com/lattice/quda/tree/rocm-devel.

In term of the QUDA building options, the porting progress are listed in Tab.(2)

Table 3. The single precision peak performance and memory bandwidth of V100 and MI60, v.s. the D-slash performances using the GWU-code and QUDA with the same single precision. The SU(3) is suppressed to 12 real numbers in both the GWU-code and QUDA cases to save the memory bandwidth. Note that the tests on V100 uses CUDA, instead of HIP.

	Peak FP32 (TFlops)	Bandwidth (TB)	GWU-code (TFlops)	QUDA (TFlops)
Nvidia V100	14.7	0.9	0.80	1.11
AMD MI60	14 (for PCIe)	up to 1.0	0.54	0.48

3.2 GWU-Code Porting

Comparing to QUDA, porting GWU-code is much simpler. GWU-code use the macro to generate the D-slash GPU kernel without any device functions, and implement the vector operator on GPU with the CUDA Thrust [5, 6]. Thus one just need to replace the Thrust library with the rocthrust after the code has been converted with hipify-perl. The function thrust::reduce can be very slow with double precision in certain version, but it can be replaced by the other functions with normal performance.

4 Performance Test

Our test is based on AMD MI60 GPU and Nvidia V100 GPU. The D-slash performance is summarized in Tab.3. The QUDA D-slash performance is somehow lower as this kernel is much more complicated and then can not be fully optimized with hip-clang at present.

As one practical application of QUDA, the multigrid[17] inverter, one useful and efficient lattice invertor, works correctly while the performance is not very promising. With the largest $96^3 \times 192$ lattice we tested, the total performance with 324 MI60 GPUs is around 10 TFlops, using a 3-level multigrid layouts (4,4,4,4) and (2,2,2,2), which is not as good as we expected before. The performance of multigrid inverter on ROCm is much lower than that on CUDA platform, but could be optimized further.

In the GWU-code side, 200 pairs of the Overlap eigensystem of the HYP smeared $24^3 \times 64$ RBC configuration at lattice spacings a=0.11fm can be generated with 4 MI60 GPU within 4 hours, and the similar calculation on E5-2698v3 at 2.3 GHz requires 1024 cores up to 2 hours. The result is acceptable in a way, and strong scaling and weak scaling are reasonable. But again, it's much lower compared to that on CUDA platform. We still need more effort to improve the performance. The test with larger lattice size is in progress.

5 Summary

In Summary, we ported two Lattice QCD CUDA packages, QUDA and GWU-code to the AMD GPU platform ROCm using HIP. The performance is around half of that on CUDA platform when the memory bandwidth of both are similar. The multigrid inverters of QUDA works correctly with the lattice as large as $96^3 \times 192$, and the overlap eigensystem can be calculated correctly with GWU-code. We will try to optimize the performance and scaling in the further study. All the present tests using HIP are performed on AMD GPUs, and will be investigated on NVIDIA GPUs as well.

This work is supported in part by the **National Key Research and Development Program of China (No. 2017YFB0203200)** and the Strategic Priority Research Program of Chinese

Academy of Sciences (Grant No. XDC01040100). M. Gong is supported in part by the National Science Foundation of China (NSFC) under the project No. 11775229, P. Sun is supported by Natural Science Foundation of China under grant No. 11975127, Y. Yang is supported in part by Chinese Academy of Science CAS Pioneer Hundred Talents Program.

References

[1] Top 500 site:https://www.top500.org/lists.

[2] H. J. Rothe, Quantum Gauge Theories: An Introduction, World Scientific, 1997.

[3] Rajan Gupta. Introduction to Lattice QCD. arXiv:hep-lat/9807028

[4] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs. Comput. Phys. Commun., 181:1517–1528, 2010.

[5] R. Babich, M. A. Clark, B. Joo, G. Shi, R. C. Brower, and S. Gottlieb. Scaling Lattice QCD beyond 100 GPUs. In SC11 International Conference for High Performance Computing, Networking, Storage and Analysis Seattle, Washington, November 12-18, 2011, 2011.

[6] M. A. Clark, Blint Jo, Alexei Strelchenko, Michael Cheng, Arjun Gambhir, and Richard Brower. Accelerating Lattice QCD Multigrid on GPUs Using Fine-Grained Parallelization. arXiv:1612.07873 [hep-lat].

[7] Peter Boyle, Azusa Yamaguchi, Guido Cossu, and Antonin Portelli. Grid: A next generation data parallel C++ QCD library. arXiv:1512.0348 [hep-lat].

[8] A. Alexandru, C. Pelissier, B. Gamari, and F. Lee. Multi-mass solvers for lattice QCD on GPUs. J. Comput. Phys., 231:1866–1878, 2012.

[9] Andrei Alexandru, Michael Lujan, Craig Pelissier, Ben Gamari, and Frank X. Lee. Efficient implementation of the overlap operator on multi-GPUs. In Proceedings, 2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC'11): Knoxville, Tennessee, July 19-20, 2011, pages 123–130, 2011.

[10] Owe Philipsen, Christopher Pinke, Alessandro Sciarra, and Matthias Bach. CL2QCD - Lattice QCD based on OpenCL. PoS, LATTICE2014:038, 2014.

[11] ROCm: https://rocm.github.io/

[12] HIP: https://github.com/ROCm-Developer-Tools/HIP

[13] hipify tool: https://github.com/ROCm-Developer-Tools/HIPIFY

[14] Corresponding libs between ROCm and CUDA: HIP Porting Guide

[15] HSA Foundation website: http://www.hsafoundation.com/standards/

[16] J Wu, A Belevich, and E Bendersky. gpucc: an open-source gpgpu compiler. Proceedings of the 2016 International Symposium on Code Generation and Optimization. ACM, 2016: 105-116, 2016.

[17] M. A. Clark, A. Strelchenko, M. Cheng, A. Gambhir, and R. Brower, "Accelerating Lattice QCD Multigrid on GPUs Using Fine-Grained Parallelization," International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2016 [arXiv:1612.07873 [hep-lat]].