Möbius domain wall fermion method on QUDA

Hyung-Jin Kim[*]
Brookhaven National Laboratory, USA
E-mail: hjkim@bnl.gov

Taku Izubuchi
RIKEN-BNL Research Center, USA
E-mail: izubuchi@quark.phy.bnl.gov

Möbius Domain Wall Fermion (DWF) method is an extended form of Shamir's domain wall fermion action, which provides the same overlap action correspondence in the limit of $L_s \to \infty$. Furthermore, Möbius DWF has an advantage in smaller size of chiral violation effect coming from residual mass compared with Shamir's DWF. At $\alpha = O(L_s)$, $O(1/L_s)$ of M_{res} error in Shamir's DWF can be significantly reduced in Möbius DWF method[8]. This chiral enhancement on Möbius operator enables us to use the smaller 5th dimensional size of lattice data without scarifying the precision.

Furthermore, smaller size of lattice data is very helpful to compute the DWF algorithm on GPU environment. In last 5 years, GPU has been widely used in lattice QCD applications and succeeded in high performance computation. However, limited size of available device memory is still a weak point of GPU and which makes the DWF computation especially difficult. By using the Möbius DWF method on GPU, we can relieve this limit and this is the reason why we have implemented the Möbius DWF operator on the QUDA library. For preconditioned conjugate gradient (CG) update, 4D even-odd preconditioning method is used but the detail of the algorithms are modified for GPU optimization.

31st International Symposium on Lattice Field Theory - LATTICE 2013
July 29 - August 3, 2013
Mainz, Germany

[*]Speaker.

1. Introduction

There are several efforts to implement the GPU environment on the QCD Applications[1][2][3]. QUDA[1] is one of well-known physics library based on the CUDA to accelerate the QCD algorithms. It provides officially 5 different types of fermion operators (wilson, clover, twisted mass, asqtad, domain wall) with $O(10^2)$ GFLOPS of performance per GPU on multi node GPU platform.

Möbius DWF method[4][5] is an extended form of Shamir type domain wall fermion[9][10]. In the limit of infinite 5-th dimensional size, Möbius DWF operator exactly corresponds to the overlap action. Usually, Möbius DWF has an advantage in smaller chiral violation effect compared with Shamir type DWF[8]. Therefore, by using the Möbius method, one can achieve better residual mass values even in the smaller 5th dimensional size of lattice data. This is definitely beneficial for the applications which is strongly dominated by chiral symmetry breaking effect.

Furthermore, the bigger and the finer lattice requires huge size of computational memory also. This situation is especially severe in DWF simulation. Unfortunately, the size of GPU memory is small compared with system memory so, to handle such a huge size of lattice data, we need to use many GPUs. But in that case, multi GPU scalability decreases as we increase the number of GPUs because of the large surface data communication[2]. So if we can reduce the physical data size without sacrificing the precision by Möbius DWF method, it will be very useful to save the memory usage on GPU computation.

This is our main motivation of this research. There are several developed packages using Möbius DWF method[6][7] on CPU based computer system. On the basis of QUDA library, we have implemented the Möbius type of domain wall fermion CG inverter.

2. Möbius domain wall Fermion method

The explicit form of Dirac equation on Möbius DWF is given in Eq. (2.1)[8].

$$\begin{aligned}\bar{\psi}D^{DW}(m)\psi = \sum_{s=1}^{L_s}\bar{\psi}_s D_+^{(s)}\psi_s + \sum_{s=2}^{L_s}\bar{\psi}_s D_-^{(s)}P_+\psi_{s-1} + \sum_{s=1}^{L_s-1}\bar{\psi}_s D_-^{(s)}P_-\psi_{s+1} \\ -m\bar{\psi}_1 D_-^{(1)}P_+\psi_{L_s} - m\bar{\psi}_{L_s}D_-^{(L_s)}P_-\psi_1 \end{aligned} \tag{2.1}$$

where

$$D_+^{(s)} = b_5 D^{wilson}(M_5) + 1, \quad D_-^{(s)} = c_5 D^{wilson}(M_5) - 1 \tag{2.2}$$

The main difference with Shamir type DWF is new parameter b_5 and c_5. If b_5 is set to 1 and c_5 is equal to 0, this equation gives the exactly same result with Shamir‘s DWF equation.

Because the Möbius DWF form of equation is different with Shamir type of equation, preconditioning form of dslash operator should be also changed. Like other lattice simulation, preconditioned dslash operator has been well known prescription to reduce the computation and data size. Instead of calculating a full size of vector, we can get the same result by calculating the even site or odd site only. Here, even-odd means the sum of target lattice position indices is even or odd number.

[1]see official web-page, https://github.com/lattice/quda

In Shamir type of DWF case, even-odd condition is normally determined by summing over all 5 dimensional indices. But to calculate the Eq. (2.1), we need to use the complete data set in 5th dimensional index. Therefore, Möbius DWF uses only 4 of (x,y,z,t) lattice indices for even-odd preconditioning strategy and 5th s-index is given as complete set. Then, 4D even-odd form of Dirac equation can be written as Eq. (2.3).

$$D^{dwf} = \begin{pmatrix} M_5 & -\kappa_b M^4_{eo} \\ -\kappa_b M^4_{oe} & M_5 \end{pmatrix} \tag{2.3}$$

where

$$M^4_{eo,oe} = \not{D}^W_{x,y}(b_5\delta_{s,t} + c_5\not{D}^5) \quad , \quad M_5 = 1 + \frac{\kappa_b}{\kappa_c}\not{D}^5$$

and $\not{D}^W_{x,y}$, $\not{D}^5_{s,s'}$ are defined by,

$$\not{D}^W_{x,y} = \sum_\mu [(1+\gamma_\mu)U^\dagger_{x-\mu,\mu}\delta_{x-\mu,y} + (1-\gamma_\mu)U_{x,\mu}\delta_{x+\mu,y}]$$

$$\not{D}^5_{s,s'} = P_R\delta_{s-1,t} + P_L\delta_{s+1,t} - m_f P_R \delta_{s,0}\delta_{t,L_s-1} - m_f P_L \delta_{s,L_s-1}\delta_{t,0}$$

Projection operator $P_{R,L}$ and coefficients $\kappa_{b,c}$ are given,

$$P_R = (1+\gamma_5)/2 \ , \ P_L = (1-\gamma_5)/2$$

$$\kappa_b^{-1} = 2(b_5(4-M)+1) \ , \ \kappa_c^{-1} = 2(c_5(4-M)-1)$$

The diagonalized form of preconditioned matrix can derived by multiplying transformation matrices on both side of Eq. (2.3)

$$\begin{aligned} \tilde{D}^{dwf}_{PC_{4D}} &= \begin{pmatrix} 1 & \kappa_b M^4_{eo} M_5^{-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} M_5 & -\kappa_b M^4_{eo} \\ -\kappa_b M^4_{oe} & M_5 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \kappa_b M_5^{-1} M^4_{oe} & M_5^{-1} \end{pmatrix} \\ &= \begin{pmatrix} M_5 - \kappa_b^2 M^4_{oe} M_5^{-1} M^4_{eo} & 0 \\ 0 & \delta_{ee} \end{pmatrix} \end{aligned} \tag{2.4}$$

Practically, we will calculate the odd-odd part of preconditioned matrix $\tilde{D}^{dwf}_{PC_{4D}}$ and from this result, we can easily get the even-even component also. Odd-odd part of Eq. (2.4) can be computed by combinations of M_5, $M^4_{eo,oe}$ and M_5^{-1} matrix operation. In the following section, we will explain how we implement each operations with more detail.

3. Möbius implementation on QUDA

Most of the operations for preconditioning method can be easily implemented by modifying the original QUDA code. Basically, M_5 and $M^4_{eo,oe}$ subroutines are expressed as combinations of $\not{D}^W_{x,y}$ and $\not{D}^5_{s,s'}$. Because both operators are also used in Shamir‘s type of DWF method, there are already well implemented subroutines for those operations in QUDA library. Based on these conventional QUDA subroutines, new $\not{D}^4_{pre}$ is defined,

$$\not{D}^4_{pre} \equiv b_5\delta_{s,t} + c_5\not{D}^5 \tag{3.1}$$

	FLOP/site	Bytes/site (single precision)
$\not{D}^W_{x,y}$	1320	1440 (8 Read, 1 Write, 1 Gauge)
$\not{D}^4_{pre}$	168 + 48/Ls	384 (3 Read, 1 Write)
M_5^{-1}	3Ls + 141	192 (1 Read, 1 Write)
M_5^{XPAY}	192 + 48/Ls	480 (4 Read, 1 Write)

Table 1: The number of floating point operations and data size per lattice site on each operators. $M_5^{XPAY}(v_1, v_2) \equiv M_5 v_1 - \kappa^2 v_2$. Ls means lattice size of 5th dimension.

And then, $M^4_{eo,oe}$ operation can be simply noted as $M^4_{eo,oe} = \not{D}^W_{x,y}\not{D}^4_{pre}$. Therefor, actual preconditioned form of matrix operation will become $\tilde{D}^{dwf}_{oo} = M_5 - \kappa_b^2 \not{D}^W_{oe}\not{D}^4_{pre} M_5^{-1} \not{D}^W_{eo}\not{D}^4_{pre}$.

In the $\not{D}^W_{x,y}$ of Table 1, the FLOP/Bytes ratio is given 0.9 but the FLOPS/bandwidth ratio of C2050 GPU is about 9 in single precision. So we can consider that the performance of Dirac operator on GPU is highly bounded by memory access bandwidth not by the FLOPS value. And in the rest of the matrix operations in Table 1, the situations are the same. The major factor of the bottle-neck in each operation is also the memory bandwidth.

But M_5^{-1} is totally new operation in QUDA program. M_5^{-1} can be divided into left-hand and right-hand part as shown in Eq. (3.2) and again, each part can be written as a multiplication of two matrices (Eq. (3.3)).

$$M_5^{-1} = M_{5,R}^{-1} P_R + M_{5,L}^{-1} P_L \tag{3.2}$$

$$\begin{aligned} M_{5,R}^{-1} = \begin{pmatrix} 1 & 0 & \dots & \kappa m_f \\ \kappa & 1 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1} &= \begin{pmatrix} 1 & 0 & \dots & 0 \\ \kappa & 1 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1+(-\kappa)^{Ls} & (-\kappa)^{Ls-1} & \dots & -\kappa m_f \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \\ &\equiv A^{-1} B^{-1} \end{aligned} \tag{3.3}$$

Actually, matrix A and B can be inverted explicitly. Practically, we calculate the output vector (w) elements from multiplying the M_5^{-1} to source vector (v). We can solve this matrix inversion in two different approaches, one is solving the output vector elements in sequential way and the other is calculate the whole elements simultaneously from explicit matrix inversion. Each method has an advantage depending on the computational environment.

The first sequential method is used in the CPU based Möbius DWF programs[6][7],

$$\begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_{Ls-1} \end{pmatrix} = M_{5,R}^{-1} \begin{pmatrix} v_0 \\ v_1 \\ \dots \\ v_{Ls-1} \end{pmatrix} = \begin{pmatrix} \frac{v_0 - 2\kappa m_f v_{Ls-1} - \dots - (2\kappa)^{Ls-1} m_f v_1}{1+(2\kappa)^{Ls} m_f} \\ 2\kappa w_0 + v_1 \\ \dots \\ 2\kappa w_{Ls-2} + v_{n-1} \end{pmatrix} \tag{3.4}$$

From Eq. (3.4), the first element w_0 can be explicitly calculated from input vector elements (v_i) and $M_{5,R}^{-1}$ matrix. And next element w_1 (or w_i) element can be derived from the information of w_0 and v_1 (or w_{i-1} and v_i) . In that way, we can get the whole data elements and it also remarkably reduces the number of arithmetic operations from matrix-vector operation ($O(n^2)$) to vector-vector operation ($O(n)$).

But because of the data dependance, this method is not appropriate for the parallel computation on GPU. In the GPU computation approach, each vector elements should be calculated in parallel way, so we are using the explicit form of M_5^{-1} which is given in Eq. (3.5).

$$\underbrace{\begin{bmatrix} \text{thread 0} \\ \text{thread 1} \\ \text{thread 2} \\ \cdots \end{bmatrix}}_{\text{same thread block}} \overset{\text{do}}{\Rightarrow} M_{5,R}^{-1} v = \frac{1}{1+(-\kappa)^{Ls} m_f} \underbrace{\begin{pmatrix} 1 & -(-\kappa)^{Ls-1} m_f & -(-\kappa)^{Ls-2} m_f & \ldots \\ -\kappa & 1 & -(-\kappa)^{Ls-1} m_f & \ldots \\ (-\kappa)^2 & -\kappa & 1 & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix}}_{\underset{\Rightarrow}{\text{summation direction on loop}}} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \ldots \end{pmatrix} \tag{3.5}$$

Even though this method increases numerical costs from $O(n)$ to $O(n^2)$, we expect that the GPU arithmetic performance is good enough to handle such a number of arithmetic operations. On the other hand, explicit calculation of M_5^{-1} operation decreases the amount of data access from 2 read, 1 write to 1 read, 1 write, so we can expect that this strategy could be useful for GPU implementation.

4. Performance

For the performance test, we have used 4 of Nvidia C2050 and K20m GPUs. And QUDA (Ver 5.0), Columbia Physics System (CPS, Ver 5.0.23) library is used for physics measurement. Our main target is optimizing the $\tilde{D}_{oo}^{dwf}$ operation. According to the Table 1, it can be calculated by total 33 times of vector access, 4 gauge data access and 3309 + 3Ls + 144/Ls number of floating point calculations per lattice site in Möbius case. Shamir type of Dirac operation needs total 23 times of vector access, 4 gauge data access and 2880 + 96/Ls number of floating point calculations per lattice site.

Therefor, the FLOPS/bandwidth ratio of Shamir type of Dirac operation is slightly bigger ($\sim$ 10% at Ls = 8) than Möbius case. Because the bottle-neck of CG algorithm is in the memory bandwidth, it is reasonable that Shamir type Dirac operator will show about 10% better performance than the Möbius type operator.

$24^3 \times 64 \times 8$	MDWF	DWF
C2050	390	470
K20m	840	924

Table 2: preconditioned Dirac operator performance on Möbius DWF and DWF scheme. unit is GFLOPS. Number is 4 GPUs performance.

Tested performance result on Möbius and Shamir type of operator is given in Table 2. In actual computation, Möbius operator is slower about 20% than Shamir type of operator. It looks the performance of Möbius DWF operator is much slower than our expectation. This is because the efficiency of embodied M_5^{-1} operator is not good enough yet.

Table 3 shows execution time of each matrix-vector operation. We can check whether implemented sub routines are optimized enough or not by comparing the times with theoretical peak performance. *Comp. time* at 3rd column of Table 3 means computational time for floating point operations in peak performance. And *Data access* at 4th column of Table 3 means data Input-Output (IO) time from GPU memory to GPU in peak memory bandwidth. For example, in $\not{D}^{W}_{x,y}$ case, we can calculate these values as follow.

$$\text{Comp. Time} : 1320 \times (\text{local volume})/2(\text{Even-Odd})/1\text{TFLOPS} \sim 1.2\text{ms}$$

$$\text{Data Access} : (9(\text{spinor}) + 8(\text{gauge link}^{2})) \times 4(\text{bytes}) \times (\text{local } V)/2(\text{EO})/126(\text{GB/sec})^{3} \sim 7.9\text{ms}$$

As you can see in Table 3, experimental time and ideal data access time of $\not{D}^{W}_{x,y}$ is very similar which means most of the times are used for data transfer between GPU and GPU memory and half of the computational time is overlapped with the data IO times. This is almost maximum performance which we can get unless reducing the data size. In $\not{D}^{4}_{pre}$ and M_5^{XPAY} cases, the situations are similar. Even though the overlap between the data IO and arithmetic computation is not obvious in those cases, at least ∼80% (in M_5^{XPAY}) of execution times are used for data access. Considering the *Data access* times are calculated with maximum memory bandwidth, we can conclude the these two operations are also optimized enough.

Operator type	Exp.	Comp. time (peak)	Data access (peak)
$\not{D}^{W}_{pre}$	3.0	0.15	2.7
$\not{D}^{W}_{x,y}$	8.6	1.2	7.9
M_5^{-1}	7.1	0.15	1.4
M_5^{XPAY}	4.1	0.18	3.4
total $\tilde{D}^{dwf}_{oo}$	~ 34	~ 3.0	~ 26

Table 3: Time table of the subroutines in preconditioned Dirac operator. unit is milli second. There are 2 of $\not{D}^{W}_{pre}$ and $\not{D}^{W}_{x,y}$ operators, 1 of M_5^{-1} and M_5^{XPAY} operators per $\tilde{D}^{dwf}_{oo}$. All measurements are done with single precision.

But unlike these operators, the M_5^{-1} operator shows very huge discrepancy in experimental result and ideal prediction. We suspect that caching failure is happening during the data IO sequence. In Table 1, we assumed that 2 (1 read, 1 write) of vector data accesses are needed for M_5^{-1} operation. But this assumption is not valid in current implemented program. In Eq. (3.5), our strategy is that assigning the each parallel thread to the corresponding row of M_5^{-1} matrix. And then, m-th thread will multiply n-th input element with (m,n) element of M_5^{-1} and add them to the *row* direction by using loop sequence. More specifically, let‘s think about the multiplication sequence of v_0 element with 0-th column of M_5^{-1} matrix. Once v_0 element is loaded to shared cache memory area, it can be accessed with small latency to every thread processor.

But if the data sharing in cache memory is not successful, unnecessary data reloading is needed and consequently, we need more data IO in the M_5^{-1} operation. If the 5th dimensional site are divided by n times in thread geometry, it means we have to read the same input vector n times. Unfortunately, the basic strategy of thread geometry in QUDA library is not appropriate to this

[2] 8 parameter SU(3) reconstruction method is applied to reduce the data IO

[3] Memory bandwidth is limited by 7/8 because of ECC

cache re-usability. The single thread block does not contains complete set of 5th dimensional sites so there are unnecessary multiple data accesses on input vector.

The one solution for this problem is changing the thread geometry to appropriate M_5^{-1}. But in this case, coalesced memory access pattern in QUDA library could be broken so we need to be very careful for this modification. Now we are testing several combination of thread block size and optimization of M_5^{-1} operator is still going on.

5. Conclusion

We have successfully implemented Möbius DWF method in QUDA library. Developing version of QUDA program can be downloaded at official QUDA developer web page. Current version of Möbius code shows 10$\sim$20% slower performance in single Dirac operation sequence than Shamir type of DWF operator. But even with developing version, user can easily achieve a TFLOPS scale of computational capability within 4 of GPUs. We hope our Möbius implementation will be useful in a future domain wall fermion research.

6. Acknowledgments

The research of Hyung-Jin Kim is supported by SciDAC-3: Searching for Physics Program (KA2401021) funded by U.S. Department of Energy.

References

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

[2] R. Babich, M. A. Clark, B. Joo, G. Shi, R. C. Brower, and S. Gottlieb, Scaling lattice QCD beyond 100 GPUs," International Conference for High Performance Computing, Networking, Storage and Analysis. arXiv:1109.2935 [hep-lat]

[3] Yong-Chull Jang, Hyung-Jin Kim, Weonjong Lee, Multi GPU Performance of Conjugate Gradient Solver with Staggered Fermions in Mixed Precision. PoS, Lattice 2011:309, 2011.

[4] Richard C. Brower, Hartmut Neff, and Kostas Orginos, Moebius fermions: Improved domain wall chiral fermions. Nucl. Phys. Proc. Suppl.,140:686-688, 2005.

[5] Richard C. Brower, Hartmut Neff, and Kostas Orginos, Moebius fermions. Nucl. Phys. Proc. Suppl.,153:191-198, 2006.

[6] Hantao Yin and Robert D. Mawhinney, Improving DWF Simulations: the Force Gradient Integrator and the Möbius Accelerated DWF Solver. PoS, LATTICE2011:051, 2011.

[7] Andrew Pochinsky, Writing efficient QCD code made simpler: QA(0). PoS, LATTICE2008:040, 2008, See http://www.mit.edu/avp/mdwf/

[8] Richard C. Brower, Harmut Neff, and Kostas Orginos, The Möbius Domain Wall Fermion Algorithm. arXiv:1206.5214v1 [hep-lat]

[9] David B. Kaplan, A method for simulating chiral fermions on the lattice. Phys. Lett.,B288:342-347, 1992.

[10] Yigal Shamir, Chiral fermions from lattice boundaries. Nucl. Phys., B406:90-106, 1993