Mobius domain wall fermion method on QUDA

BROOKHAVEN NATIONAL LABORATORY | Hyung-Jin Kim

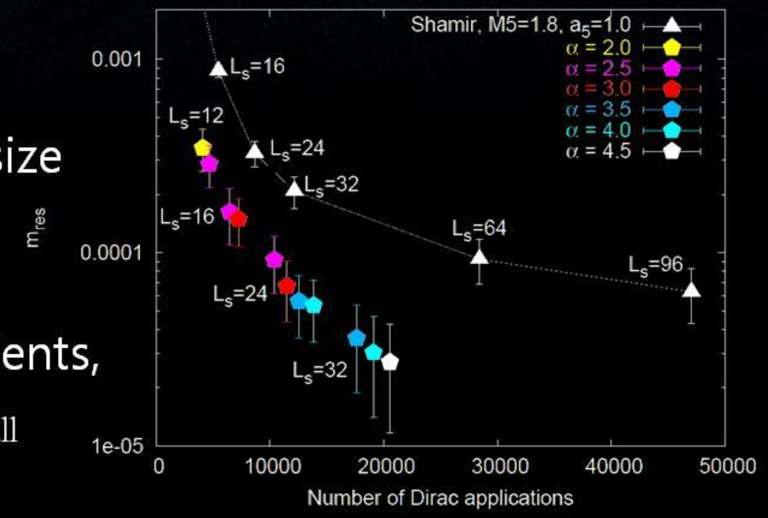
Contents

- Mobius Domain Wall Fermion
- Mobius operator on QUDA
- Performance

Mobius Domain Wall Fermion

- Mobius Domain wall Fermion?
  - Extended version of domain wall fermion
  - Reduced residual mass even in the smaller size of 5^{th} dimension
  - Under the optimized values of b_5, c_5 coefficients,

    Ref) arXiv:1206.5214, R. C. Brower et al, The Möbius Domain Wall Fermion Algorithm



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

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

Mobius DWF dirac equation

Mobius Domain Wall Fermion

- Preconditioning Method

4D Even-Odd preconditioning

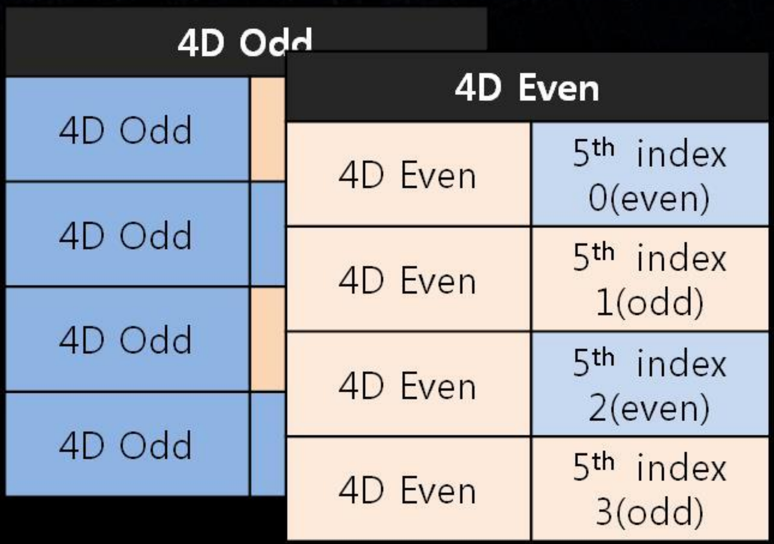
$$M^{dwf} = \begin{pmatrix} M_5 & -\kappa_b M_{eo}^{W_4} \\ -\kappa_b M_{oe}^{W_4} & M_5 \end{pmatrix}$$

※ After some transformation

⋮

$$\tilde{M}_{4D}^{dwf} = \begin{pmatrix} \delta_{ee} & 0 \\ 0 & M_5 - \kappa_b^2 M_{oe}^{W_4} M_5^{-1} M_{eo}^{W_4} \end{pmatrix}$$

4D E-O PC data structure on memory



※ $\kappa_b^{-1} = 2(b_5(4-M)+1)$

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

$P_R = (1+\gamma_5)/2$

$P_L = (1-\gamma_5)/2$

$$D\!\!\!/^W_{x,y} = \sum_\mu [(1+\gamma_\mu)U^\dagger_{x-\mu,\mu}\delta_{x-\mu,y} + (1-\gamma_\mu)U_{x,\mu}\delta_{x+\mu,y}]$$

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

$$M_{eo}^{W_4} = D\!\!\!/^W_{x,y}(b_5\delta_{s,t} + c_5 D\!\!\!/^5)$$

$$M_5 = 1 + \frac{\kappa_b}{\kappa_c} D\!\!\!/^5$$

Mobius Domain Wall Fermion

- Preconditioning Method

4D Even-Odd preconditioning

$$M^{dwf} = \begin{pmatrix} M_5 & -\kappa_b M_{eo}^{W_4} \\ -\kappa_b M_{oe}^{W_4} & M_5 \end{pmatrix}$$

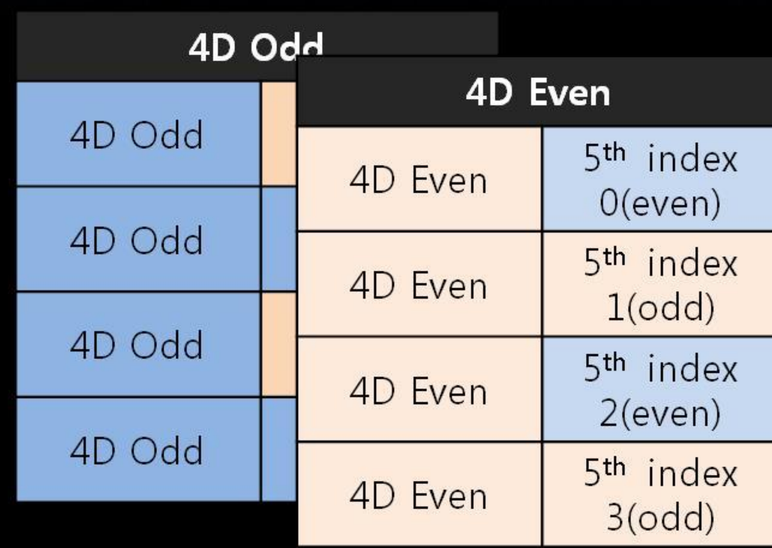
※ After some transformation

⋮

$$\tilde{M}_{4D}^{dwf} = \begin{pmatrix} \delta_{ee} & 0 \\ 0 & M_5 - \kappa_b^2 M_{oe}^{W_4} M_5^{-1} M_{eo}^{W_4} \end{pmatrix}$$

M_5^{-1} → ≡ **Dslash5inv**

4D E-O PC data structure on memory



※ $\kappa_b^{-1} = 2(b_5(4-M)+1)$

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

$P_R = (1+\gamma_5)/2$

$P_L = (1-\gamma_5)/2$

$$D\!\!\!/^{W}_{x,y} = \sum_\mu [(1+\gamma_\mu)U^\dagger_{x-\mu,\mu}\delta_{x-\mu,y} + (1-\gamma_\mu)U_{x,\mu}\delta_{x+\mu,y}] \equiv \textbf{Dslash4}$$

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

$$M_{eo}^{W_4} = D\!\!\!/^{W}_{x,y}(b_5\delta_{s,t} + c_5 D\!\!\!/^{5}) \longrightarrow \equiv \textbf{Dslash4 * Dslash4pre}$$

$$M_5 = 1 + \frac{\kappa_b}{\kappa_c} D\!\!\!/^{5} \longrightarrow \equiv \textbf{Dslash5}$$

Mobius Domain Wall Fermion

- M_5^{-1} Operation $(= M_{5,R}^{-1} P_R + M_{5,L}^{-1} P_L)$

※ (Ls = 4 case)

$$M_{5,R}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -\kappa m_f \\ \kappa & 1 & 0 & 0 \\ 0 & \kappa & 1 & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ \kappa & 1 & 0 & 0 \\ 0 & \kappa & 1 & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1}}_{\equiv A^{-1}} \underbrace{\begin{pmatrix} 1+(-\kappa)^4 & (-\kappa)^3 & (-\kappa)^2 & -\kappa m_f \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}}_{\equiv B^{-1}}$$

M_5^{-1} can be explicitly solved by using LU decomposition in serial or parallel way for the elements.

In CPS, we solve this inversion of matrix by sequential processing

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix} = M_{5,R}^{-1} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} \frac{v_0 - 2\kappa m_f v_3 - (2\kappa)^2 m_f v_2 - (2\kappa)^3 m_f v_1}{1+(2\kappa)^4 m_f} \\ 2\kappa w_0 + v_1 \\ 2\kappa w_1 + v_2 \\ 2\kappa w_2 + v_3 \end{pmatrix}$$

Mobius Domain Wall Fermion

- M_5^{-1} Operation $(= M_{5,R}^{-1} P_R + M_{5,L}^{-1} P_L)$

※ (Ls = 4 case)

$$M_{5,R}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -\kappa m_f \\ \kappa & 1 & 0 & 0 \\ 0 & \kappa & 1 & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ \kappa & 1 & 0 & 0 \\ 0 & \kappa & 1 & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1}}_{\equiv A^{-1}} \underbrace{\begin{pmatrix} 1+(-\kappa)^4 & (-\kappa)^3 & (-\kappa)^2 & -\kappa m_f \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}}_{\equiv B^{-1}}$$

M_5^{-1} can be explicitly solved by using LU decomposition in serial or parallel way for the elements.

In CPS, we solve this inversion of matrix by sequential processing

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix} = M_{5,R}^{-1} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} \frac{v_0 - 2\kappa m_f v_3 - (2\kappa)^2 m_f v_2 - (2\kappa)^3 m_f v_1}{1+(2\kappa)^4 m_f} \\ 2\kappa w_0 + v_1 \\ 2\kappa w_1 + v_2 \\ 2\kappa w_2 + v_3 \end{pmatrix}$$

Mobius Domain Wall Fermion

- M_5^{-1} Operation $(= M_{5,R}^{-1} P_R + M_{5,L}^{-1} P_L)$

※ (Ls = 4 case)

$$M_{5,R}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -\kappa m_f \\ \kappa & 1 & 0 & 0 \\ 0 & \kappa & 1 & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ \kappa & 1 & 0 & 0 \\ 0 & \kappa & 1 & 0 \\ 0 & 0 & \kappa & 1 \end{pmatrix}^{-1}}_{\equiv A^{-1}} \underbrace{\begin{pmatrix} 1+(-\kappa)^4 & (-\kappa)^3 & (-\kappa)^2 & -\kappa m_f \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}}_{\equiv B^{-1}}$$

M_5^{-1} can be explicitly solved by using LU decomposition in serial or parallel way for the elements.

In QUDA, we uses explicit matrix inversion for parallel processing

For general size of Ls,

$$M_{5,R}^{-1} = \frac{1}{1+(-\kappa m_f)^{Ls}} \begin{pmatrix} 1 & -(-\kappa)^{Ls-1} m_f & -(-\kappa)^{Ls-2} m_f & -(-\kappa)^{Ls-3} m_f & \cdots \\ -\kappa & 1 & -(-\kappa)^{Ls-1} m_f & -(-\kappa)^{Ls-2} m_f & \cdots \\ (-\kappa)^2 & -\kappa & 1 & -(-\kappa)^{Ls-1} m_f & \cdots \\ (-\kappa)^3 & (-\kappa)^2 & -\kappa & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

FLOPS vs Bandwidth

- **Floating point operations and Data access**

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

4 x 6 $[\chi(x)]$ + 8 x 4 x 6 $[\chi(x\pm\mu)]$ + 8 x 18 $[U_\mu(x)]$ = 360 : 1440 bytes(32bit)

Arithmetic intensity : 1320 floating point calculations per site

Wilson dirac operation - FLOPS/Bandwidth = 0.92

FLOPS/Bandwidth(@C2050, SP) = 8.7

→ Highly bounded by memory accessing speed~!

※ In other type of fermions(Staggered, wilson, TM,…), Dirac operation is still severely bounded in data accessing, not in the arithmetic operation

	FLOPS/site	Bytes/site
$\not{D}_{4pre} = b_5\delta_{s,t} + c_5\not{D}^5$	168 + 48/Ls	384 (3 Read, 1 Write)
$M_{5,XPAY} = M_5 - \kappa^2(\text{Temp vector})$	192 - 144/Ls	480 (4 Read, 1 Write)
$M_{5,Inv}$	3Ls + 141	192 (1 Read, 1 Write)

CPS Library

Official Web page
http://qcdoc.phys.columbia.edu/cps.html

- CPS : Columbia Physics System
- Mainly developed by CU, BNL, UK QCD group
- Package for lattice QCD application
- Support various lattice actions
  - Domain-wall, Staggered, Wilson, Twisted mass,...

Collaborated by
BROOKHAVEN NATIONAL LABORATORY
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK
UKQCD collaboration

QUDA

Official Web page
http://lattice.github.com/quda/

- Library for lattice QCD based on CUDA environment
- Provides highly optimized CG and BiCGstab inverters on Nvidia GPUs
- Optimized solvers for following fermion actions
  - Wilson, Wilson-clover, Twisted mass, Staggered, DWF (new! mobius DWF),...

Supported by SciDAC Scientific Discovery through Advanced Computing, NVIDIA, NSF, PRACE

Implementation on QUDA

- Mat operation in DWF

$$\tilde{M}^{dwf}_{5D} = \begin{pmatrix} \delta_{ee} & 0 \\ 0 & \delta_{oo} - \kappa^2 \not{D}^{dwf}_{oe} \not{D}^{dwf}_{eo} \end{pmatrix}$$

⇨

```
void DiracDomainWallPC::M(&out, &in)
{
   …
   DslashX(*tmp1, in, ODD_PARITY,…); //10 read, 1 write
   DslashXpay(out, *tmp1, EVEN_PARITY, in,…);//11R,1W
} // 23 vector + 4 gauge data accessing I/O
  // 2880 + 96/Ls (FLOPS/site)
```

- Mat operation in Mobius DWF

$$\tilde{M}^{dwf}_{4D} = \begin{pmatrix} \delta_{ee} & 0 \\ 0 & M_5 - \kappa_b^2 M^{W_4}_{oe} M_5^{-1} M^{W_4}_{eo} \end{pmatrix}$$

⇨

```
void DiracMobiusDomainWallPC::M(&out, &in)
{  …
   Dslash4pre(*tmp1, in, ODD_PARITY);//3R, 1W
   Dslash4(out, *tmp1, EVEN_PARITY); //8R, 1W
   Dslash5inv(*tmp1, out, ODD_PARITY);//1R, 1W
   Dslash4pre(out, *tmp1, EVEN_PARITY);//3R, 1W
   Dslash4(*tmp1, out, ODD_PARITY);//8R, 1W
   Dslash5Xpay(out, in, EVEN_PARITY, *tmp1,…) //4R, 1W
} // 33 vector + 4 gauge data accessing I/O
  // 3192 + 3Ls + 141 + 144/Ls
```

Performance(1)

For 24x24x24x64x8 lattice on 4 C2050 GPUs

1. Dslash4 operation ($\not{D}^W_{x,y} = \sum_\mu [(1+\gamma_\mu)U^\dagger_{x-\mu,\mu}\delta_{x-\mu,y} + (1-\gamma_\mu)U_{x,\mu}\delta_{x+\mu,y}]$)

- Theoretical Peak

  Computation time : 1320 x half Vol / 1TFOPS ≈ 1.2 ms
  Data Access : (9 x 24 + 2 x 8 x 4) x 4 x half Vol/ 115GB/sec ≈ 8.6 ms

- Experimental result

  8.6 ms is needed

2. D4pre operation ($\not{D}^{4pre}_{s,s'} = b_5\delta_{s,t} + c_5(P_R\delta_{s-1,t} + P_L\delta_{s+1,t} - m_f P_R\delta_{s,0}\delta_{t,L_s-1} - m_f P_L\delta_{s,L_s-1}\delta_{t,0})$)

- Theoretical Peak

  Computation time : (171 x half Vol / 1TFOPS ≈ 0.16 ms
  Data Access : (4 x 24) x 4 x half Vol/ 115GB/sec ≈ 3.0 ms

- Experimental result

  3.0 ms is needed

Performance(2)

For 24x24x24x64x8 lattice on 4 C2050 GPUs

3. Dslash5inv operation (${D\!\!\!/}^{5}_{s,s'}{}^{-1} = (P_R\delta_{s-1,t} + P_L\delta_{s+1,t} - m_f P_R \delta_{s,0}\delta_{t,L_s-1} - m_f P_L \delta_{s,L_s-1}\delta_{t,0})^{-1}$)

- Theoretical Peak

  Computation time(Ls = 8) : 165 x half Vol / 1TFOPS ≈ 0.15 ms
  Data Access : 2 x 24 x 4 x half Vol/ 115GB/sec ≈ 1.48 ms
  Maximum time at peak speed : 1.65 ms

- Experimental result

  7.1 ms is needed

- Possible problem : Data broadcasting is not working, “Ls” times of access

$$b_{x,s} = (\; b_{x,0} \quad b_{x,1} \quad b_{x,2} \quad b_{x,3} \quad \dots \;)$$

x x x x

$$M^{-1}_{5,R} = \frac{1}{1+(-\kappa m_f)^{Ls}} \begin{pmatrix} 1 & -(-\kappa)^{Ls-1}m_f & -(-\kappa)^{Ls-2}m_f & -(-\kappa)^{Ls-3}m_f & \dots \\ -\kappa & 1 & -(-\kappa)^{Ls-1}m_f & -(-\kappa)^{Ls-2}m_f & \dots \\ (-\kappa)^2 & -\kappa & 1 & -(-\kappa)^{Ls-1}m_f & \dots \\ (-\kappa)^3 & (-\kappa)^2 & -\kappa & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

Performance(3)

Time Table(MDWF) (Tested on C2050, 4 GPUs. 24x24x24x64x8 lattice, 12 Recon & 8 Recon method used in double and single precision)

```
void DiracMobiusDomainWallPC::M(&out, &in)
{   ...
    Dslash4pre(*tmp1, in, ODD_PARITY);//3R, 1W
    Dslash4(out, *tmp1, EVEN_PARITY);//8R, 1W
    Dslash5inv(*tmp1, out, ODD_PARITY);//1R, 1W
    Dslash4pre(out, *tmp1, EVEN_PARITY);//3R, 1W
    Dslash4(*tmp1, out, ODD_PARITY);//8R, 1W
    Dslash5Xpay(out,in,EVEN_PARITY,*tmp1,...);//4R, 1W
}  // 33 vector + 4 gauge data accessing I/O
   // 3192 + 3Ls + 141 + 144/Ls
```

Dslash type	Single(ms)	Double(ms)
D4pre	3.0	8.5
D4	8.6	22.5
M5inv	7.1	16.3
D4pre	3.0	8.5
D4	8.6	22.5
M5Xpay	4.1	15.7
total	~34	~94

※ 1 vector accessing needs ~0.57ms(SP)

Ideally, M5inv operator can be done within 2ms, it needs to be optimized more.

Performance(4)

CG performance (DWF vs MDWF) (GFLOPS for all GPUs, Single-Double mixed precision)

24x24x24x64 (Fermi C2050)	MDWF	DWF
4 node(s=8)	390	470

MDWF CG performance (Fermi Vs Kepler) (GFLOPS for all GPUs, Single-Double mixed precision)

24x24x24x64	C2050	K20m
4 node	390	840

※ Current version of mobius CG invertor is slower then normal DWF CG inverter about 17%.

Optimization is still in progress

Future Work

- **Lanczos Algorithm on QUDA**
  - Numerical algorithm for finding an eigenvector set
  - Needed for accelerating the EigCG algorithm
  - Highly dominated by memory IO
  - GPU has an advantage in memory bandwidth
  - Communications through PCIE bus should be optimized(Key point)
  - Not started yet…

Summary

- Mobius DWF operator is newly introduced on QUDA
  - Current performance is slightly slower than original DWF
  - Optimization is still in progress
- Mobius inverter is not in the"Master branch of QUDA" but in the "Mobius_DWF branch of QUDA"
- Mobius DWF on QUDA will be very helpful for reducing the chiral error in DWF method