Mobius domain wall fermion method on QUDA

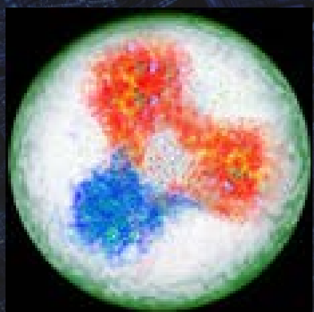
BROOKHAVEN NATIONAL LABORATORY

Hyung-Jin Kim

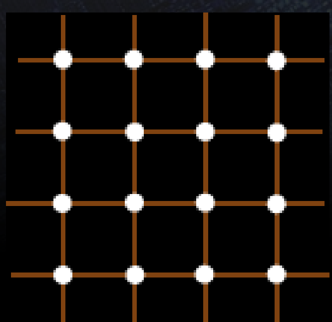
Contents

- Lattice QCD
- Mobius Domain Wall Fermion
- QUDA
- Mobius operator implementation in QUDA
- Performance

Lattice QCD



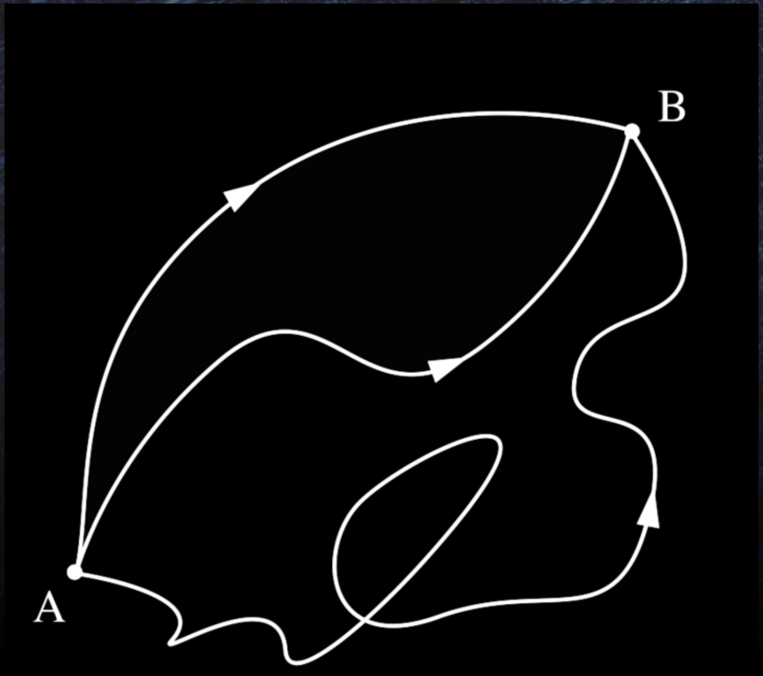
- **QCD ?**
  - Quantum field theory of the strong interaction.
  - Non perturbative at low energy region.



- **Lattice QCD**
  - Non perturbative method to solve the QCD on a discretized space-time.
  - The finite-dimensional path integral, evaluated by stochastic simulation.

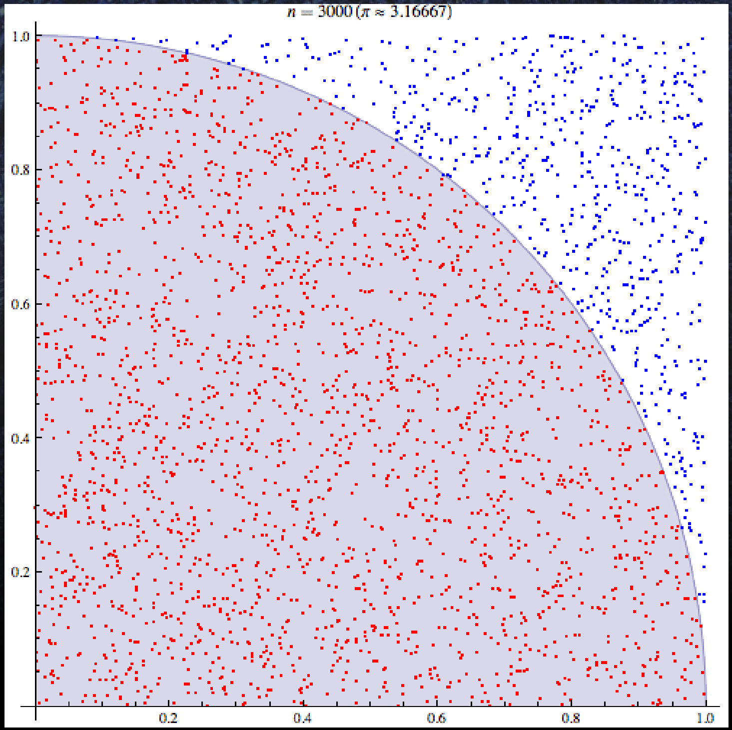
Path Integral & Monte Carlo Method

- Feynman`s Path Integral



Consider every path from A to B!

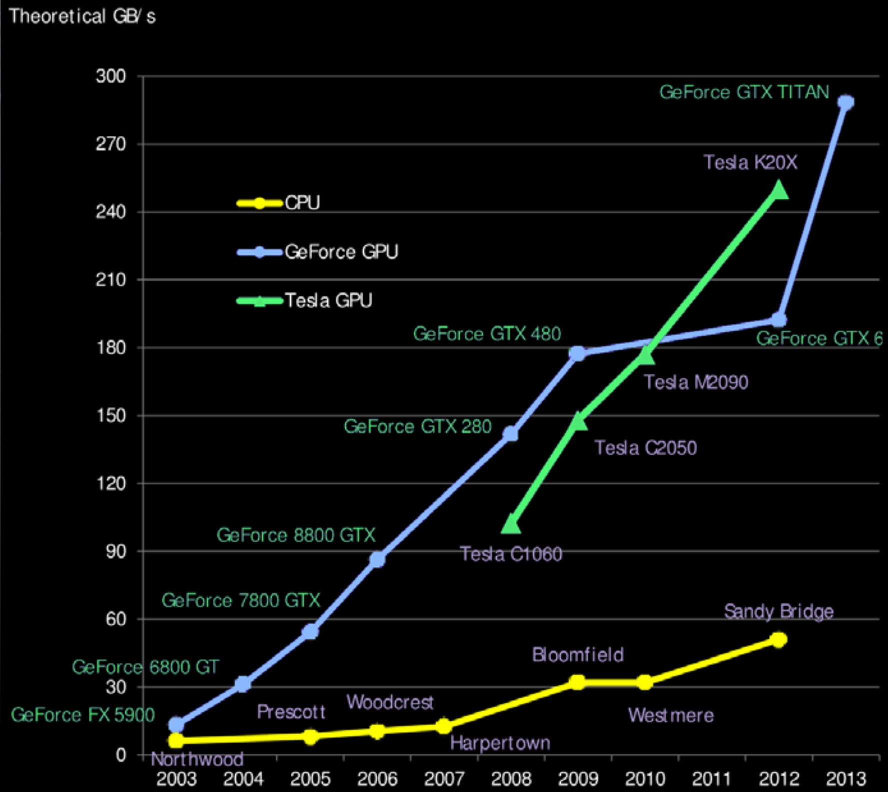
- Monte Carlo Method



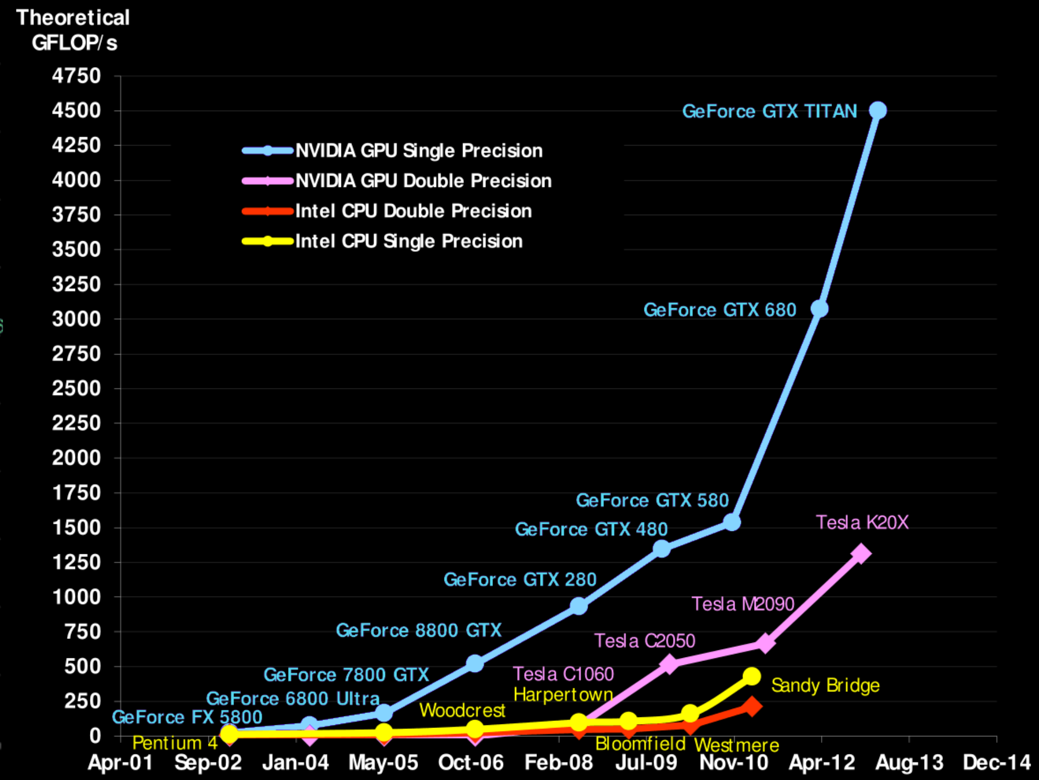
Consider every point in 2D space!

Why GPU?

Evolution of GPU performance



memory bandwidth, GPU vs CPU



FLOPS, GPU vs CPU

GPU is O(10) times faster than CPU in FLOPS,

In memory bandwidth, GPU is still faster at least 5 times than CPU

Physical Observables

- **Quantum mechanical observables**

$$< \mathcal{O} > = \int \prod_{x,\mu} dU_\mu(x) < \mathcal{O} >_U \exp(-S_g(U)) \mathrm{Det}[D(U)]$$

$$= \int \prod dU \prod d\phi \exp(-S_g(U) - \phi^* D(U)^{-1} \phi)$$

$O(10^6)$

$$D(U)^{-1}\phi = \begin{pmatrix} a00 & a01 & a02 & \dots \\ a10 & a11 & a12 & \dots \\ a20 & a21 & a22 & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}^{-1} \begin{pmatrix} b0 \\ b1 \\ b2 \\ \dots \end{pmatrix}$$

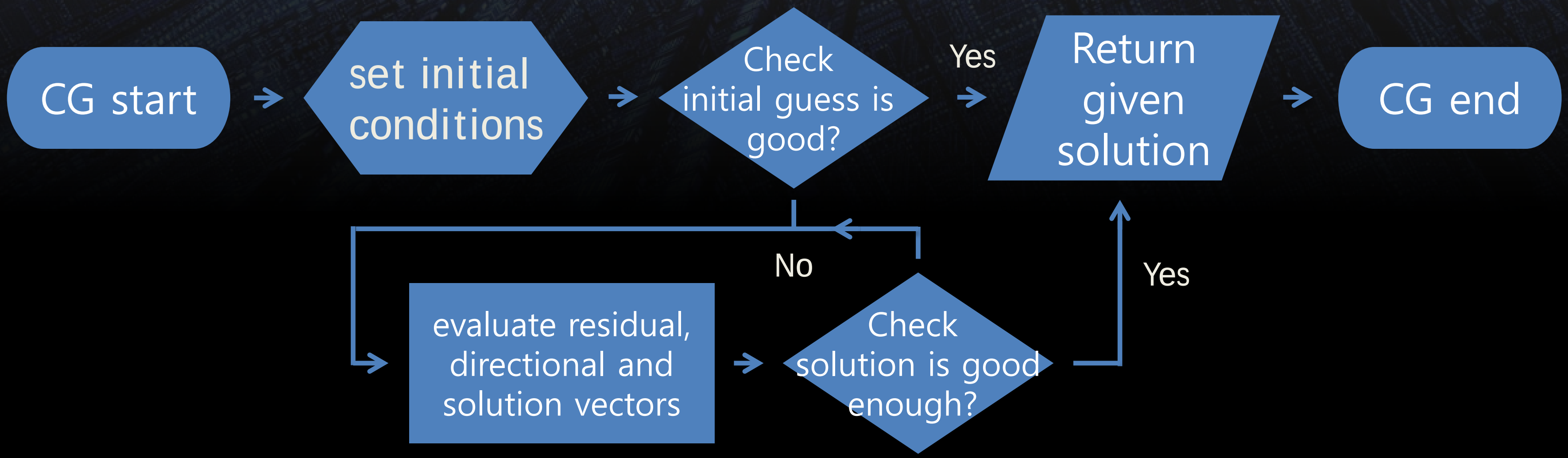
$O(10^6)$

Simple, but huge matrix inversion !

Matrix Inversion Algorithm

- **Conjugate Gradient(CG) method**
  - Iterative method for solving linear algebraic equations : $b = Ax$
  - A : n x n positive definite Hermitian matrix

    x, b : n dimensional complex vectors
  - CG iteration flow



CG iteration

- Conjugate gradient operation

$r = b - Ax$

$d = r$

$\delta_{new} = r^\dagger r$

$\delta_0 = \delta_{new}$

Initial Condition

for $(i = 0;\ i < N_{dim}$ and $\delta_{new} > \varepsilon^2\delta_0;\ i{+}{+})\{$

$Tmp = Ad$

$\alpha = \delta_{new} / d^\dagger Tmp$

$r = r - \alpha Tmp$

$\delta_{old} = \delta_{new}$, $\delta_{new} = r^\dagger r$

$x = x + \alpha d$

$\beta = \delta_{new} / \delta_{old}$

$d = r + \beta d$

$\}$

Update process

r : residual vector
d : directional vector
ε : tolerance
Ax(or Ad): Dirac operation

※ Matrix inversion can be replaced with several linear algebra operations!

Dirac Operator($D(U)$)

- **ex) Wilson Dirac operator**

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

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

Arithmatic intensity : 1320 floating point calculations per site

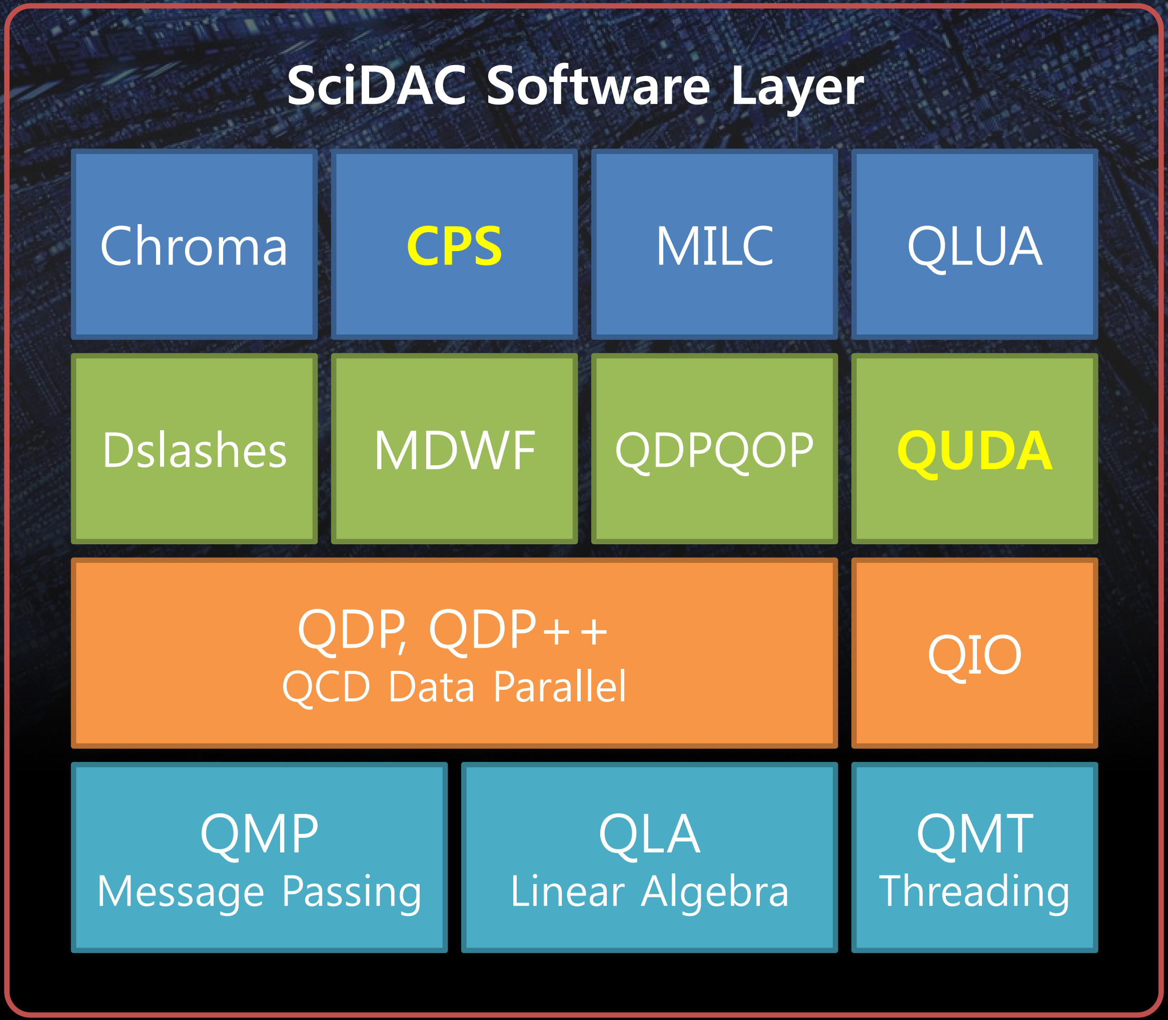
Wilson dirac operation - FLOPS/Bandwidth = 0.92

FLOPS/Bandwidth(@K20x, DP) = 5.24

→ Highly bounded by memory accessing speed~!

※ In other type of fermions, Dirac operation is still severely bounded in data accesssing, not in the arithmetic operation

QCD software



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
- Object oriented high level codes,

  Easy to develop for various lattice action
  - Domain-wall
  - Staggered
  - Wilson(or Clover improved)
  - Twisted mass

Collaborated by

BROOKHAVEN NATIONAL LABORATORY

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

UKQCD collaboration

QUDA

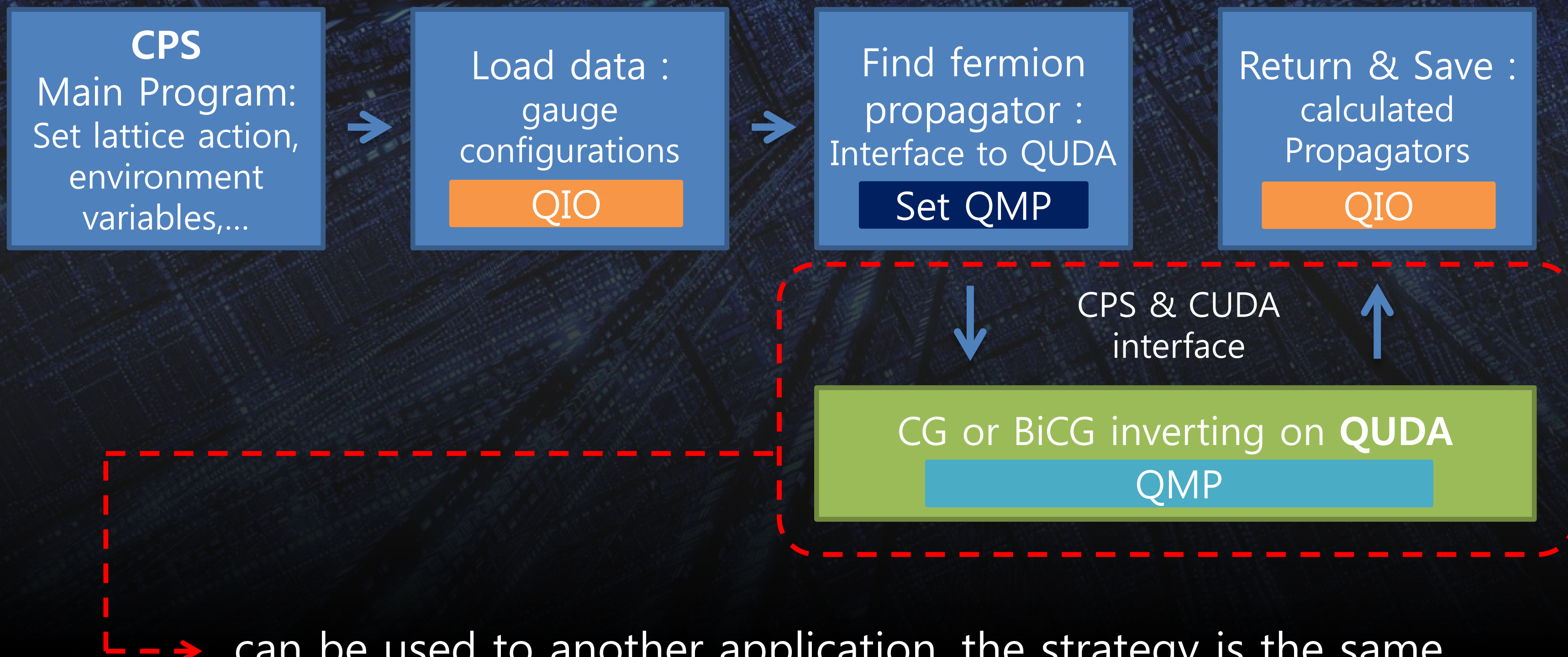
Official Web page
http://lattice.github.com/quda/

- Library for lattice QCD based on CUDA environment
- Provides high performance CG and BiCG invertors
- Optimized solvers for following fermion actions
  - Wilson(or Clover-improved)
  - Twisted mass
  - Improved staggered (asqtad or HISQ)
  - Domain wall(new! : mobius DWF)
- Mixed precision(double, single, half) solver is available

Supported by SciDAC Scientific Discovery through Advanced Computing, NVIDIA

Work Flow

CPS with QUDA work Flow



can be used to another application, the strategy is the same.

Ex) Chroma, MILC and other QCD code

▪ Processor Hour

Ex) for connected diagram

m_s = 0.04 & 0.03 CG hours for HVP measurement with DWF

Lattice size	1 iteration(sec)	CG number	# of Src	# of data	# of GPU nodes	Total Hours
24x24x24x64	0.047	500	12	600	8	376
32x32x32x64	0.114	600	12	600	8	1095

To calculate disconnected diagrams, more computational power is needed!

"Dilution method" will be used for disconnected diagram

O(10~20) of source points are needed

Mobius Domain Wall Fermion

- Mobius DWF
  - Extended version of domain wall fermion
  - Reduced residual mass even in the smaller size of 5th dimension
  - Under the optimized values of $b_5\,, c_5$ coefficients,

$$m_{res} : O(1/L_s) \rightarrow O(1/L_s^2)$$

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

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

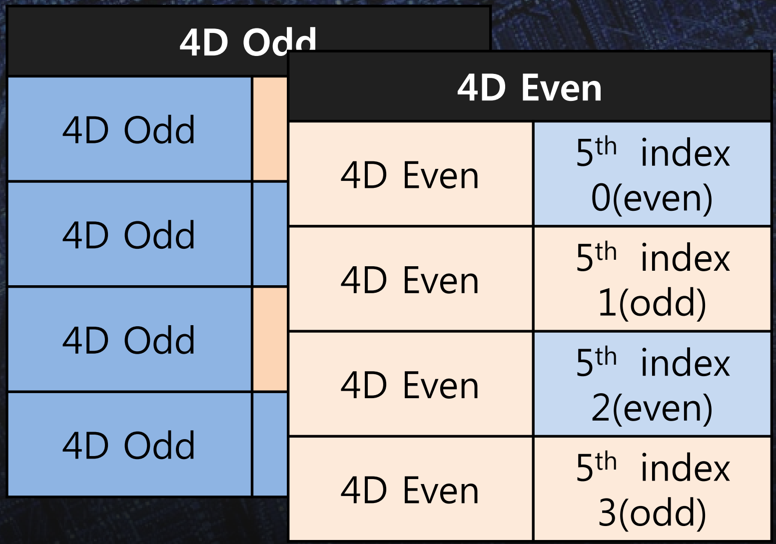
Mobius DWF dirac equation

  - QUDA implementation is in progress

Implementation on QUDA(1)

- Preconditioning Method 1

4D E-O PC data structure on memory



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

⋮

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

※ $M_5 = 1 + \frac{\kappa_b}{\kappa_c} \not{D}^5$

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

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

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

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

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

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

$$M_{eo}^{W_4} = \not{D}^W_{x,y}(b_5\delta_{s,t} + c_5\not{D}^5)$$

Implementation on QUDA(2)

- Preconditioning Method 2

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

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

(Ls = 4 case)

M_5^{-1} can be explicitly solved in serial or simultaneous way for elements.

In QUDA, we uses explicit matrix inversion for parallel processing

For general size of Ls,

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

Implementation on QUDA(3)

- How to use MDWF (Tested on CPS program)

Set QUDA interface variables as …

```
gauge_param.gauge_order = QUDA_CPS_WILSON_GAUGE_ORDER;
gauge_param.type = QUDA_WILSON_LINKS;
…
inv_param.dslash_type = QUDA_DOMAIN_WALL_DSLASH;
inv_param.b_5 = CPS_b5;
inv_param.c_5 = CPS_c5;
inv_param.dirac_order = QUDA_CPS_WILSON_DIRAC_ORDER;
inv_param.solve_type = QUDA_MDWF_EQ_PC_SOLVE;
…
```

Performance(DWF)

CG performance on QUDA
GFLOPS for all GPUs, Single-Double mixed precision

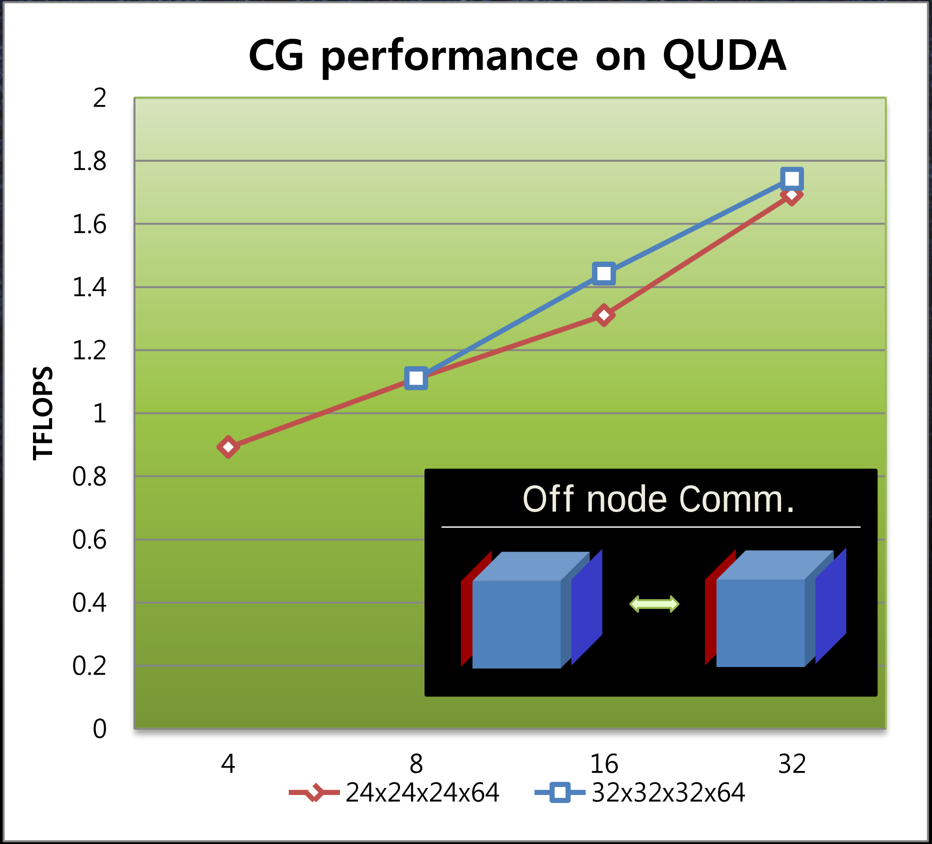
	4	8	16	32
24^3x64x16	892 (1,1,1,4)	1110 (1,1,1,8)	1310 (1,2,2,4)	1692 (1,1,4,8)
32^3x64x16	N/A	1110 (1,1,2,4)	1442 (1,2,2,4)	1743 (2,2,2,4)

Scalability on virtual size of lattices (= # node/1 node)

1 (24^3x16x16)	4 (24^3x64x16)	8 (24^3x128x16)	16 (24^3x256x16)
248.0(1.00)	890(3.59)	1691(6.82)	3367(13.6)

Surface / Volume ratio

Nodes→ ↓Lattice	4 (1,1,1,4)	8 (1,1,1,8)	16 (1,1,2,8)	32 (1,1,4,8)
24^3x64x16	0.067	0.143	0.263	0.412
32^3x64x16	N/A	0.143	0.231	0.333



Performance

CG performance on various platforms (GFLOPS for all GPUs, Single-Double mixed precision)

24x24x24x64	RICC (Fermi C2075)	Jlab (Kepler K20m)	Geforce Titan (Kepler GK110)
2 node(s=8)	232	502.5	793
4 node(s=16)	399	892	No data

CG performance in 48x48x48x96 lattice (GFLOPS for all GPUs, Single-Double mixed precision)

Node geometry	72(2,3,3,4)	72(1,3,3,8)
48x48x48x96	3547	3617

Future Work

- **Lanczos Algorithm**
  - Numerical algorithm for finding an eigenvector set
  - 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

- Cutting edge GPUs are very powerful even for the HPC
- QUDA is the best solution for the lattice QCD computations on GPU.
- Mobius DWF operator is newly introduced in QUDA
  - Optimization is still in progress
- With QUDA CG inverter, TFLOPS scale of computational performance can be easily achieved.