Accelerating lattice QCD simulations using multiple GPUs

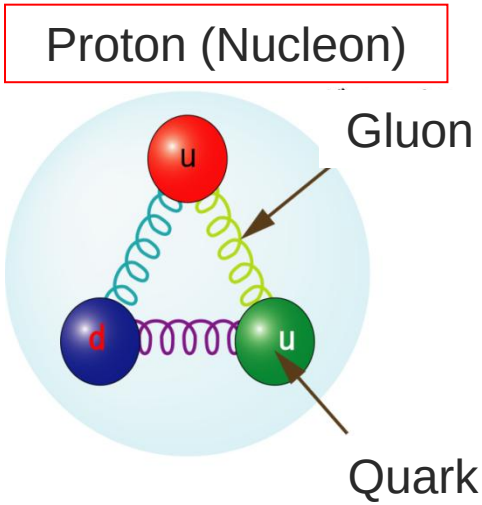
Ken-Ichi Ishikawa
(Dept. of Physical Science,
Hiroshima Univ.)

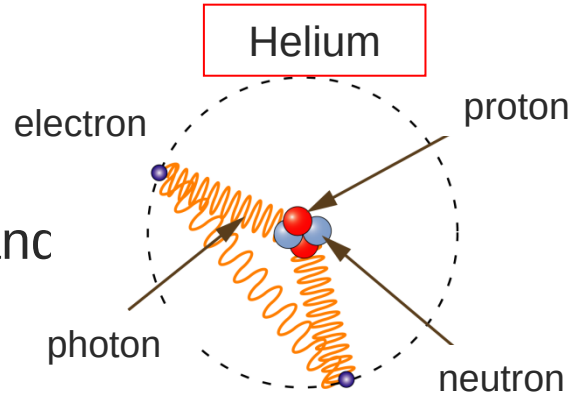
Contents

- 1. Introduction to QCD
- 2. Lattice QCD
- 3. LQCD with GPU/Accelerator
- 4. Accelerating *D[U]x=b* solver using single GPU
- 5. Many flavor lattice QCD with multiple GPUs on single node
- 6. Towards parallel GPU computation
- 7. Summary

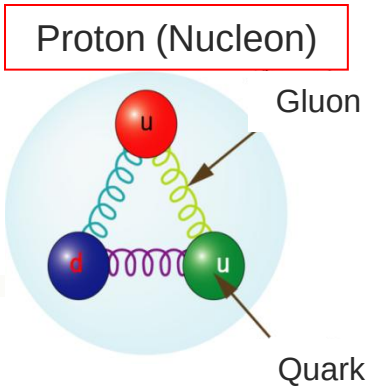
1. Introduction to QCD

- QCD (Quantum Chromodynamics)
  - Describes properties of
    - Nucleons (Proton, Neutoron)
    - (Mesons, Baryons =Hadrons)
    - Strong interaction
  - from more fundamental particles Quarks and Gluons
    - Dynamics by exchanging gluons.
  - e.g. QED (Quantum Electrodynamics)
    - Molecules/Atoms from nucleons/electrons anc photons





1. Introduction to QCD (contd')



- QCD (Quantum Chromodynamics)
  - Quantum Field theory (QFT)
    - Quark Field and Gluon Field with space-time index. They have color charge (based on SU(3)).

      (Electric charge based on U(1))
      - Gluon action : similar to Maxwell's equation (photon action),

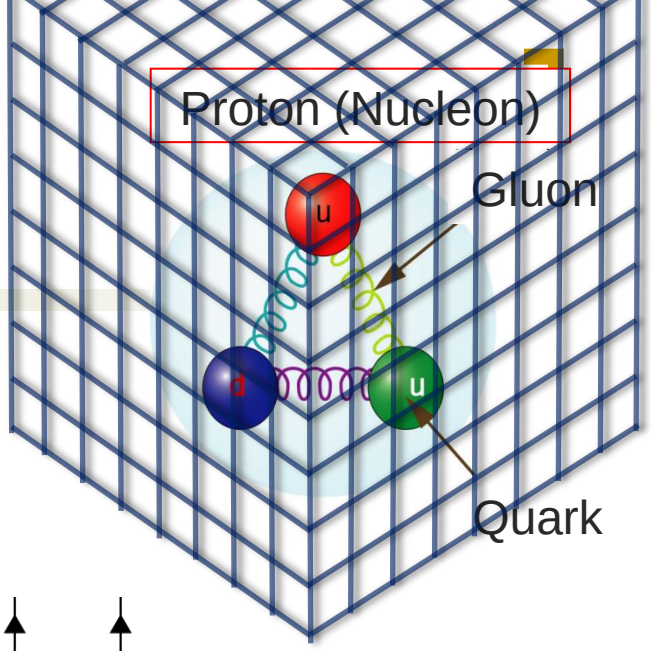
        Gluon carries its color charge.

        (photon does not have electric charge.)
      - Quark action: similar to electron's equation (Dirac's equation/action)

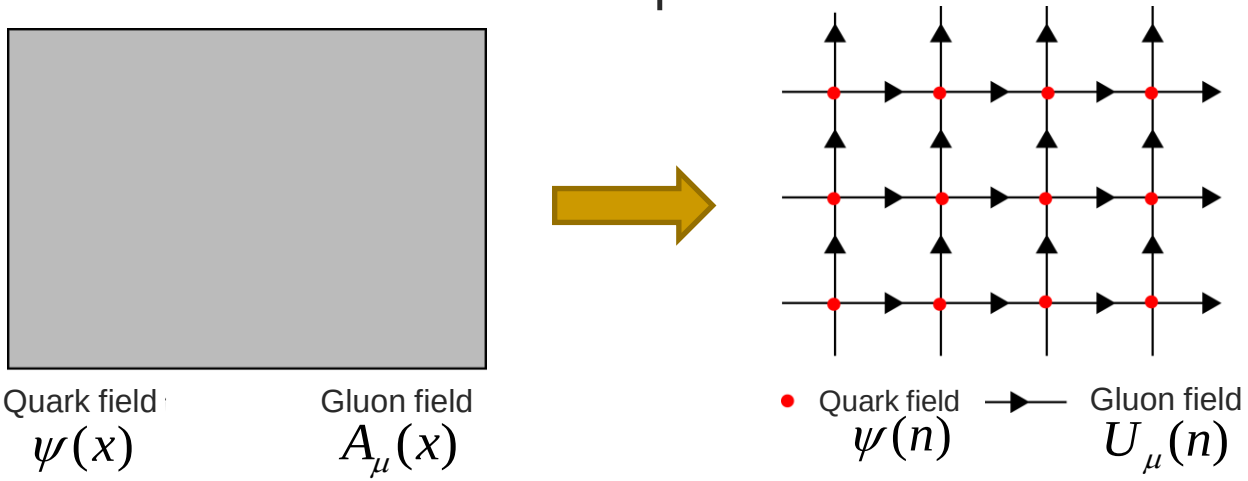
        Quark carries color charge. (electron has electric charge)
    - Quatntized by Feynman's path-integral. Partition function with action (similar to statistical physics).

$$Z = \int DAD\overline{\Psi}D\Psi \exp\left[\int d^4x\left(-S_{gluon}(A) - S_{quark}(A,\overline{\Psi},\Psi)\right)\right]$$

2. Lattice QCD

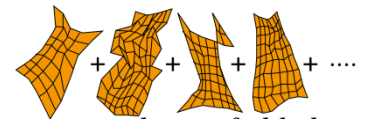


- Lattice QCD [K.G.Wilson (1974)]
  - 4D Space Time => 4D Lattice Box
  - Fields on Discretized Space-Time

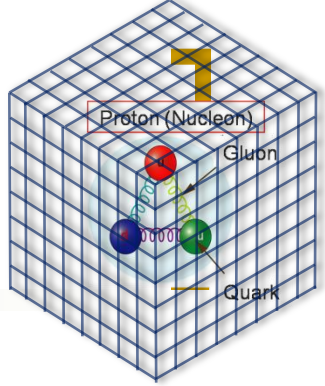


  - Integration on Field Shape => Integration on many var's.

$$\int DAD\overline{\Psi}D\Psi \quad \Rightarrow \quad \int \prod_{n,\mu} dU_\mu(n) \prod_n d\overline{\Psi}(n) d\Psi(n)$$



Lattice QCD [K.G.Wilson (1974)]



- LQCD Partition function

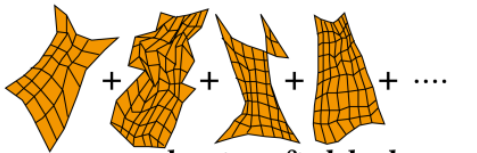
$S(U,\bar{\psi},\psi)$: Discretized version of QCD action

$$Z = \int\prod dU d\bar{\psi} d\psi \exp\left[-S(U,\bar{\psi},\psi)\right]$$

$$= \int\prod dU \exp\left[-S_{eff}(U)\right]$$

analytic integration on Grassmann vars $\bar{\psi},\psi$.

- Integration on many var's.



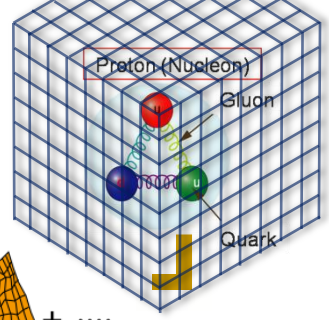
- Observable: O

$$\langle O \rangle = \frac{1}{Z}\int\prod dU O(U) e^{-S_{eff}(U)}$$

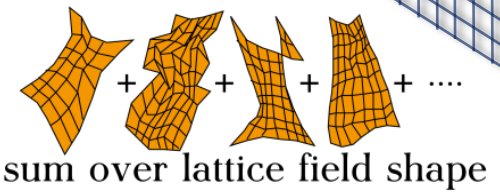
Computation of hadron masses, etc…

- Similar to Statistical Physics

Monte Carlo Integration with Supercomputer!!



Lattice QCD [K.G.Wilson (1974)]



- Monte Carlo Importance Sampling

$$Z = \int \prod dU e^{-S_{eff}(U)} \qquad \langle O \rangle = \frac{1}{Z} \int \prod dU O(U) e^{-S_{eff}(U)}$$

- Markov chain Monte Carlo to generate sequence of *U*

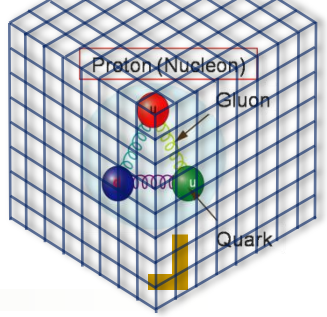
$$U^{(1)} \to U^{(2)} \to U^{(3)} \to \cdots \to U^{(j)} \to \cdots \qquad \mathrm{Prob}[U] \propto e^{-S_{eff}(U)}$$

- Statistical Average

$$\langle O \rangle = \frac{1}{N_{sample}} \sum_{j=1}^{N_{sample}} O(U^{(i)}) \qquad (N_{sample} \to \infty)$$

- To generate the Markov chain of *U* (gluon field)

Hybrid Monte Carlo (HMC) algorithm
[Duane, Kennedy, Pendleton, Roweth(1987)]
is usually employed.



Lattice QCD

HMC algorithm [Duane, Kennedy, Pendleton, Roweth(1987)]

- The most time consuming part of the HMC algorithm is the inversion of quark matrix *D[U].*

$$e^{-S_{eff}(U)} \Rightarrow D\phi^{\dagger} D\phi \exp\left[-S_{gluon}(U) - \left|\left(D[U]\right)^{-1}\phi\right|\right]$$

- Molecular Dynamics (MD) evolution is used to generate a sequence of *U* (Markov chain) in the HMC. This requires huge number of solution of the *linear equations*.

$$D[U]x = y \quad \Rightarrow \quad y = \left(D[U]\right)^{-1} x$$

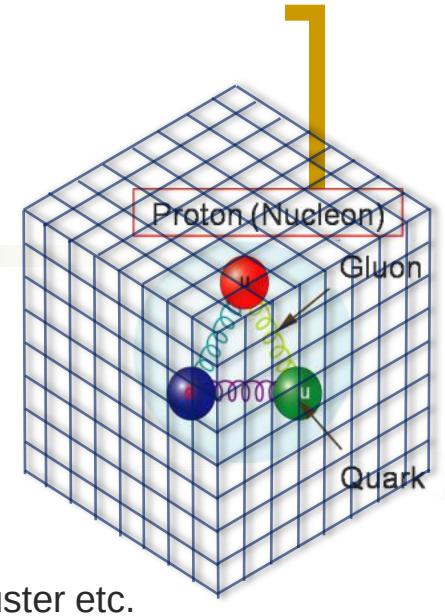
D : size > 10^6 x10^6

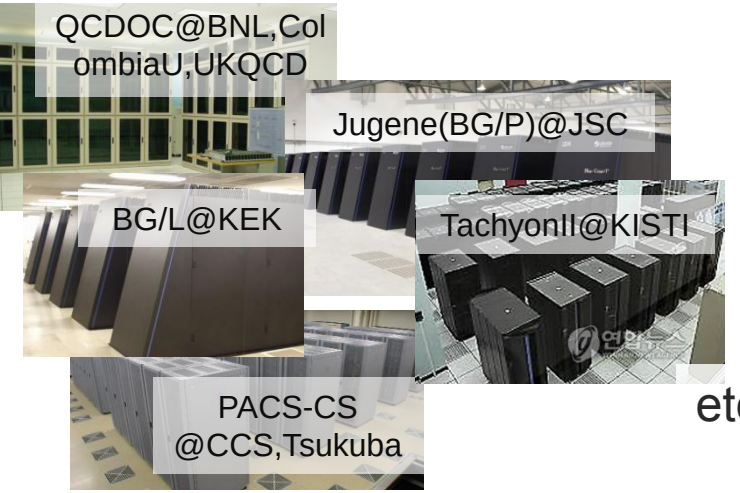
- To get better statistics, we need O(100) configurations of *U.* This needs O(1000) the HMC cycles. In each HMC cycle, MD needs O(100) time steps. At each time step we need two inversion of D[U]. In total we have to invert *D[U]* by 2x100x1000x100 = O(10^7) times!.
- Speeding up the large scale linear equations solver is the key of LQCD simulations.

Lattice QCD

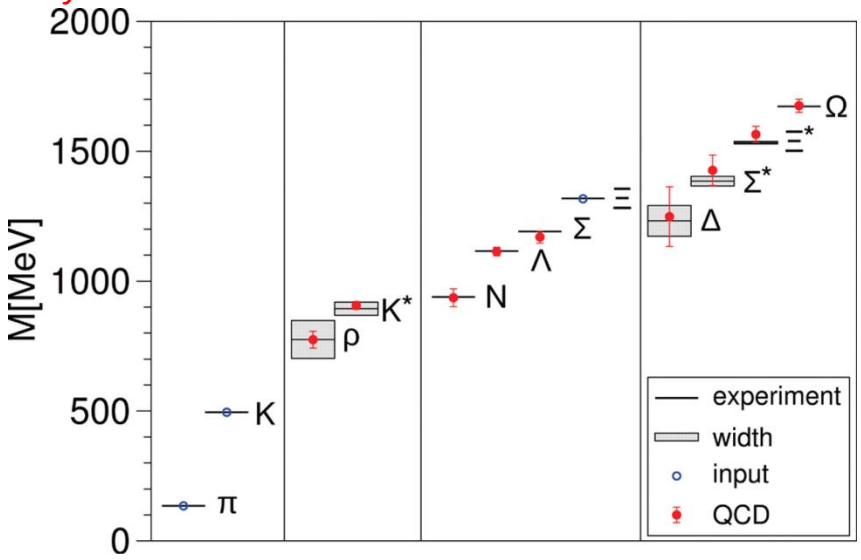
Hadron masses from LQCD (~2010).

- Proton size = 1~2fm (experimental)
- Typical Lattice size L = 16~32
- Typical Lattice spacing a = 0.05~0.1 fm
- Typical Lattice extent La = 2 ~ 5 fm
- Machines:
  - Supercomputers, BG/L/P, SX, SR, VPP.. , Custom made PC cluster etc.
  - Uses O(10) TFlops machines for several years.





etc……



Durr et al. (Budapest-Marseille-Wuppertal collab. ,"Ab Initio Determination of Light Hadron Masses", *Science* 322, 1224(2008).

3. LQCD with GPU/Accelerator

- For all computation in HMC, Parallel Supercomputers are usually employed, where 4D space-time is domain-decomposed and the task is distributed to each node.
- (O(10)Tflops x 2-3 years) machine can now well reproduce single hadron in the computer.
- But such machines are still too expensive… for a novice researcher or to do a trial computer experiment …..
- Improvements on the algorithm and computer arch.s are highly desired.
- GPGPU and Accelerator are the candidate to get more efficient and economical machines for LQCD.
- Speeding up the quark matrix linear solver *D[U]x=b.*

- LQCD experience on GPGPU/Accelerator
  - CELL B.E. (PlayStation3)

Spary,Hill,Trew hep-lat/0804.3654;

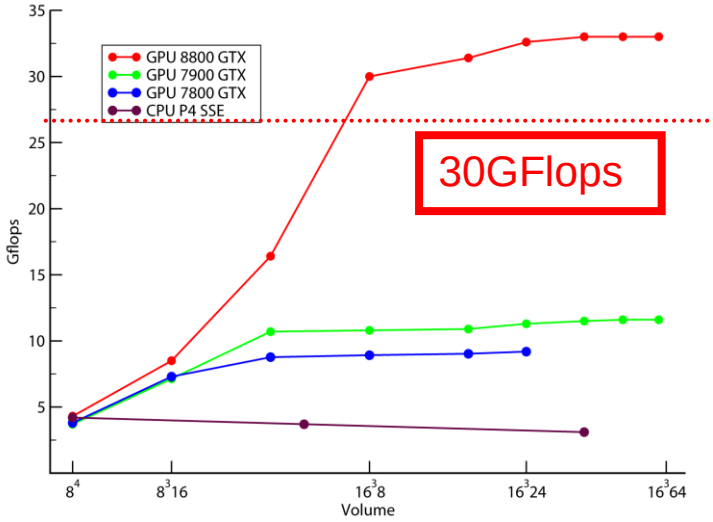
S.Motoki & A. Nakamura Lat2007;

F.Belletti et al. LAT2007

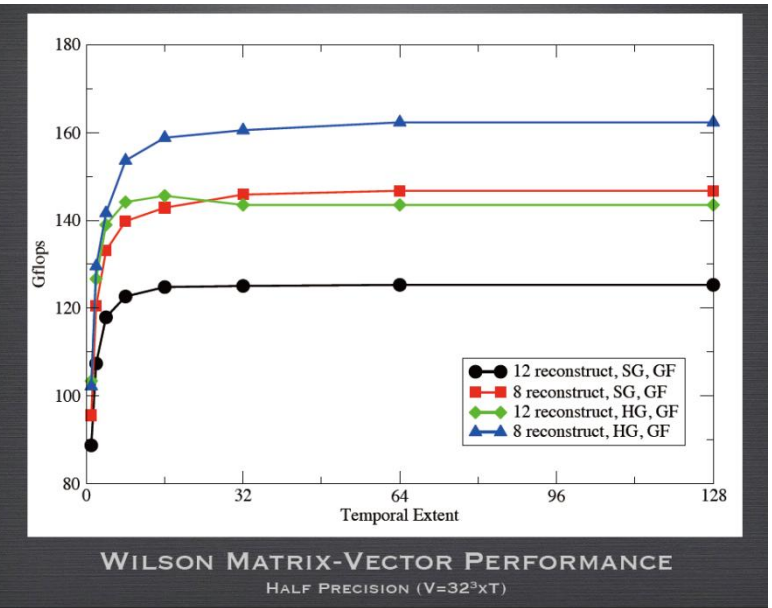
A. Nobile et.al, DESY group, **QPACE** project (**Q**CD **PA**rallel computing on the **CE**ll/B.E.)

For details of Lattice QFT on CELL .B.E. see Next talk by Motoki.

- GPGPU
  - **"Lattice QCD as a video game"**, G.I.Egri, Z.Fodor, S.D.Katz, D.Nogradi, K.K.Szabo, (2006)hep-lat/0611022.
    - NVIDIA G80 arch. > 300 GFlops(SP)
    - Lattice Wilson kernel > 30 GFlops
    - Difficult to program using Graphic API (OpenGL)
  - CUDA version (2008~) with new GPU G200 arch.
    - Now CUDA (a C/C++ simple extension) is avairable.
    - Easy to learn, but requires hardware/memory model knowledge



Quark solver spped (2006)



- C. Rebbi, "Blastigng Through Lattice Calc. using CUDA", Lat08.
- F. Di Renzo, "GPU computing for 2-d spin systems:CUDA vs OpenGL", Lat08
- And many studies….

M. Clark et al. Work shop@CCS, 10-12 March, 2009
They got **140 Gflops** (SP) for Wilson-D[U] computation using Nvidia GTX280.

GPGPU

- AMD GPU, Firestream, OpenCL ….
- applications to LQCD
  - "Pseudo-random number generators for Monte Carlo simulations on Graphics Processing Units", V.Demchik, hep-lat/1003.1898.
  - "Monte Carlo simulations on Graphics Processing Unists", V.Demchik and A.Strelchenko, hep-lat/0903.3053v2.

They employed CAL of ATI Stream.

- For more detailed GPGPU usage in LQCD application, see,
  "QCD on GPUs: cost effective supercomputing" by M.A. Clark , Lattice2009, PoS(LAT2009)003 [hep-lat/0912.2268].

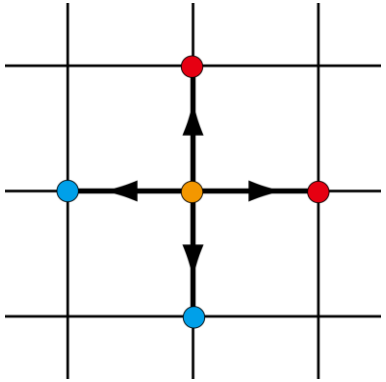
GPU acceleration LQCD Kernel *D[U]*

- There are several discretization scheme for Dirac equation *D*. (*D* : 4D 1st order diff. eq.)
- Wilson-Dirac discretization is the most generic one and used widely. And this has an important kernel called Hopping matrix, which is common to other discretization.
- Improvement on the Hopping kernel is very important.

Hopping kernel (Hopping matrix)

$$M(n,m) = \sum_{\mu=1}^{4} \left[(1-\gamma_\mu) \otimes U_\mu(n) \otimes \delta_{n+\hat{\mu},m} + (1+\gamma_\mu) \otimes U_\mu^\dagger(m) \otimes \delta_{n-\hat{\mu},m} \right]$$

$$D_W[U](n,m) = 1 \otimes 1 \otimes \delta_{n,m} - \kappa \; M(n,m)$$



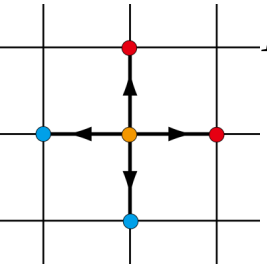
- 4D, first-difference operator.
- sites connected by gluon *U*, by which quark-color are mixed.
- quark-spin are mixed by Dirac's gamma matrix γ_μ
- Memory bandwidth intensive operation.
  - Byte/Flop ~ 3 B (D.P.), ~1.5 B (S.P.)

GPU acceleration LQCD Kernel *D[U]*

- My personal experience on the CUDA programming for the Hopping matrix (single GPU)

Hopping kernel (Hopping matrix)

$$M(n,m)=\sum_{\mu=1}^{4}\left[\left(1-\gamma_\mu\right)\otimes U_\mu(n)\otimes\delta_{n+\hat{\mu},m}+\left(1+\gamma_\mu\right)\otimes U_\mu^{\dagger}(m)\otimes\delta_{n-\hat{\mu},m}\right]$$



$$D_W[U](n,m)=1\otimes 1\otimes\delta_{n,m}-\kappa\ M(n,m)$$

- We want to solve $D_W[U]x=y$ for *x* with given *y* and *U*.
- Lin. eq. with large sparse matrix :=> Iterative solver (CG, BiCGStab etc..)
- CUDA/GPU: Single precision is very fast (Tflops), but we need double precision solution *x*.
- We use the mixed precision iterative solver (generalization of iterative refinment/Richardson iteration). This guarantees double precision accuracy with almost S.P. arithmetics.

GPU acceleration LQCD Kernel *D[U]*

- Mixed precision solver (strategy) [early proposal by Buttari, Dongrra, Langou, Langou, Luszczek, Kurzak (2007)]
  - To solve $Dx = b$.

(0) [given r and x satisfy $r = b - Dx$. (double prec.)]

(1) [Solve $Dv = r$ in single precision]

(2) $q = Dv$ [doble prec.]

(3) $x = x + v$ [doble prec.]

(4) $r = r - q$ [doble prec.]

[new r and x still satisfy $r = b - Dx$.]

(5) [Check $|r|$ and goto (1)]

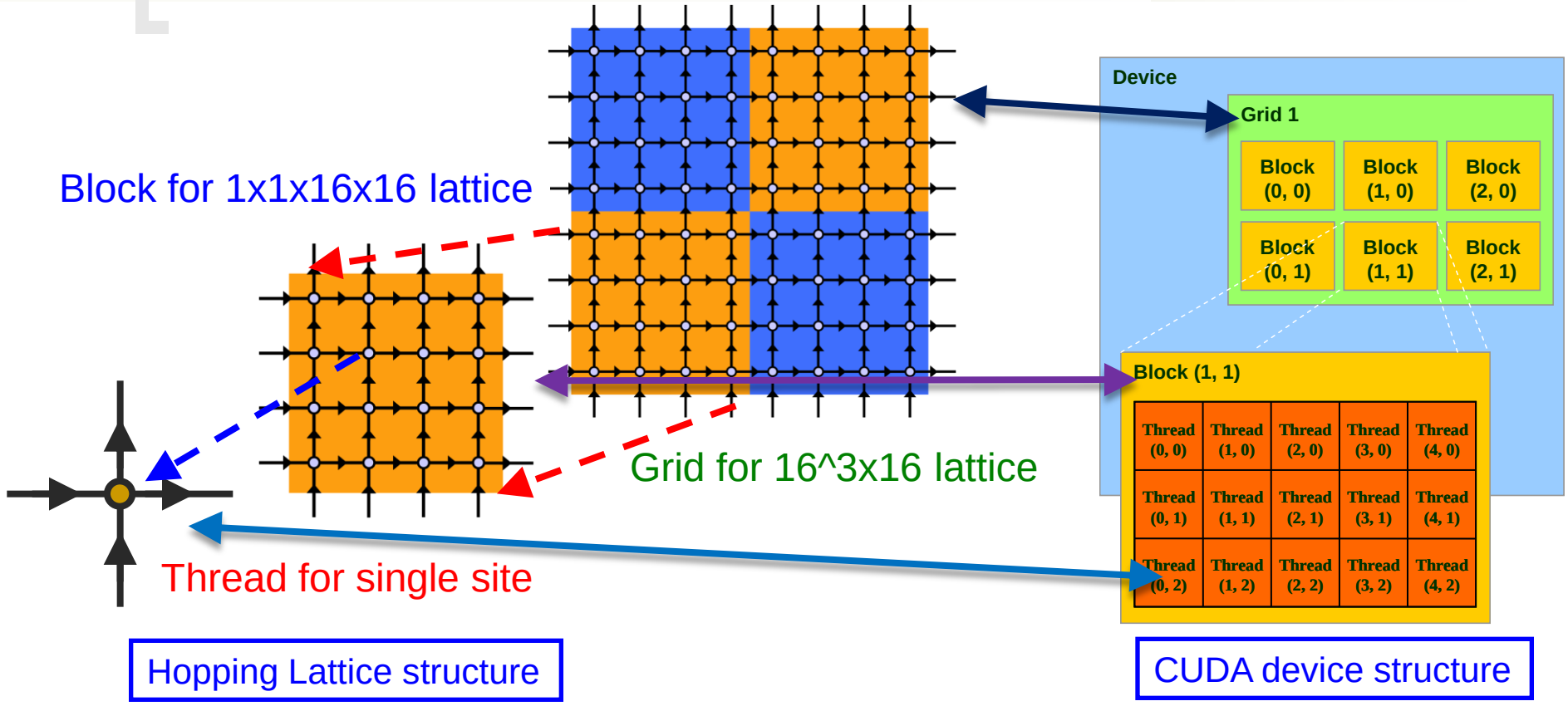
GPU task
CUDA code

CPU task
HOST code

  - The iterative refinement technique with full single precision solver (10^-7) can solve full double precision (10^-14) solution within 3-5 refinement iterations.
  - Most computing time is spent in the S.P. solver.
  - GPU is employed for Full single precision solver.

GPU acceleration LQCD Kernel *D[U]*

- Hopping kernel CUDA code



Mapping between
CUDA structure and Lattice structure

GPU acceleration LQCD Kernel *D[U]*

Various tuning tips using CUDA for LQCD!

Hopping kernel CUDA code

Memory structure (data ordering)

- CUDA requires appropriate data ordering to get efficient memory bandwidth (like vector processor).
- We have to organize quark/gluon field component (spin and color) appropriately to match with CUDA memory alignment.
- Gluon field on single site in a direction, *U*, has 3x3 complex elements. (real*4)x18 = 72bytes. (not match16bytes alignment) We make use of SU(3) property of *U* to reduce data size. 3x2 complex elements are sufficient to reconstruct 3x3 *U.* (12 real*4)=48 bytes matches 16bytes alignment.
- We use y(3,4,T,Z,Y,X) (in Fortran array form) for quark field.
- Complex-Color-Spin (3,4) indexes are encoded to float4 array.
- A part of TZ plane is assigned to CUDA thread parallelization. Remaining TZ,Y,X indexes are assigned to CUDA block parallelization.
- Texture fetching is used to read memory (to cache data y and U)

My understanding on the CUDA programming for LQCD application is:

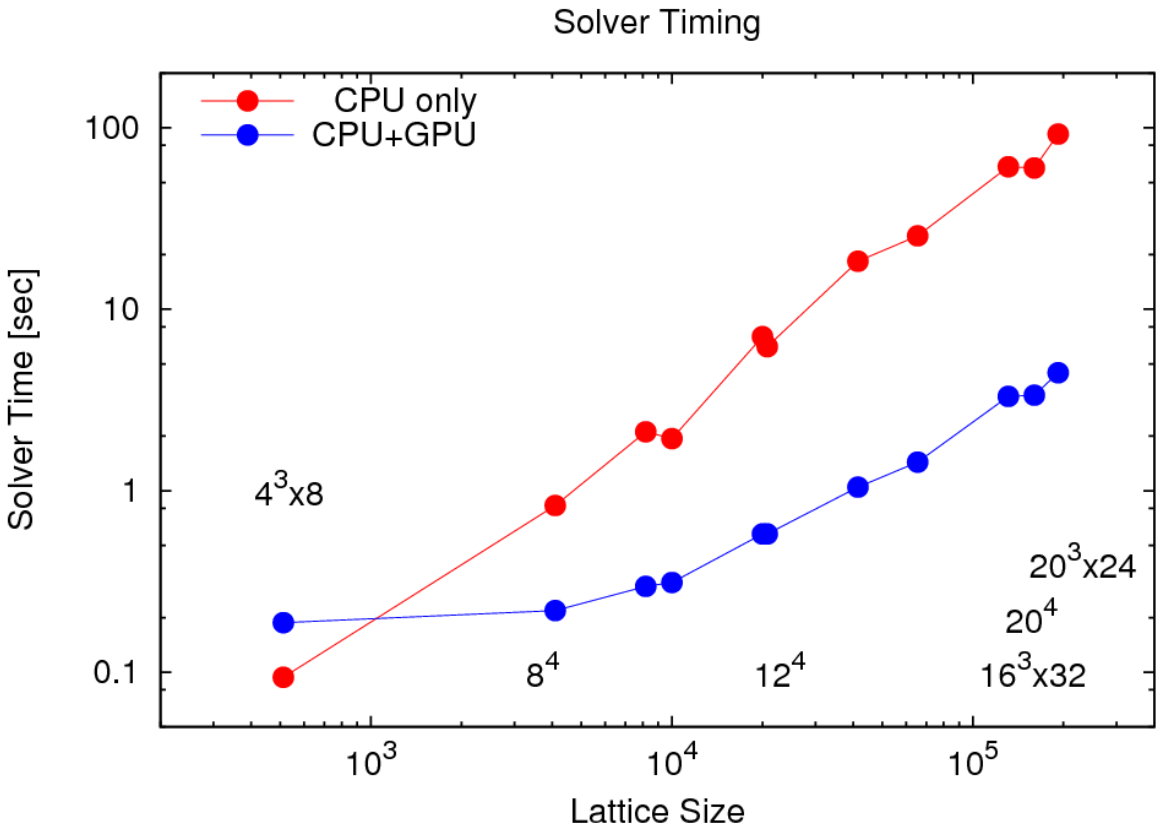
- The CUDA thread parallelization is similar to vetorization on vector machines.
- The block parallelization is similar to usual parallelization as MPI.
- Calling the CUDA kernel is similar to MPI job submitting.
- This strategy is very familiar to lattice guys….

4. Accelerating *D[U]x=b* solver using single GPU.

- Some results [K.-I.I. and Y.Osaki, study in 2008]
  - CPU: Core2Duo@3GHz, GPU: GeForce GTX 280, CentOS.
  - O(a)-improved Wilson-Dirac Ferimion
  - Red/black site prec'd, Nested-BiCGStab (mixed prec. solver)
  - Random gluon field *U.*
  - Programming language
    - HOST: Fortran90, BiCGStab, BiCGStab calls single precision BiCGStab(GPU) as a preconditioner.
    - GPU: CUDA and C/C++. Single precision BiCGStab.
  - Residual history, performance, Lattice Volume (space-time) dependence etc.

Volume dependence of performance

- CPU(DP) only vs CPU(DP)+GPU(SP)

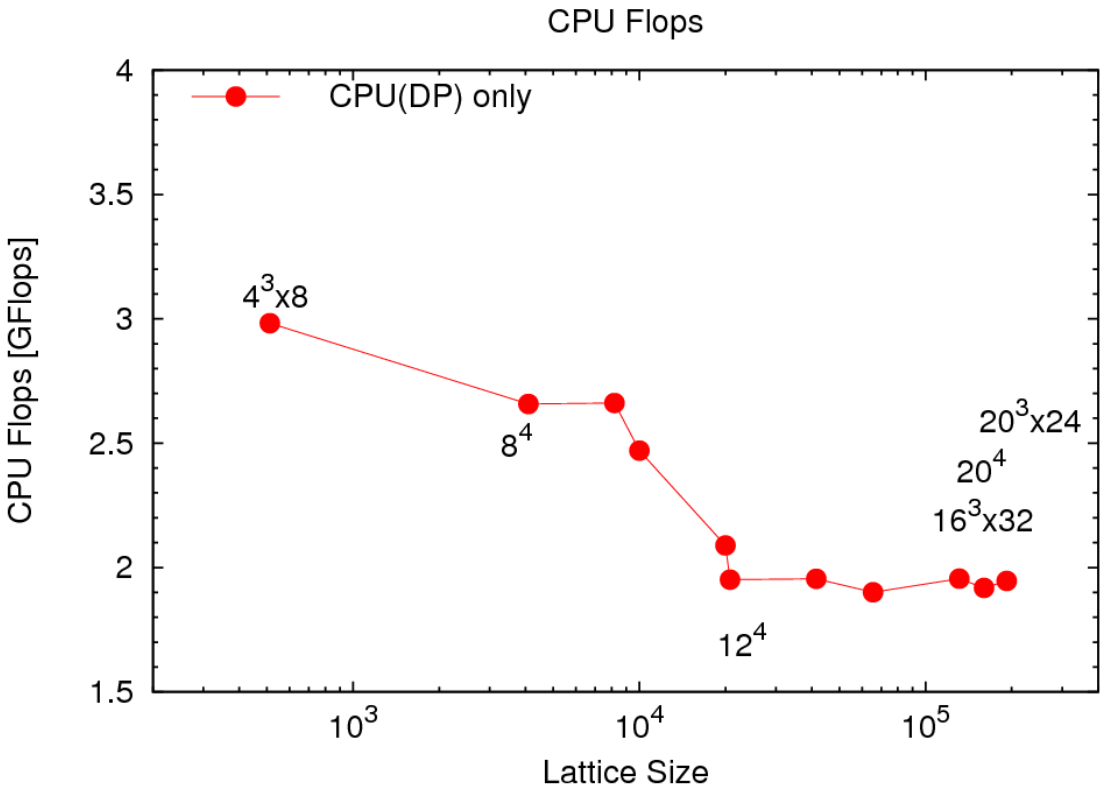


Small volume is not effective for GPU because parallelism is less than num of GPU cores(maximum threads).

For larger lattice we can achieve ~10x~20x speed up.

Volume dependence of performance

- CPU performance (GFLops)



CPU off-cache performance is about 2GFlops (D.P.).

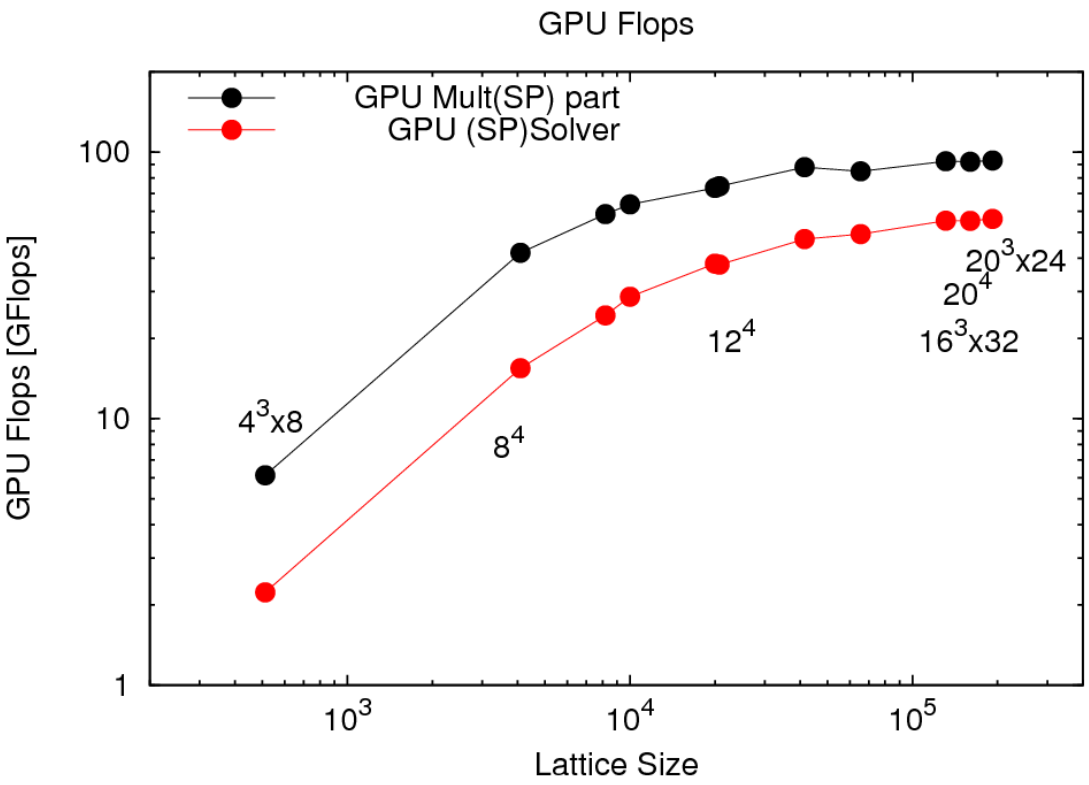
Hopping kernel requires about 3bytes/flop.

Effective memory bandwidth is about 6GByte/s.
[Stream benchmark, Triad = 7.7GB/s (meas'd)]

Lattice QCD is bandwidth intensive application.

Volume dependence of performance

- GPU performance (GFLops(SP))



GPU performance reaches about 100 GFlops (S.P.).

Hopping kernel requires about 1.4bytes/flop.

Effective memory bandwidth is about 140GByte/s!! [Bandwidth test : 115GB/s]

Texture fetching and high memory bandwidth is very important to achieve this performance.

To get good efficiency large volume should be assigned to single GPU.

4. Accelerating *D[U]x=b* solver using single GPU.

- Single GPU performance in LQCD.
  - We found 10x-20x speed up using single GPU.
  - 16^4 lattice is enough for experimental simulations or other lattice simulations.
  - But this is not our final goal because 16^4 is still small volume. (32^4 or larger is wanted).
  - To enlarge lattice size, multiple GPU or parallel GPU usage is required.
- Application class:

(1) Multiple GPUs in a single node.

for other kind of lattice simulations

(2) Multiple GPU, Multiple nodes

for O(10)Tflops machine for 32^4 lattice

5. Many flavor lattice QCD with multiple GPUs on single node

- Schrodinger Functional (SF) simulation with N_f=10 QCD.
  - Class (1) Multiple GPUs in a single node application.
  - Nf=10 QCD as a Technicolor model beyond Standard Model of elementary particles.
  - 16^4 lattice is enough. Parameter searching type simulation.
  - Simulation cost ∝ to number of quarks (10 quarks ⇔ Nf=10) (Proton/neutron : Nf=2 or 3 QCD)
  - The Technicolor model is not ye established experimentally. Numerical simulation helps the validation from theoretical side. But simulation cost is too high!! , and most computer time is given to normal Nf=2 QCD simulations…..
  - GPU acceleration can helps this situation.

- Schrodinger Functional (SF) simulation with N_f=10 QCD.
  - We would like to simulate Nf=10 QCD more economically with GPU but
    - Simulation cost ∝ to number of quarks (10 quarks ⇔ Nf=10) (Proton/neutron : Nf=2 or 3 QCD). 5x heavy for Nf=10 than Nf=2.
    - 10 quarks are independent each others in the HMC algorithm. Solver for each quark can be done independently.
    - Task can be distributed to each GPU (GPU = one quark) in single node.

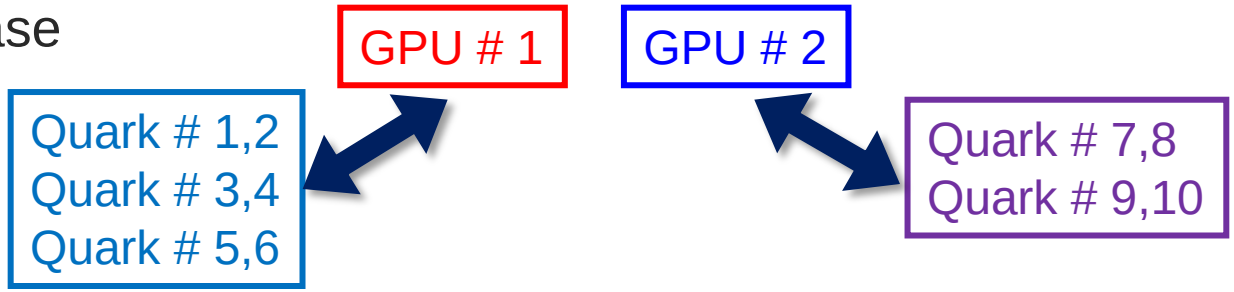
- We tested Nf=10 SF QCD using 2GPUs in a node.

- Schrodinger Functional (SF) simulation with N_f=10 QCD.

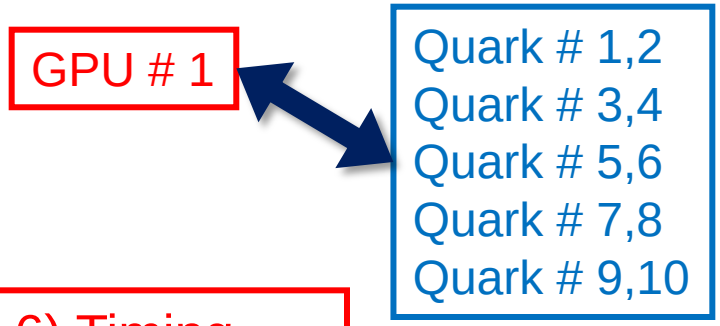
N.Yamada, M.Hayakawa, K.-I.I., Y. Osaki, S.Takeda, S.Uno.
arXiv:1003.3288, arXiv:0910.4218(Lat09).

- Test on
  - CPU: Core i7 920 (2.67GHz, 4 core)
    - DP: 43 GFlops, SP: 85 GFlops (peak)
  - 2xGPUs: 2 x (Nvidia GeForce GTX 285)
    - 240 core @ 1.48 GHz, SP: 710 GFlops (peak)
  - OS, compiler
    - CentOS 5.2 (Linux), Intel Fortran / C++, CUDA 2.3.
  - Cost, BTO one PC box + 2GPU cards + Intel compiler.

- Schrodinger Functional (SF) simulation with N_f=10 QCD.
  - Quark assignment (Using blocked BiCGStab solver algorithm)
    - 2GPU case



      - GPU#1 = 3 solver calls, GPU#2 = 2 solver calls, in a single MD step.
    - 1GPU case
      - GPU#1 = 5 solver calls.

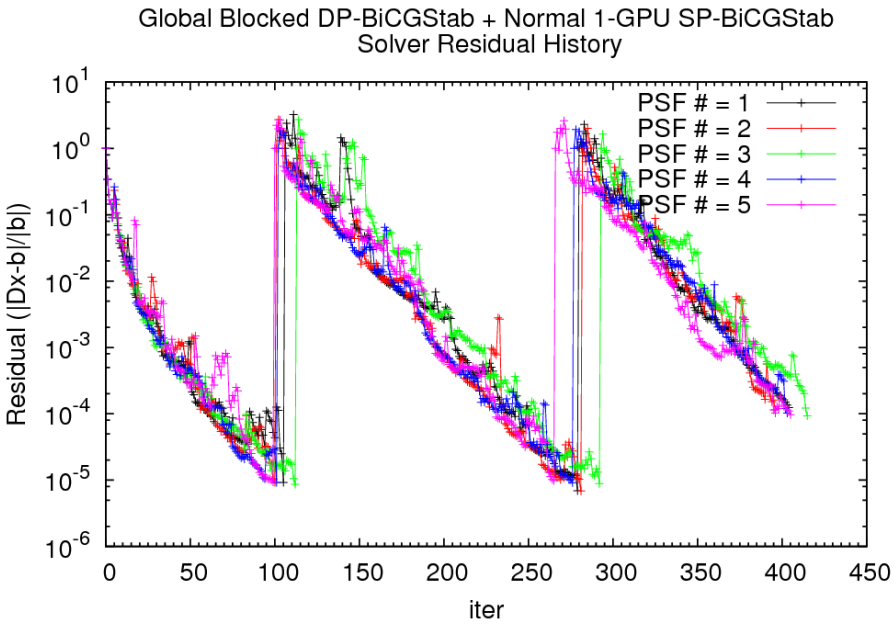


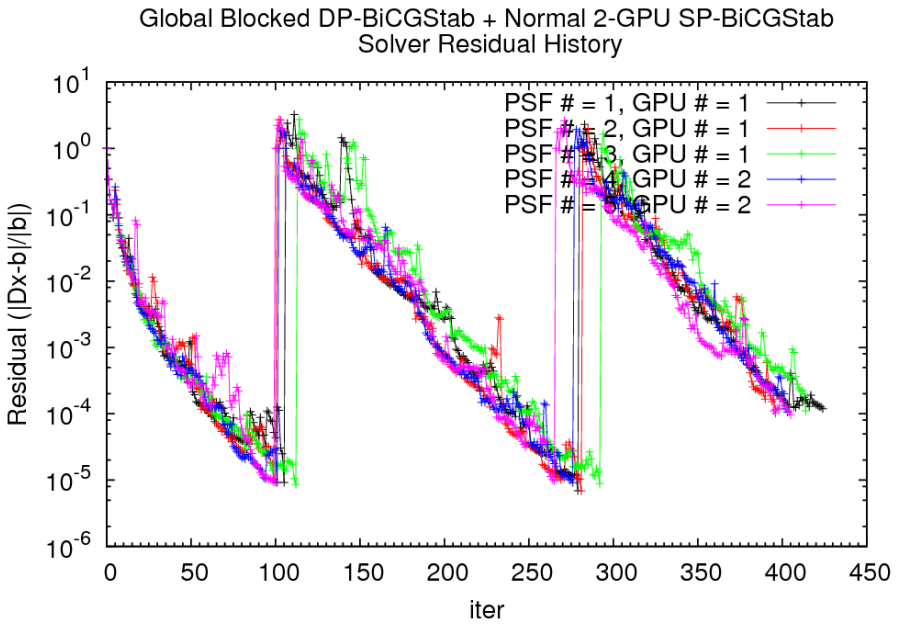
Ideally 40 %(3/5 = 0.6) Timing improvement is expected.

Quark pair is always assigned to single external scalar field.

$16^4,\ \beta = 4.52,\ \kappa = 0.15805,\ N_F = 10$
$g^2_{SF} \approx 10,\ \ aM_{PCAC} \approx 0.001$

Effect of Multiple GPUs（1GPU⇒2GPU）





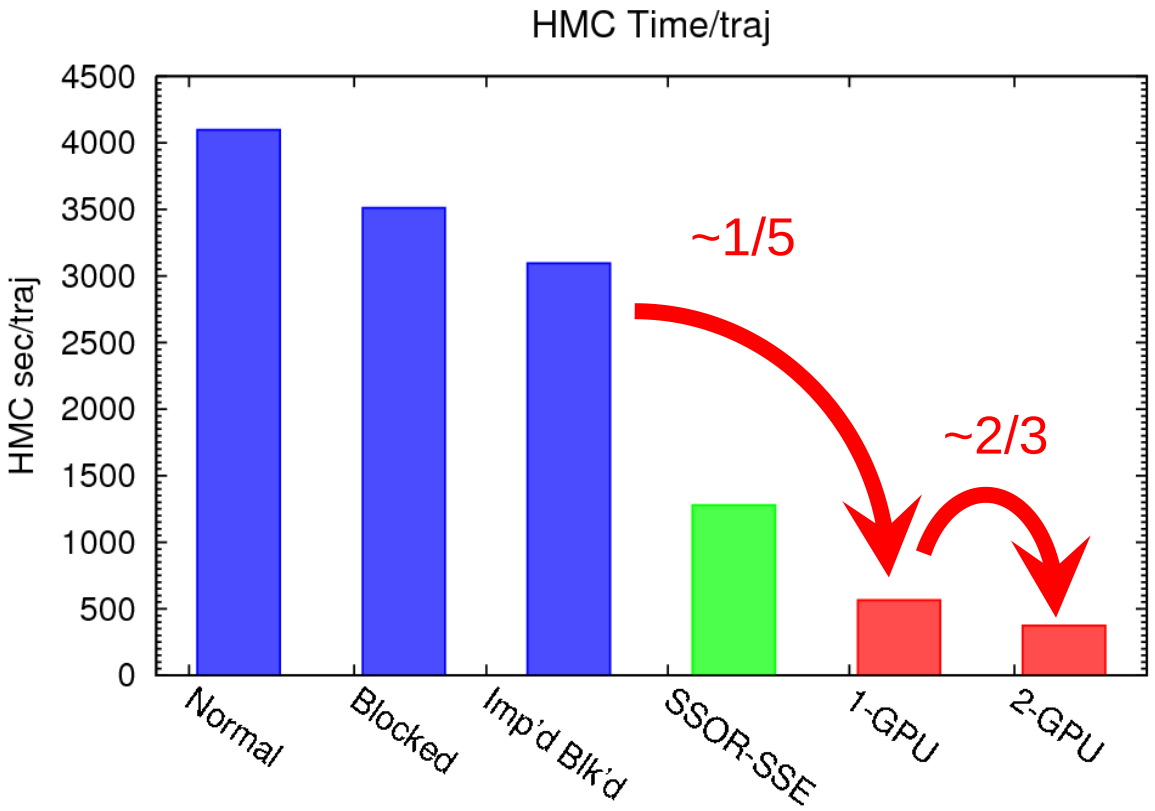
GPU solver is calle 5 times in a CPU solver iteration.
Full timing : 4.06 sec

GPU solver is called 3 times for GPU#1 and 2 times for GPU#2.
Full Timing : 2.69 sec

We observed 34% reduction in timing.
(Ideally it should be 3/5= 0.6 =40% reduction)

$16^4,\ \beta = 4.52,\ \kappa = 0.15805,\ N_F = 10$
$g^2_{\mathrm{SF}} \approx 10,\ \ aM_{\mathrm{PCAC}} \approx 0.001$

- Timing comparison for single HMC step.



I omitted the details of CPU side improvements with multi-core and mixed precision preconditioning.

We accomplished a factor 2-3 improvement for CPU only. But this does not win GPU.

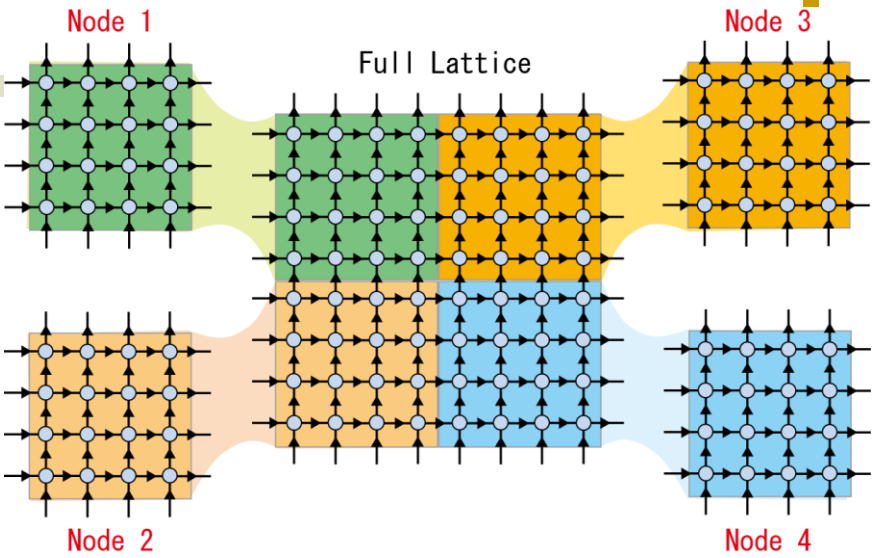
GPU is still faster than CPU code.

Using 2 GPUs we can accelerate (reduce 34%) in the timing.

6. Towards parallel GPU computation

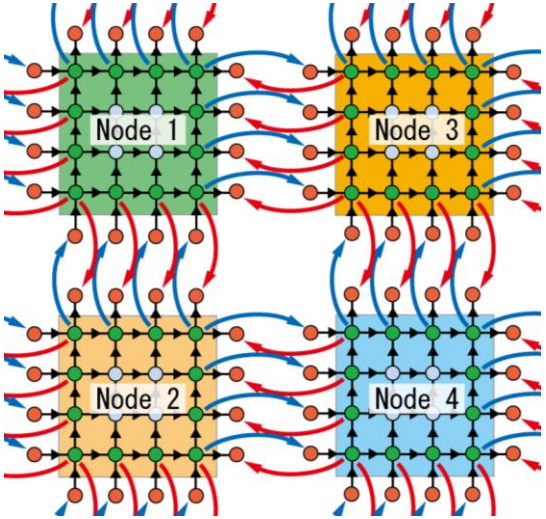
- To catch up the most advanced LQCD simulation we need true parallel GPU computation. O(10)Tflops machine for 32^4 lattice.
- Like in the parallel computation, the communication overhead is important.
- Unfortunately, GPU⇔GPU direct communication device is not available in COTS GPU cards.
- Three steps, GPU⇔CPU⇔CPU⇔GPU, communication is required in general.
- We investigated this overhead for LQCD application (*D[U]x=b* solver).

LQCD parallelization



- Data distribution:

- Data communication:
  - Exchange edge site data
  - Communication hiding behind internal site computation.
  - Edge site computation after receiving the adjacent site data from next GPUs.



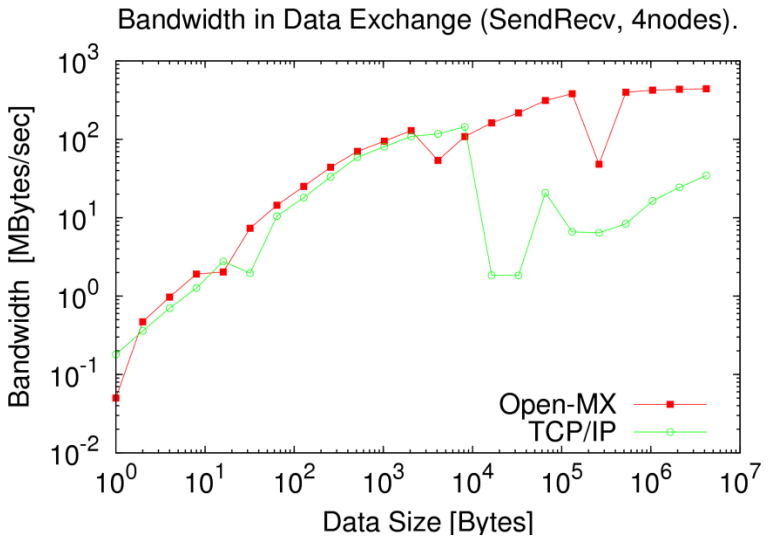
Parallel GPU implementation

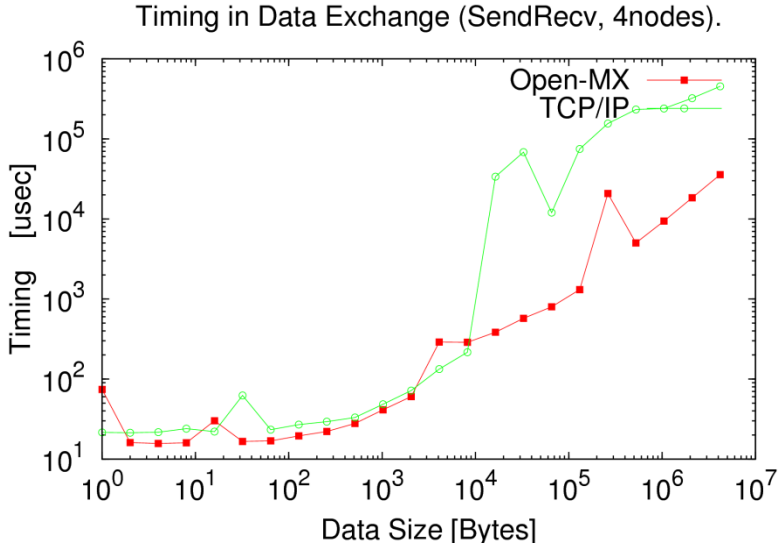
- Some results [K.-I.I. and Y.Osaki]
  - Machine: 4PC's with
  - CPU : Intel Core i7 920@2.67GHz (4cores)
  - GPU : GeForce GTX 285 × 2 (2GPUs in 1 node)
  - Memory 6GBytes
  - LAN Adapter : Intel Gigabit ET Quad Port Server. Poor man's network. (COTS, cheap)
  - CentOS 5.4, CUDA 2.3, OpenMPI
  - TCP/IP is too slow. Instead We employed Open-MX (Myrinet Express over Generic Ethernet Hardware) library [http://open-mx.gforge.inria.fr/].

Parallel GPU implementation

Some results [K.-I.I. and Y.Osaki]

- TCP/IP is too slow. Instead We employed Open-MX (Myrinet Express over Generic Ethernet Hardware) library [http://open-mx.gforge.inria.fr/].
- Network performance.

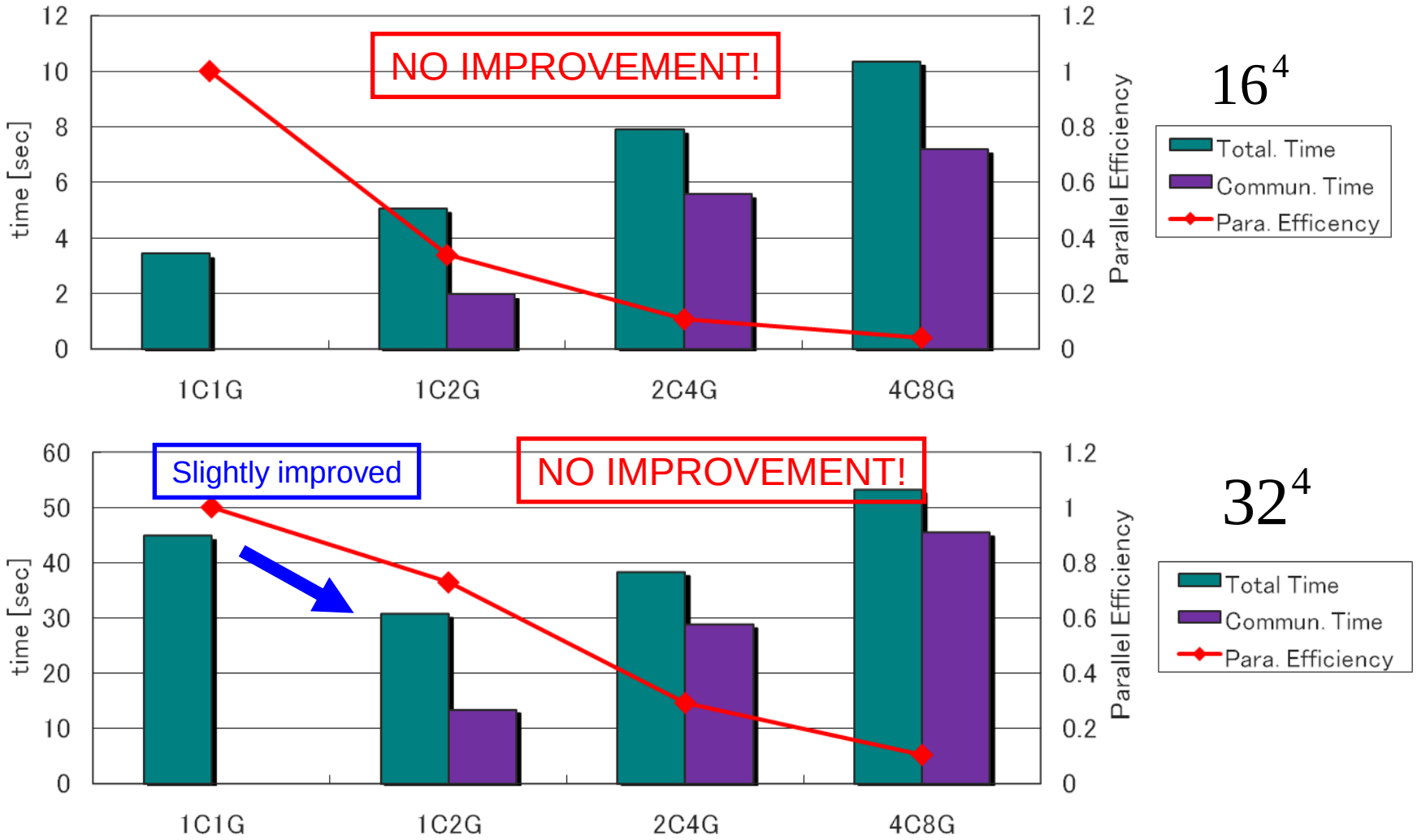




Max bandwidth ~440 MB/s, lowest latency 16usec (Open-MX)
~140 MB/s, 24usec (TCP/IP)

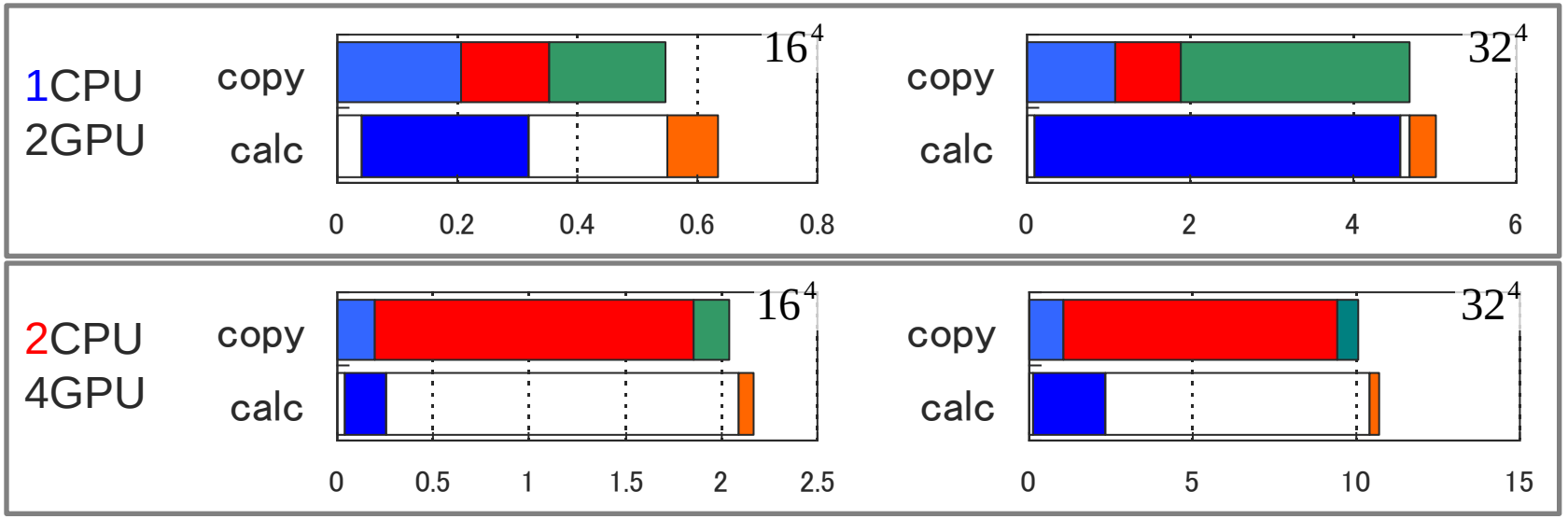
Strong Scaling test, solving D[U]x=b

- kappa=0.126
- csw=1.0
- accuracy=10^{-14}
- MixedPrecBiCGStab solver



Parallel efficiency is always worse than 1.

- Strong Scaling test
- For 16^4 lattice, there is no improvement.
  - Increasing num. of GPU and node, total time increase….
- For 32^4 lattice, there is almost no improvement.
  - Speed up by 30%: 1node1GPU => 1node 2GPU
  - No speed up: increasing nodes.
- Node-node communication (MPI) is still too slow.



MPI(CPU⇔CPU) time

Strong Scaling test

- We have tested:
- Simple domain decomposition with communication hiding.
- Gigabit Ethernet (x4 port) and Open-MX enhanced network.
- The network performance does not balance with the GPU speed. Communication hiding fails.
- We need Expensive network hardware (Infiniband, 10GEthernet, True Myrinet…..)?
- Another approach: Solver Preconditioning. Additive Schwarz Preconditioner.

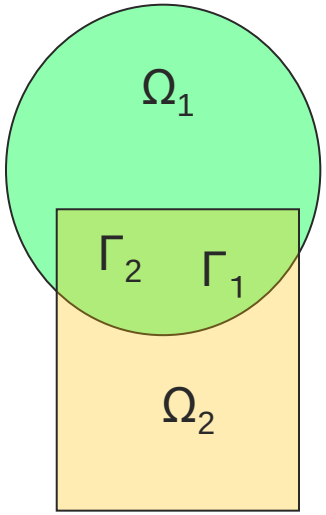
- Another approach: Solver Preconditioning.
- Overlapped Additive Schwarz Preconditioner
  - Schwarz iteration (common in fluid dynamics research?)

$$x_0, r = b - Ax_{0,}$$

$$\text{Solve}: \; A_{\Omega i} x_{\Omega i} = r_{\Omega i}, \; \text{in } \Omega_i$$

$$\text{Update}: \; x = x + \text{f}(x_{\Omega i}),$$

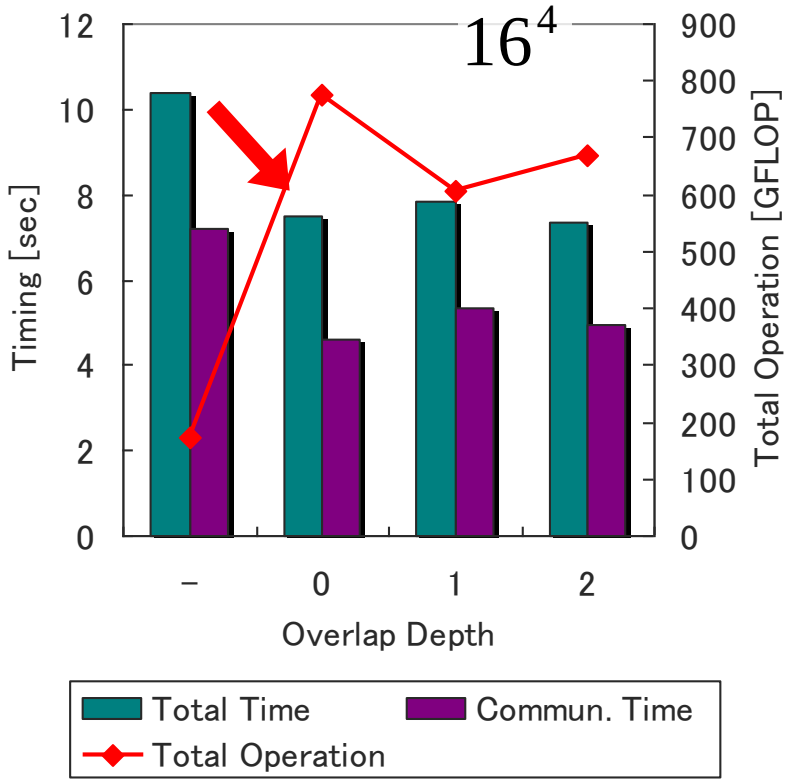
$$\text{Compute}: \; r = b - Ax,$$

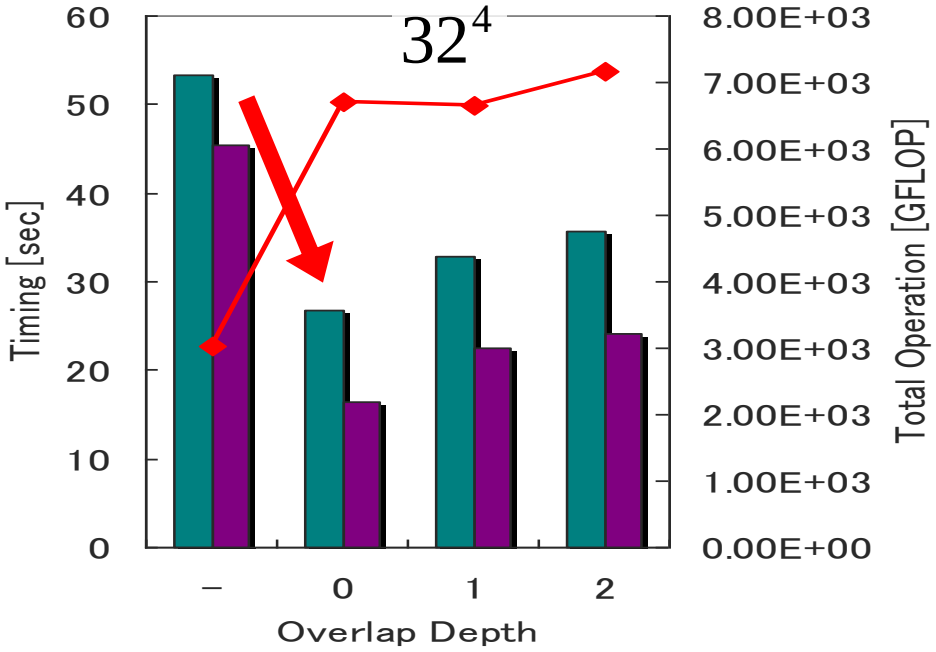


- Use this iteration as a preconditioner for iterative solver.
- Computation in each region Ω is completely independent.
- No communication is required. GPU supports this domain solver.
- Due to the overlapping domain region, total Flop count increases.
- Balance: Reduction of communication overhead and Flop count increase.

Overlapped Additive Schwarz Preconditioner

- Preliminary results [K.-I.I. and Y.Osaki]





Additive Schwarz preconditioner without overlapping is effective for larger lattice. Comm. Overhead is well reduced.

7. Summary

- Lattice QCD demands huge resource of computing power.
- Studying possibility of GPU acceleration has begun in LQCD community.
- Single/Multiple GPU/s in a node is very effective and attractive as shown here.

But, for more advanced QCD simulation,

- We need more survey on Parallel GPU computation and algorithm.

감사합니다 Thank you!