GPU computations for lattice QCD

BROOKHAVEN NATIONAL LABORATORY

Hyung-Jin Kim

Contents

1. GPU computation
   - GPU?
   - Hardware properties
   - Software properties
   - Examples
2. Lattice QCD on GPU
   - Lattice QCD
   - Algorithm
   - Lattice QCD Software
   - Examples
   - Performance

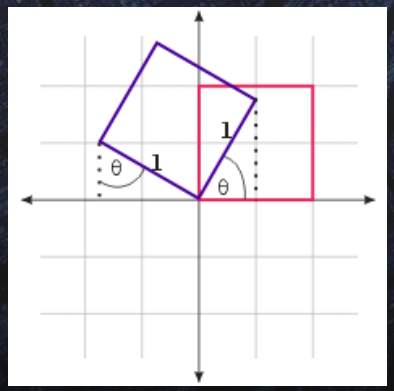
What is GPU?

GPU : Graphics Processing Unit

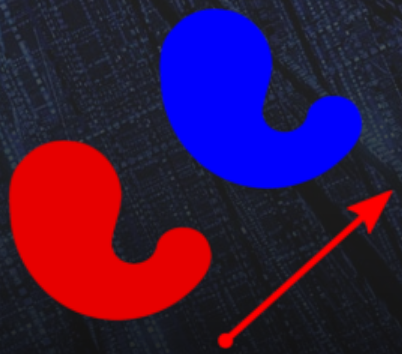
Device for managing all of image creation and displaying sequences

Basic operations : Same operations, but multiple data. Data parallel algorithm is needed

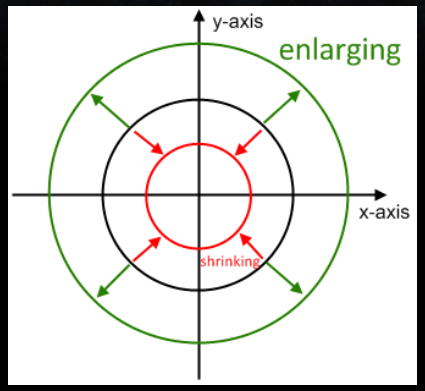
Rotation



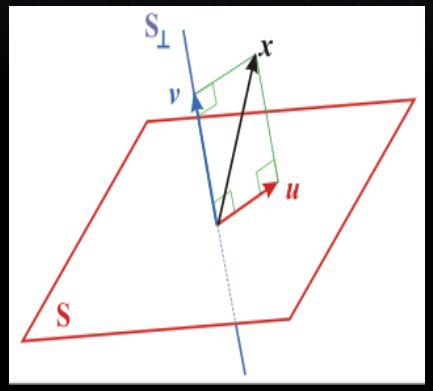
Parallel translation



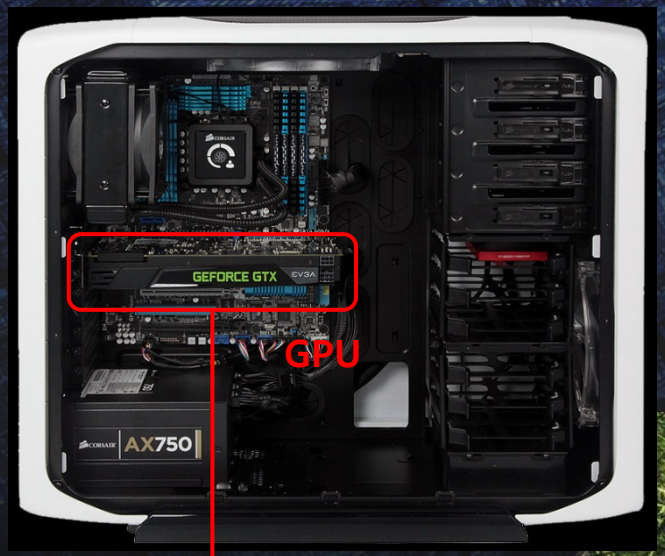
Scaling



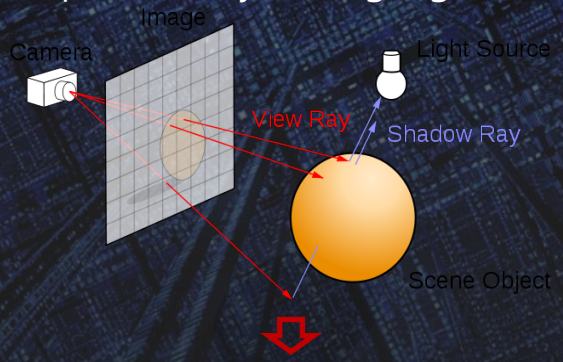
Projection



GPU for Image Processing



Complicated ray tracing algorithm,

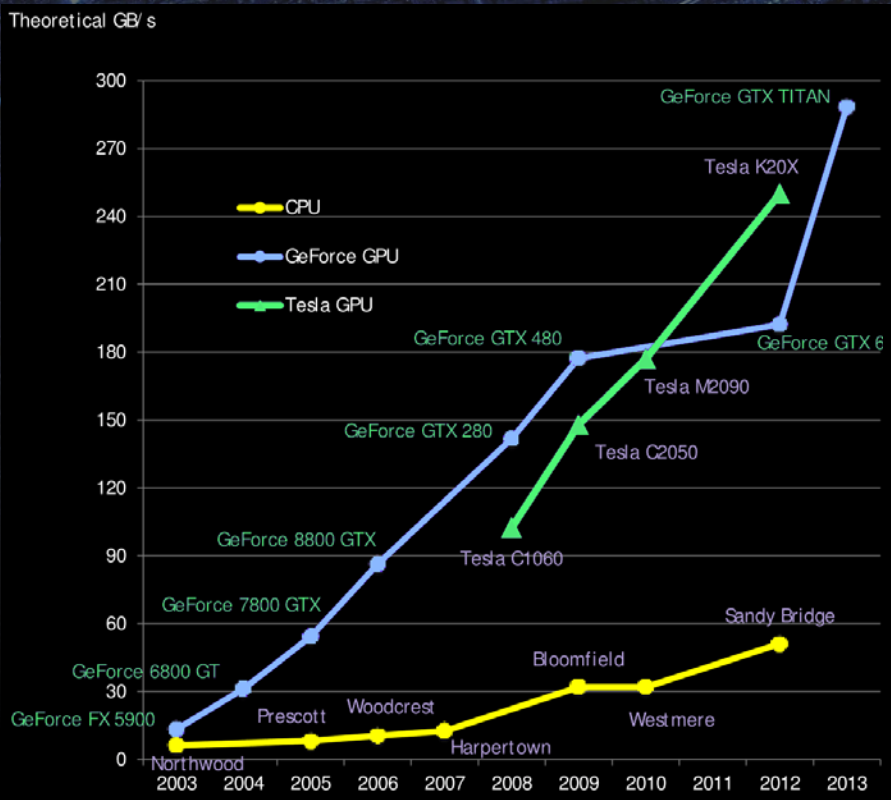


Crysis 3, EA games

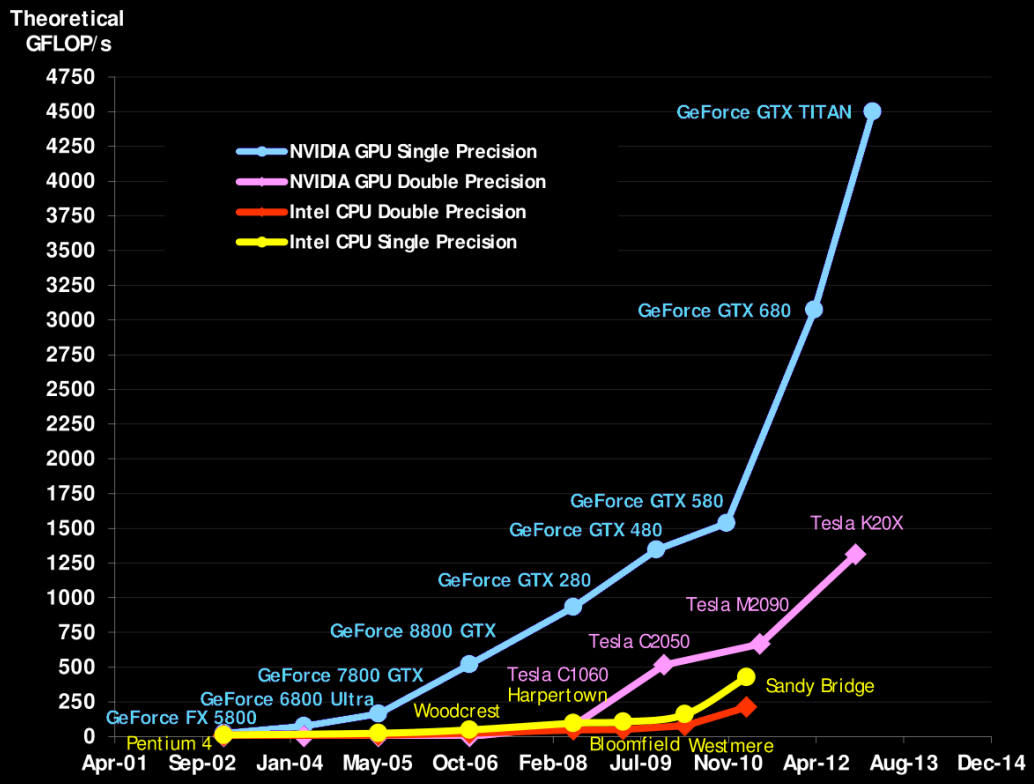
Modern GPUs can handle $O(10^{10})$ number of pixels / sec !

Why GPU?

Evolution of GPU performance



memory bandwidth, GPU vs CPU



FLOPS, GPU vs CPU

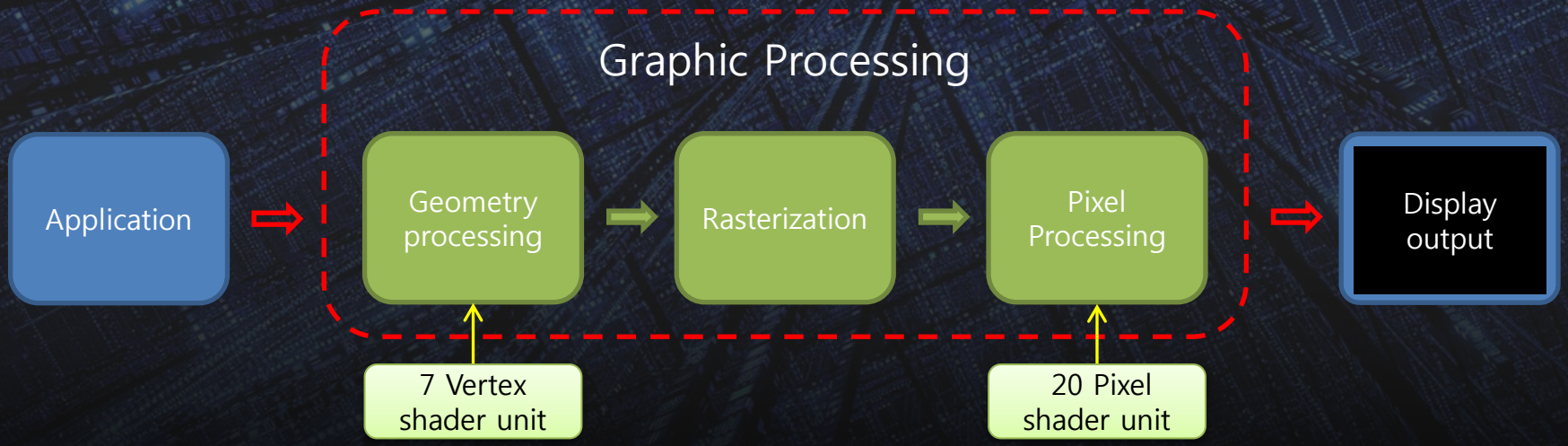
GPU is O(10) times faster than CPU in FLOPS,

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

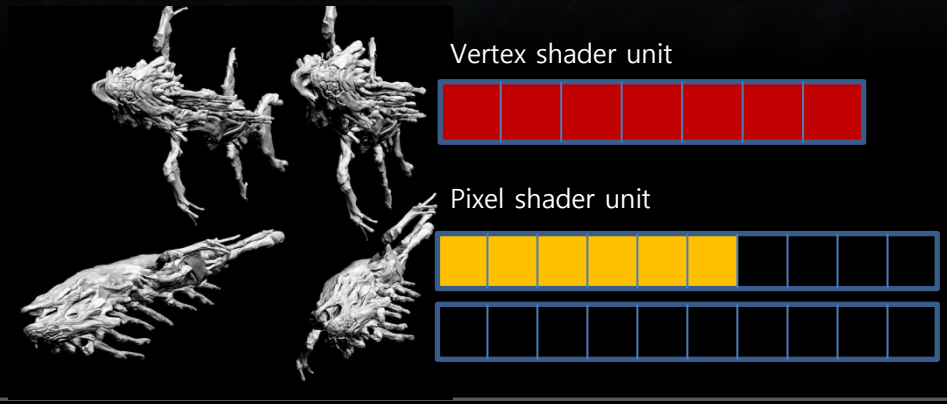
GPU for General Purpose Computation

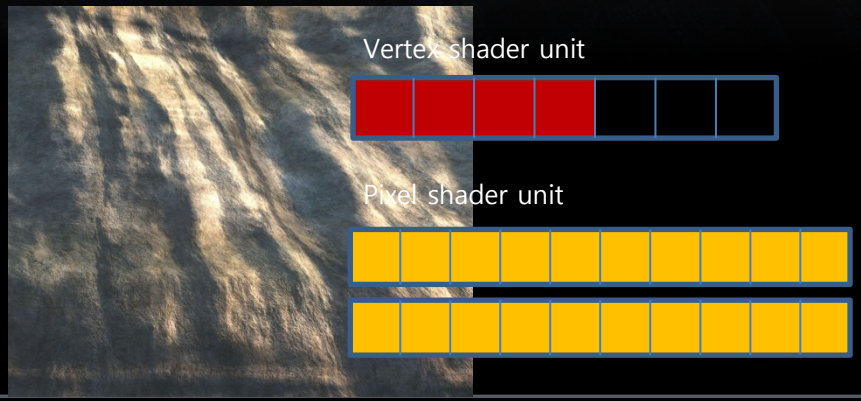
Before Nvidia G80 & ATI R600, The number of shader units are fixed

- General graphic processing pipeline



Ex) Nvidia, 7800gt has, 20 pixel shaders, 7 vertex shaders, 16 ROP





GPU for General Purpose Computation

Before Nvidia G80 & ATI R600, The number of shader units are fixed

- Discreet shader architecture makes general utilization of GPU difficult.
- Applicable algorithms are limited and poor GPU efficiency.
- But still, some application is worth to try.
  - arxiv.org/pdf/hep-lat/0611022
  - 1st lattice QCD computation on GPU, Implemented by OpenGL!
  - 13 GFLOPS on 7800 GTX

Lattice QCD as a video game

Győző I. Egri[a], Zoltán Fodor[abc], Christian Hoelbling[b], Sándor D. Katz[ab], Dániel Nógrádi[b] and Kálmán K. Szabó[b]

[a] *Institute for Theoretical Physics, Eötvös University, Budapest, Hungary*
[b] *Department of Physics, University of Wuppertal, Germany*
[c] *Department of Physics, University of California, San Diego, USA*

Abstract

The speed, bandwidth and cost characteristics of today's PC graphics cards make them an attractive target as general purpose computational platforms. High performance can be achieved also for lattice simulations but the actual implementation can be cumbersome. This paper outlines the architecture and programming model of modern graphics cards for the lattice practitioner with the goal of exploiting these chips for Monte Carlo simulations. Sample code is also given.

1 Introduction

The goal of every lattice field theorist is to use a calculational platform that maximizes the performance/price ratio. In this paper a competitive but so far unused and unappreciated (at least in the lattice community) architecture will be introduced.

So far the only available option was the usage of CPU-based platforms may it be individual PCs, PC clusters, dedicated supercomputers such as QCDOC or APE or commercial supercomputers such as BlueGene/L. The actual calculational task in all of these solutions is done by CPU's which significantly vary in terms of features but are similar in the sense that they all share a very general purpose architecture. In recent years a rapidly developing specialized architecture emerged from the graphics industry, Graphical Processing Units or GPU's which took over some of the calculational tasks of the CPU. These chips are designed to fulfill the needs of a graphics oriented audience (gamers, designers, etc.) and hence were specialized to the kind of task this set of users most frequently need i.e. graphical processing. However the complexity of this task grew to a level that general programmability of the chips was also required. The end product of this evolution is a high performance chip optimized for SIMD floating point operations on large vectors that can be utilized for general purpose calculations such as lattice field theory.

Figure 1 and 2 show sustained performances for both Wilson and staggered matrix multiplication on various lattice sizes and a comparison is given with SSE optimized CPU codes on an Intel P4. Considering the fact that the price of the current top GPU models are around $500 it becomes clear that they are very cost effective. For reference we give some numbers from figure 1 for the NVIDIA 8800 GTX card: 33 Gflops sustained performance on a $16^3 \times 60$ lattice using the Wilson kernel. Another good reason for investigating graphics hardware is the fact that the performance growth rate is still a steep exponential for GPU's [1].

The relatively low price tag of GPU's is of course the result of the large market value of their target audience (gamers, designers, etc.) which was also the reason why ordinary PCs proved to be very cost effective in the past [2, 3].

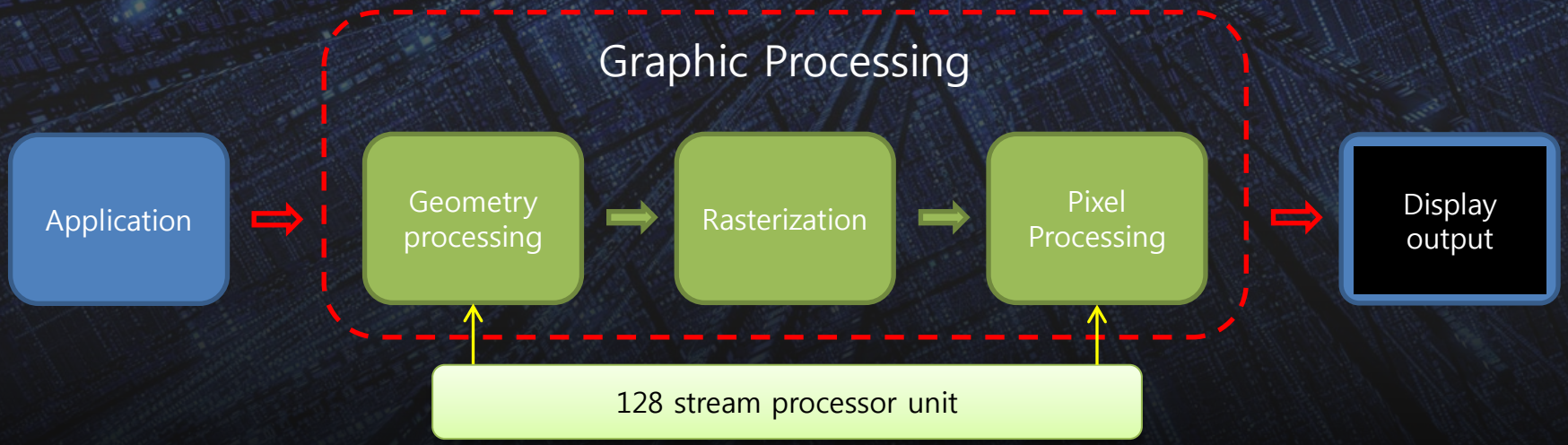
The question of scalability is of course an important one for any high performance calculational platform and for GPU's this aspect has not yet been explored in detail for lattice applications. The

1

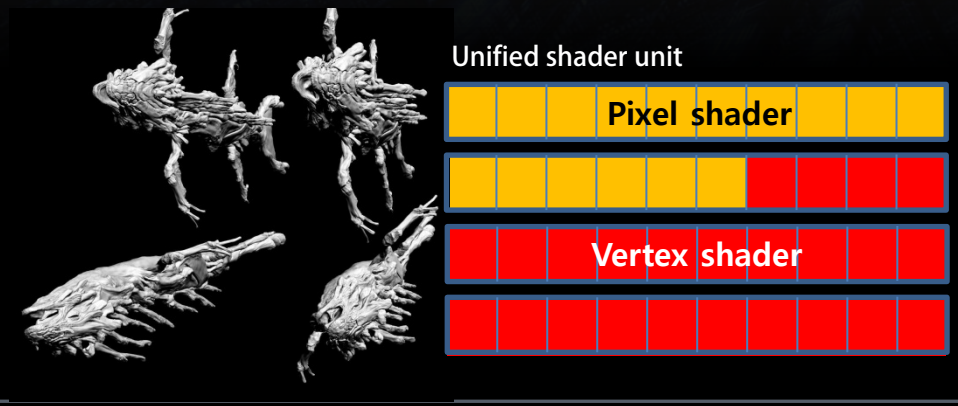
GPU for General Purpose Computation

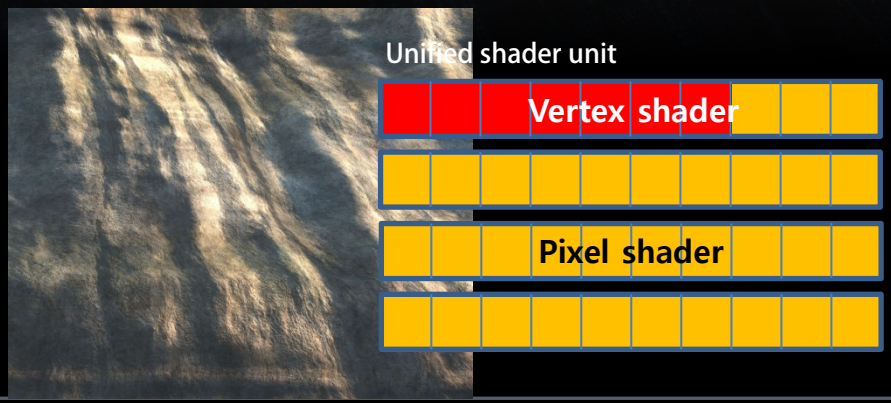
After Nvidia G80 & ATI R600, Era of the unified shader architecture~!

- General graphic processing pipeline



Ex) Nvidia, G80 has 128 streaming multiprocessors for unified shader





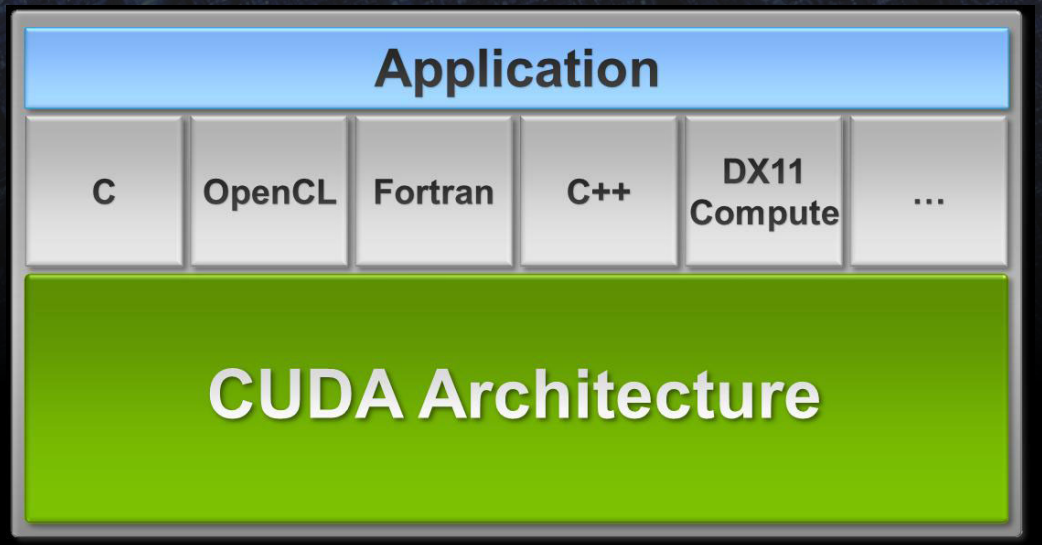
GPU for General Purpose Computation

After Nvidia G80 & ATI R600, Era of the unified shader architecture~!

- Various functions can be processed with the same device unit
- High GPU efficiency
- More flexible hardware and software structure can be easily applied for general purpose computation
- CUDA, Open CL provide a programming APIs for general purpose computation

Compute Unified Device Architecture

- Most widely used GPU Parallel computing platform and programming model invented by NVIDIA
- CUDA provides C/C++ extension, Fortran, DX11 Direct compute and python APIs for GPU application
- Well established references and developer support(long history)
- A lot of developed applications
- Available in nVIDIA device only



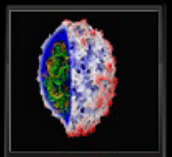
CUDA

CUDA Applications

Life Sciences	Medical Equipment	Productivity / Misc	Oil and Gas	EDA	Auto-motive	Finance	Communi-cations
FDA	GE Healthcare	WRF Weather Modeling	Hess	Synopsys	Renault	Symcor	Nokia
Robarts Research	Seimens	OptiTex	TOTAL	Nascentric		Level 3	RIM
Medtronic	Techniscan	Tech-X	CGG/Veritas	Gauda		SciComp	Philips
AGC	Boston Scientific	LabView	Chevron	CST		Hanweck	Samsung
Evolved machines	Eli Lilly	Elemental Technologies	Headwave	Agilent		QuantCatalyst	LG
Smith-Waterman DNA sequencing		Dimensional Imaging	Acceleware				Sony Ericsson
AutoDock		Manifold	Seismic City				NTT DoCoMo
NAMD/VMD		Digisens	P-Wave Seismic Imaging				Mitsubishi
Folding@Home		General Mills					Hitachi
CMATCH string matching							Radio Research Laboratory
							US Air Force

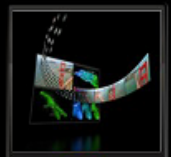
146X

Interactive visualization of volumetric white matter connectivity[1]



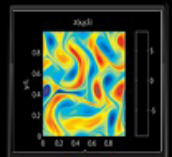
36X

Ionic placement for molecular dynamics simulation on GPU[2]



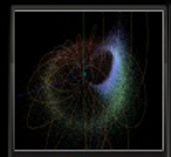
18X

Transcoding HD video stream to H.264 for portable video[3]



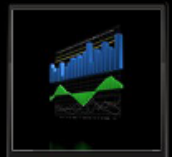
17X

Simulation in Matlab using .mex file CUDA function[4]



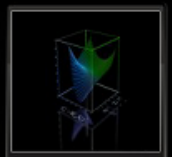
100X

Astrophysics N-body simulation[5]



149X

Financial simulation of LIBOR model with swaptions[6]



47X

GLAME@lab: M-script API for linear Algebra operations on GPU[7]

20X

Ultrasound medical imaging for cancer diagnostics[8]

24X

Highly optimized object oriented molecular dynamics[9]

30X

Cmatch exact string matching - find similar proteins & gene sequences[10]

Open CL

Official Web page
http://www.khronos.org/opencl/

OpenCL

Open Computing Language

- API for heterogeneous platforms consisting of CPUs, GPUs, DSPs or any other processors
- OpenCL is an open standard maintained by Khronos Group
- It is adopted by Intel, AMD, Nvidia, IBM, Apple, Samsung and ARM.
- On 9 December 2008, ver 1.0 released.

  Not so many references yet

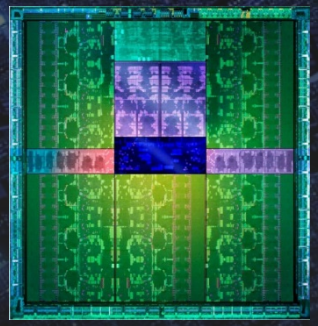
GPU powered Software

Useful GPU enabled softwares

- Magma(Matrix Algebra on GPU and Multicore Architectures) : LAPACK Lib.
- Mathematica (from ver 9) : support external CUDA function call
- MATLAB (from ver 2010b) : CUDA enabled math-functions, toolbox
- FDTD(Finite-difference time-domain) solver by acceleware
- VASP on GPU (in progress) - J Comput Chem. 2012 Dec 15;33(32)
- PGI CUDA-x86 compiler (from PGI 2010)
- Amber(from ver 11) on GPU

New Hardware

nVIDIA Kepler architecture (K20x, Geforce Titan)



7.1 billion of transistors(cf. Ivy Bridge CPU : 1.4 billion)

2688 of scalar processors, 6GB memory

3.95 TFLOPS(SP), 1.31 TFLOPS(DP), 250 GB/sec Mem. IO

Price : $1000~3800

15 SMXs in GK110 GPU

1 SMX :

192, float-32 ALU

64, float-64 ALU

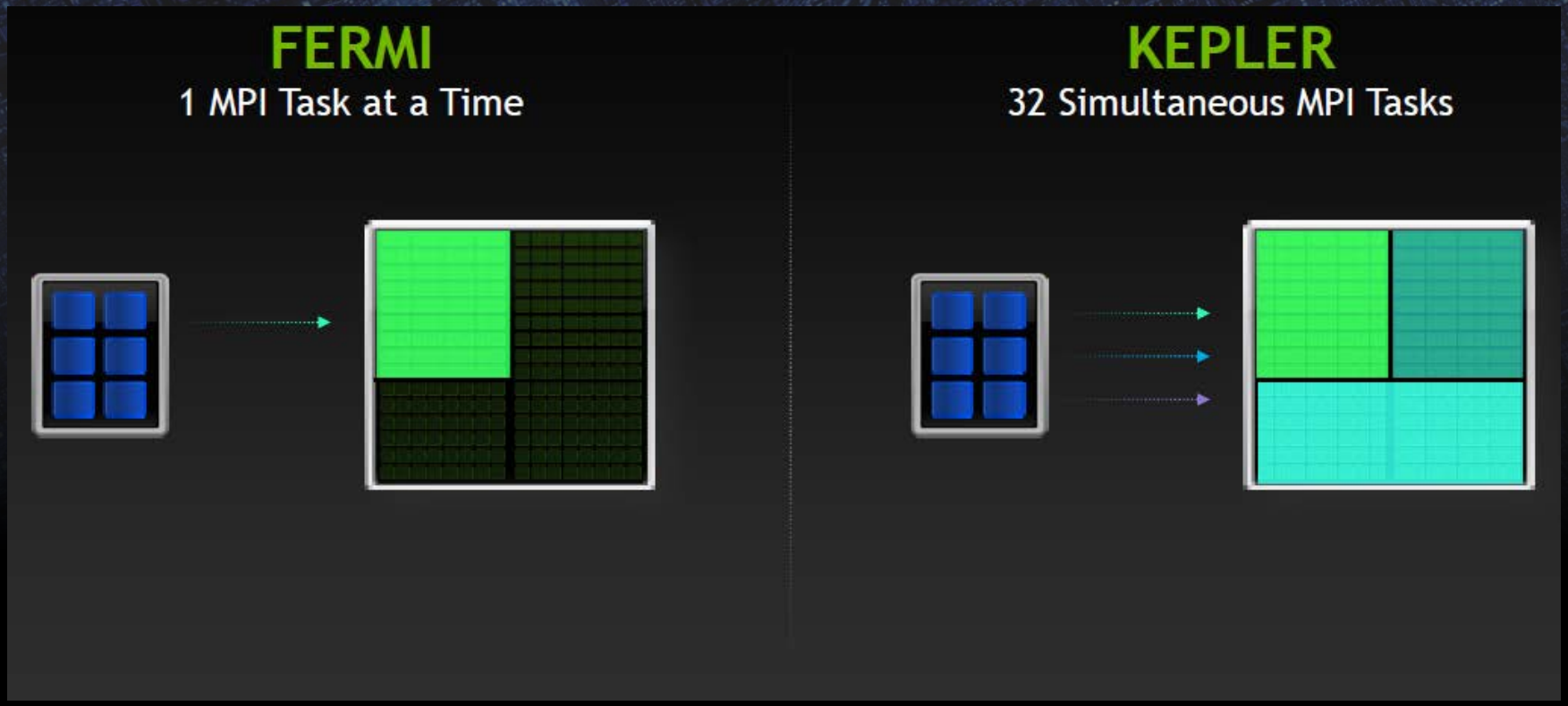
32 Special function unit

32 Mem. load/store unit

New Hardware

nVIDIA Kepler architecture

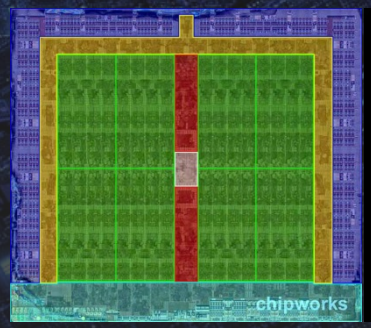
MIMD enabled : Hyper-Q



※Multiple kernels can be processed simultaneously within single GPU

New Hardware

AMD GCN(tahiti) architecture

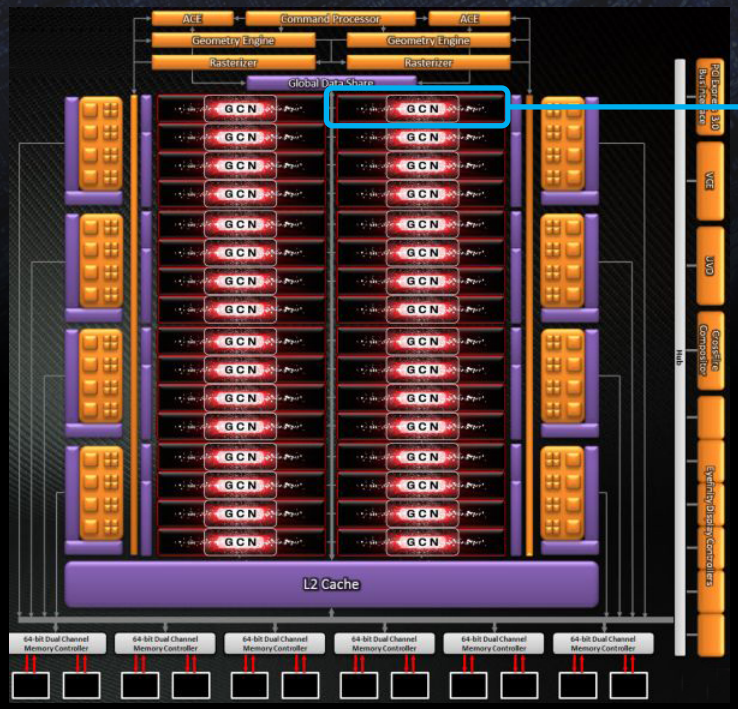


4.3 billion of transistors(cf. Ivy Bridge CPU : 1.4 billion)

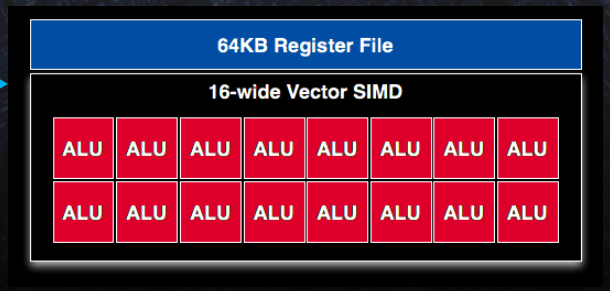
2048 of stream processors, 6GB memory

4.0 TFLOPS(SP), 1.0 TFLOPS(DP), 264 GB/sec Mem. IO

Price : $450~4000



GCN core



32 GCNs in Tahiti GPU

1 GCN has 4 SIMD(float-32) unit

+ 1 scalar unit(float-64)

SIMD unit has 16-way vector ALU

New Hardware

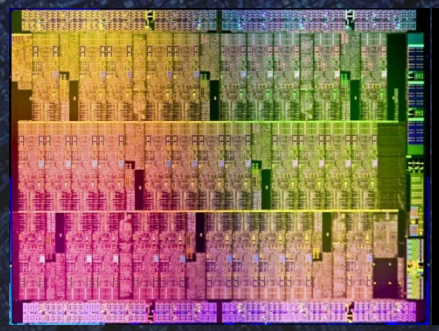
Intel Xeon Phi Processor

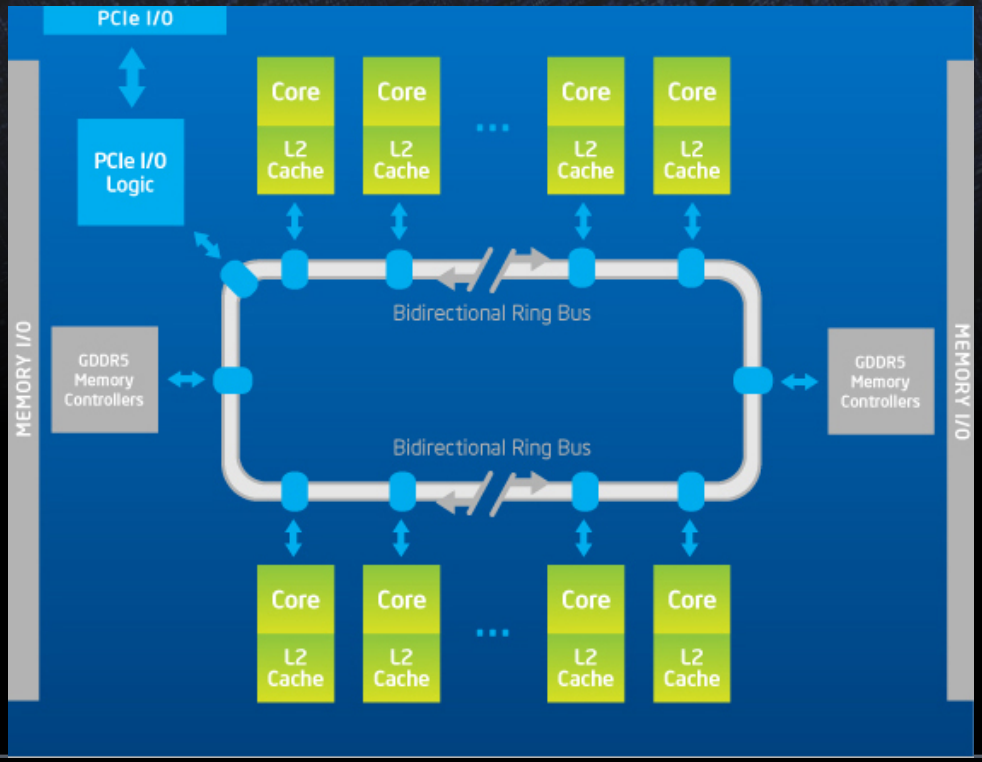
5.0 billion of transistors

60 cores of x86 processors, 8GB memory

2.0 TFLOPS(SP), 1.0 TFLOPS(DP),

320 GB/sec Mem. IO





Price : $2700~

X86-command compatibility.

New 512 bit-wide vector extension

- recompiling or optimization is needed

Large size(512KB/core) of L2 cache.

Large memory bandwidth,

but PCIE gen 2 limits PCIE IO

New Hardware

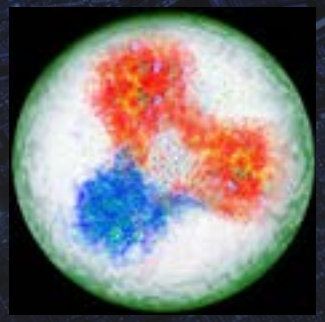
Platform comparison

	Xeon Phi	Xeon Sandy Bridge	GPU	BG/Q Power PC
SIMD Data layout	Yes	Yes	Yes	Yes
Vector processing	Yes	Yes	nVIDIA : No AMD : Yes	Yes
Blocking	Yes	Yes	Yes	Yes
Threading	Yes	Yes	Yes	Yes
Cache Management	Important	Moderate	Less important	Moderate
MPI + threads	Yes	Yes	No	Yes
Development (not Opt.)	Easy (x86 compatible)	Easy	Hard	Moderate

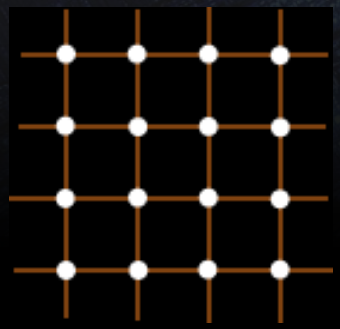
Lattice QCD on GPU

- Lattice QCD
- Algorithm
- Lattice QCD Software
- Examples
- Performance

Lattice QCD



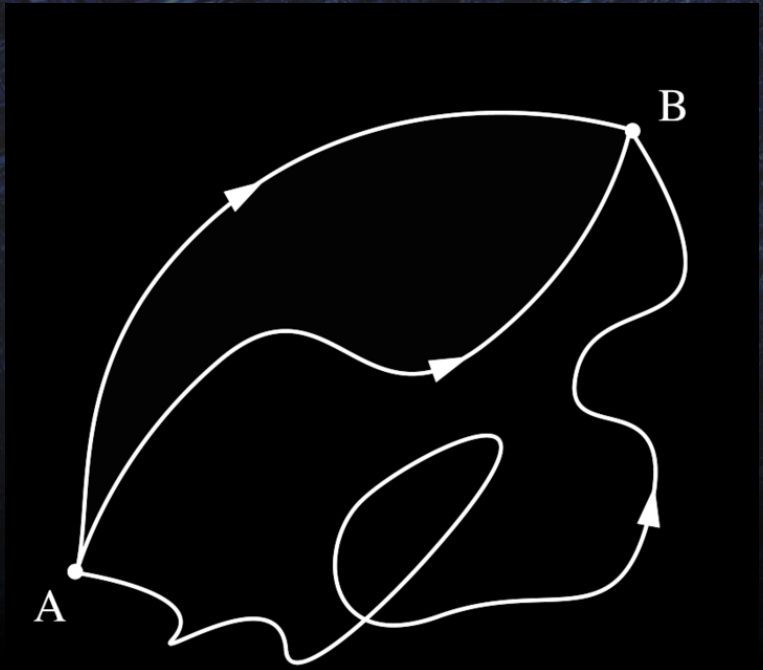
- **QCD ?**
  - Quantum field theory of the strong interaction.
  - Non perturbative at low energy region.



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

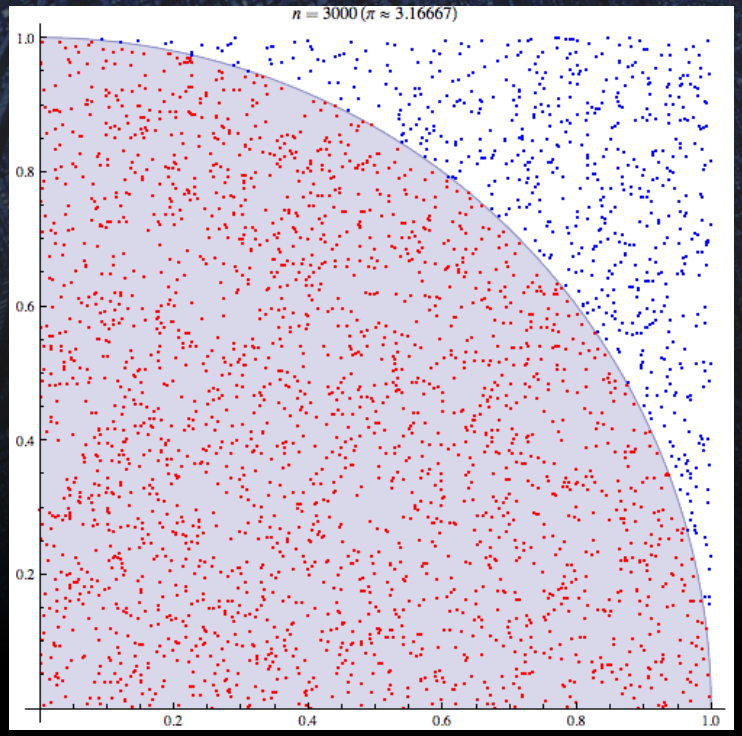
Path Integral & Monte Carlo Method

- Feynman`s Path Integral



Consider every path from A to B!

- Monte Carlo Method



Consider every point in 2D space!

Physical Observables

- Quantum mechanical observables

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

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

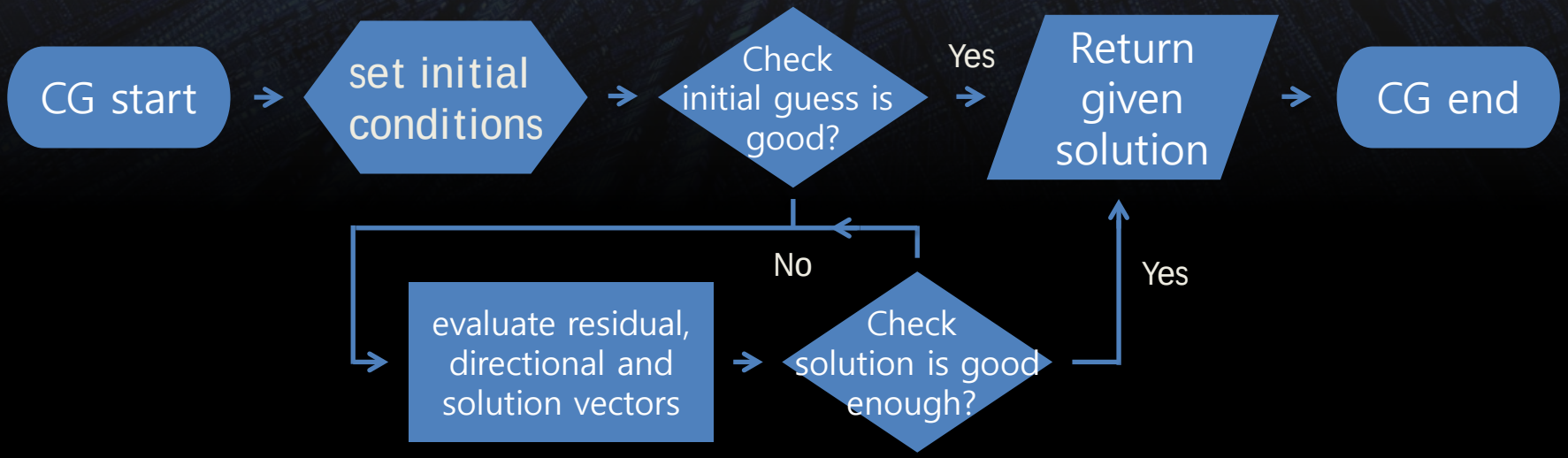
$$D(U)^{-1}\phi = \overbrace{\begin{pmatrix} a00 & a01 & a02 & \dots \\ a10 & a11 & a12 & \dots \\ a20 & a21 & a22 & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}^{-1}}^{O(10^6)} \left.\begin{pmatrix} b0 \\ b1 \\ b2 \\ \dots \end{pmatrix}\right\} O(10^6)$$

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

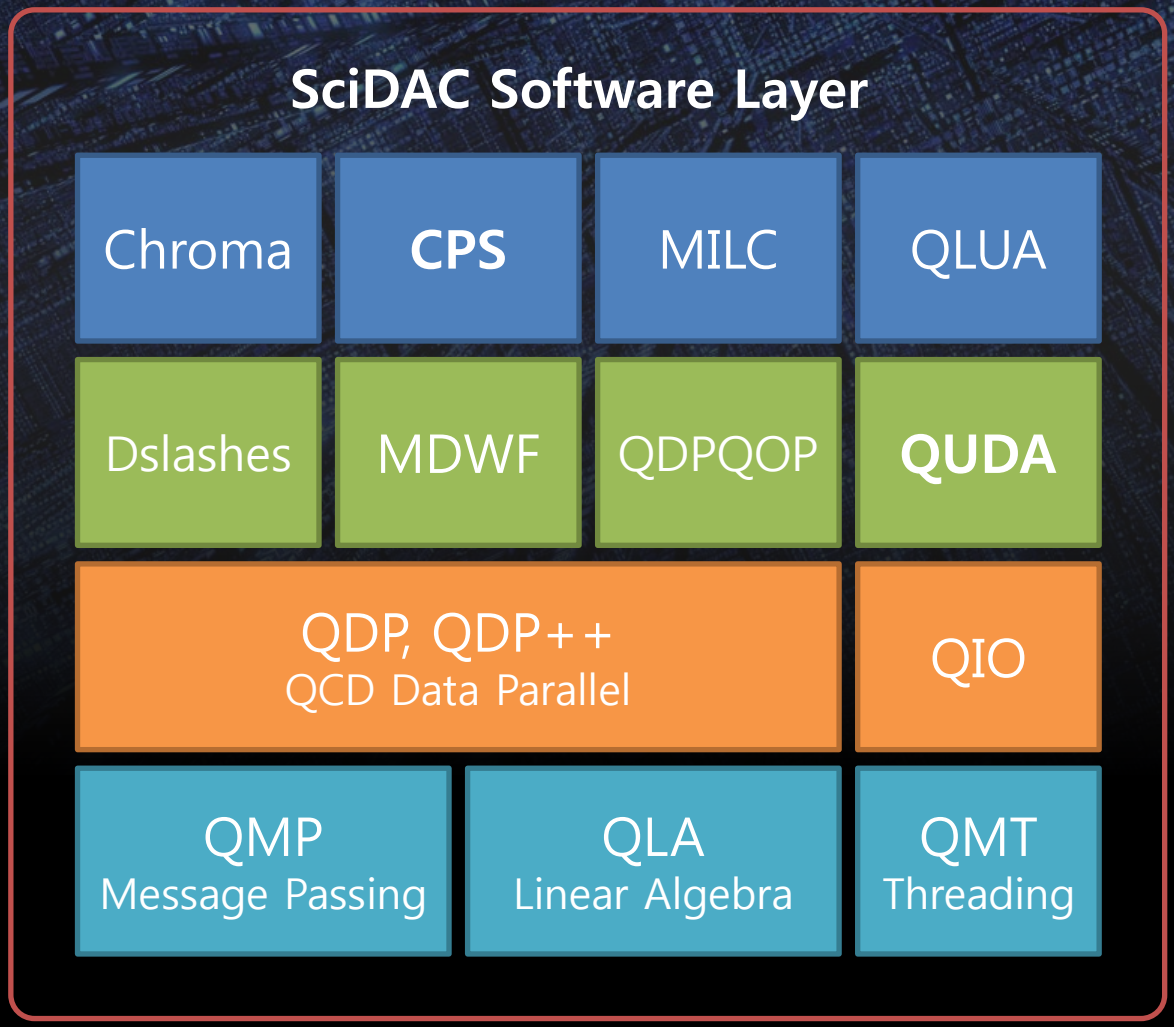
Wilson dirac operation - FLOPS/Bandwidth = 0.92

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
- Mixed precision(double, single, half) solver is available

Supported by SciDAC Scientific Discovery through Advanced Computing NVIDIA

- **Interface for external programs**

  @ PATH:`${QUDA_DIR}/lib/interface_quda.cpp`

  hardware initializing, data loading, parameter setting, matrix inverter, ….

  → Many functions are prepared for called from external programs

- **CPS Interface for QUDA** (ready for wilson & domain-wall type)

  @ PATH:`${CPS_DIR}/src/util/dirac/d_op_dwf/noarch/dwf_quda.C`

ex) …

```
inv_param.dirac_order = QUDA_CPS_WILSON_DIRAC_ORDER;
QudaGaugeParam gauge_param = newQudaGaugeParam();
inv_param.dslash_type = QUDA_DOMAIN_WALL_DSLASH;
inv_param.dirac_order = QUDA_CPS_WILSON_DIRAC_ORDER;
inv_param.inv_type = QUDA_CG_INVERTER;
…
```

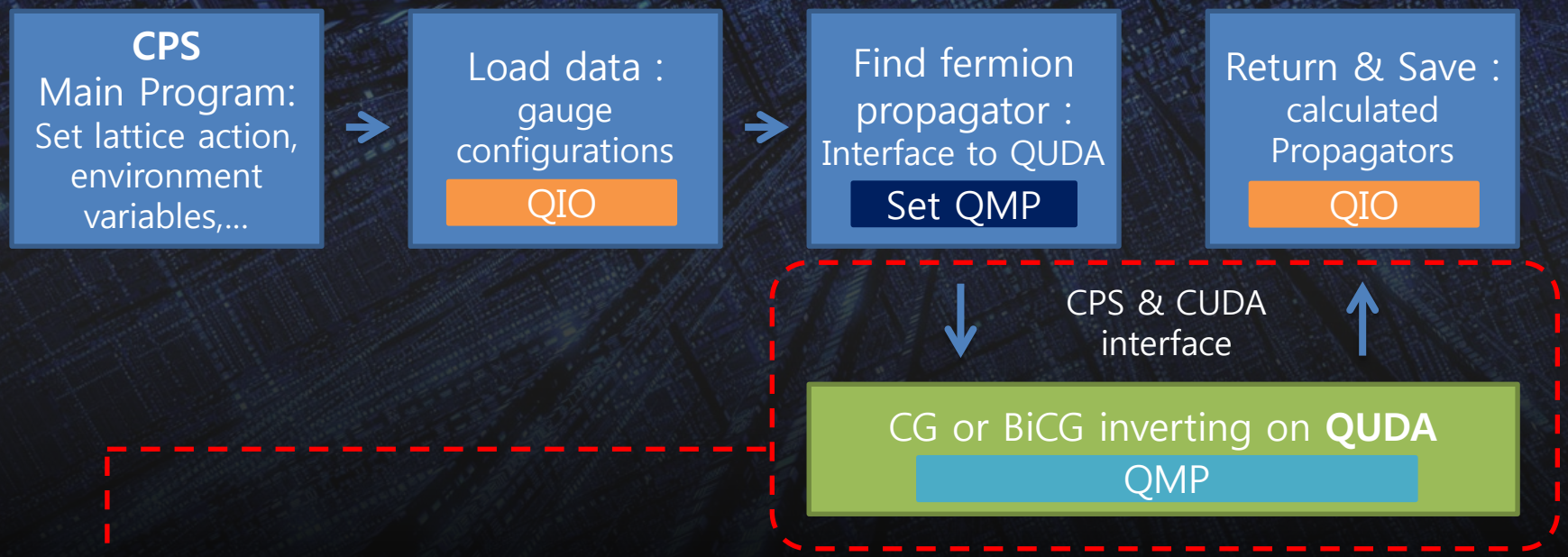
Optimization

Many optimization methods are applied

- Mixed precision matrix inverter(CG, BiCG)
- Domain decomposition method(Wilson Fermion)
- 8, 12 parameter reconstruction method for gauge field data
- Tuning mode can choose optimized value of threads & block number
- User can choose "compute capability" to appropriate to their GPU
- Optimized for each fermion schemes
- Program is still developing and will support future hardware also

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

How to Use?

- **Download & Install CUDA**
  - https://developer.nvidia.com/cuda-downloads
- **Download & Install QUDA**
  - 1. configure → 2. make
  - Ex) `./configure --enable-multi-gpu --with-qmp=${QMP_PATH} --with-cuda=${CUDA_PATH} --enable-gpu-arch=sm_20 …`
- **QUDA on CPS**
  - From 0.5.2.0 version of CPS, new configuration flags are introduced for QUDA

```
--enable-cuda=$(CUDA_INSTALL_DIR)
--enable-quda=$(QUDA_INSTALL_DIR)
```

  - Ex) `./configure --enable-cuda=/usr/local/cuda –enable-quda=/home/hjkim/quda …`

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)

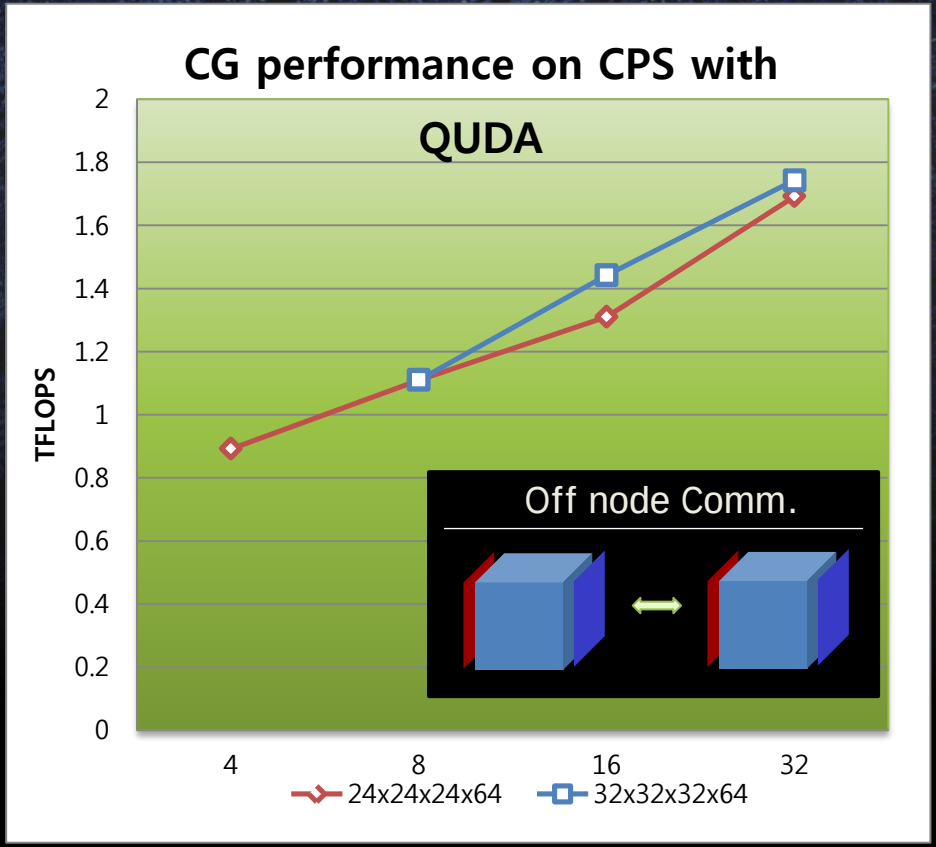
Scalability on virtual size of lattices

(= # 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

- **Mobius DWF**
  - Modified version of domain wall fermion
  - Reduced residual mass even in the smaller size of 5th dimension

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

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

Mobius DWF dirac equation

  - QUDA implementation is in progress

Future Work

- **Lanczos Algorithm**
  - Numerical algorithm for finding an eigenvector set
  - Highly dominated by memory IO
  - Still, 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
- CUDA, Open CL make GPU code developing easier
- QUDA is the best solution for the lattice QCD computations on GPU.
- With QUDA CG inverter, TFLOPS scale of computational performance can be easily achieved.