

# GPU Computing and Lattice QCD

Bálint Joó, Jefferson Lab

Extreme Computing and Its Implications for the  
Nuclear Physics/Applied Mathematics/Computer  
Science Interface  
Seattle, July 2011

# Contents

---

- Why GPUs and LQCD?
- What are GPUs and how to program them?
- “What have GPUs ever done for us?”
- What are the prospects for GPUs on the road to exascale?

# Why should I care about GPUs?

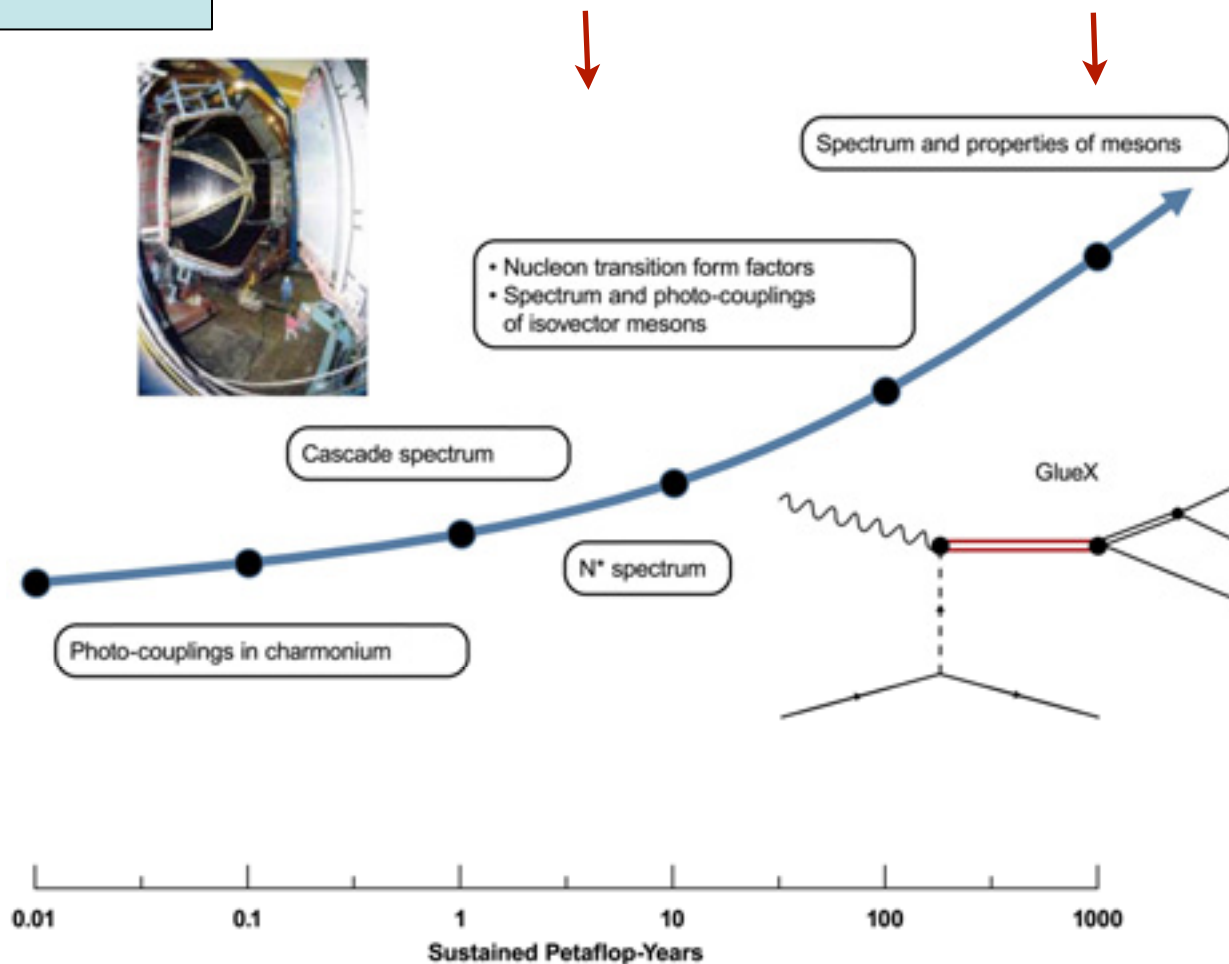
- GPUs deliver cost effective FLOPs for LQCD today.
  - You can have  $> 1$  TFlop sustained, under your desk
  - JLab clusters deliver  $\sim 2$  c/MFlop
- GPUs have features which are promised in future systems
  - High degree of concurrency (448-512 ‘cores’)
  - Complicated memory hierarchies
- GPUs are programmable (by the mainstream)
  - CUDA is well supported, and documented
  - OpenCL support is coming along
  - Compilers are providing other mechanisms (#pragma-s)
- Large scale GPU machines are already with us
  - Tianhe 1A, Keeneland, Edge etc.

# We need the Flops

Spectrum of hadrons

*Sustained Petascale*

*Exascale*



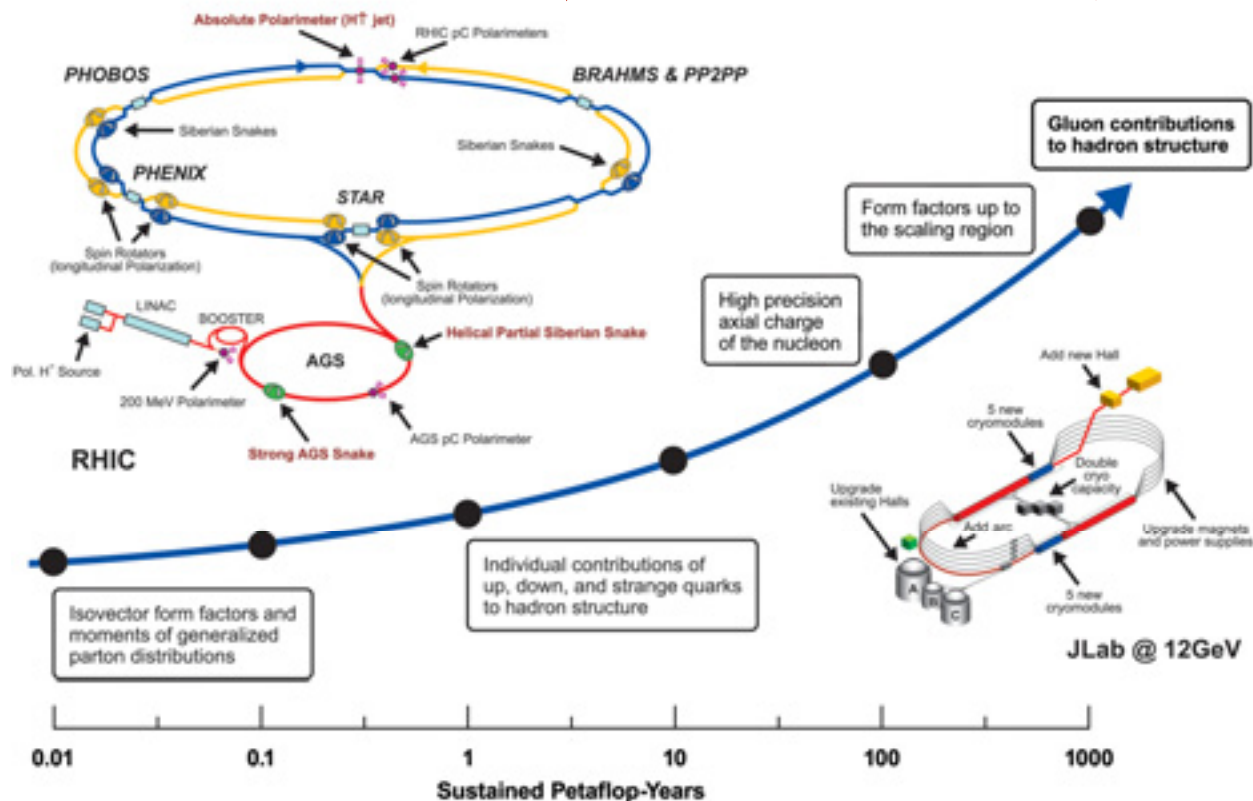
Scientific Grand Challenges: Forefront Questions in Nuclear Science and the Role of Computing at the Extreme Scale

# We need the Flops

Hadron Structure

*Sustained Petascale*

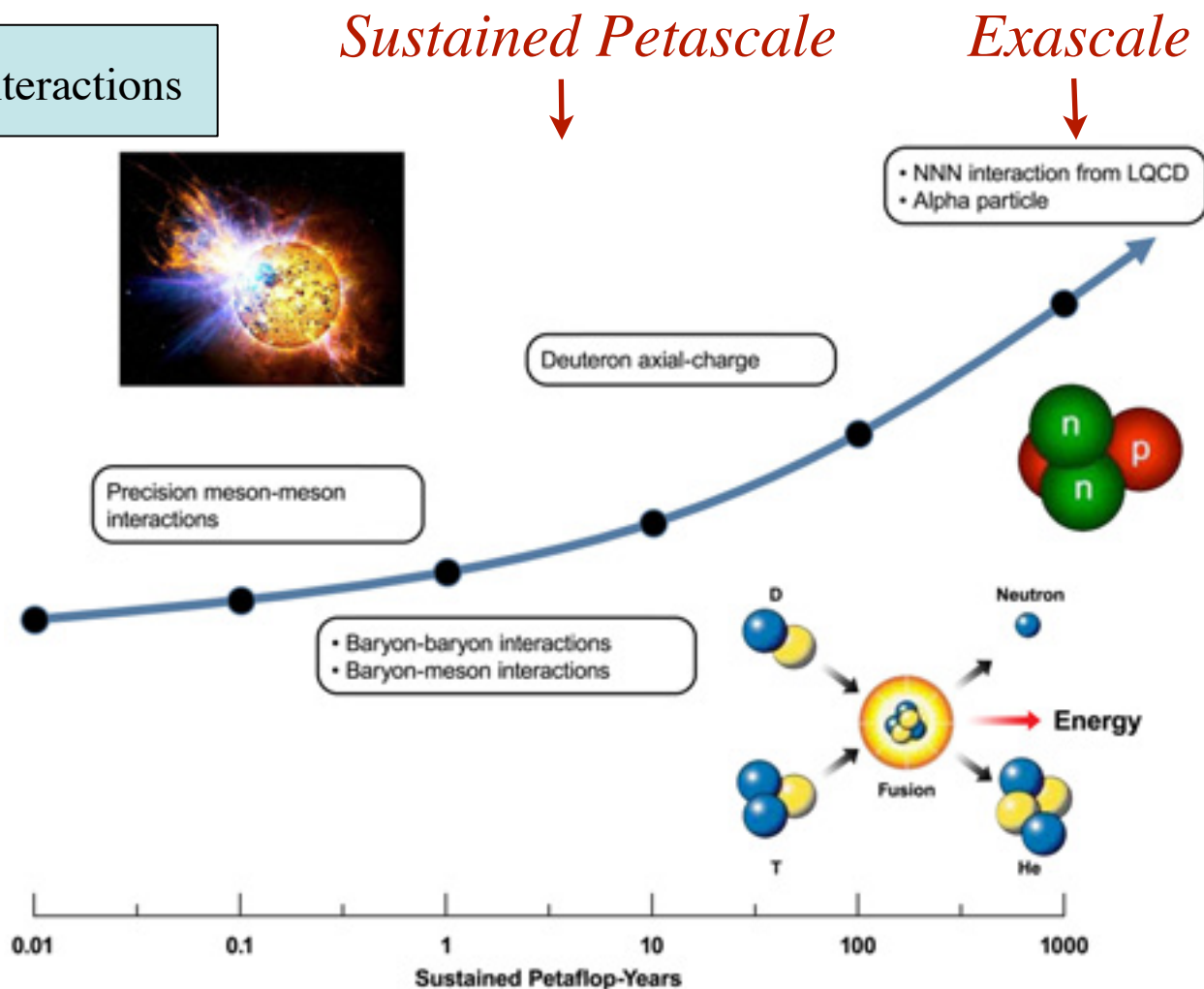
*Exascale*



Scientific Grand Challenges: Forefront Questions in Nuclear Science and the Role of Computing at the Extreme Scale

# We need the Flops

## Nuclear Interactions

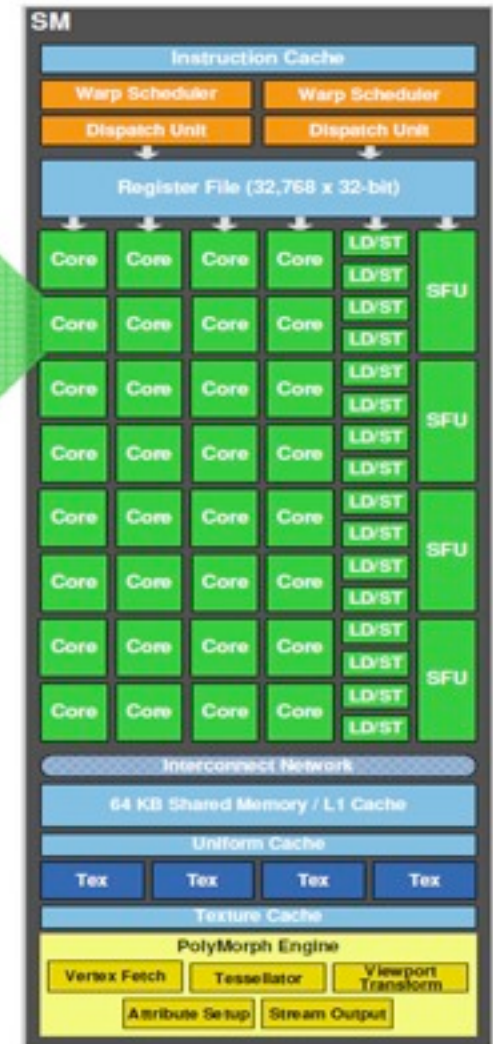
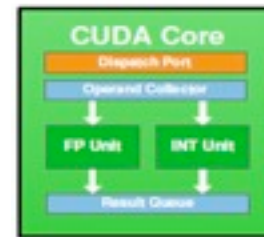


Scientific Grand Challenges: Forefront Questions in Nuclear Science and the Role of Computing at the Extreme Scale

# Anatomy of a Fermi GPU



Tesla M2090;  
512 CUDA cores  
x 2 Flops/clock  
x 1.3 GHz  
= 1.33 Tflops (SP)

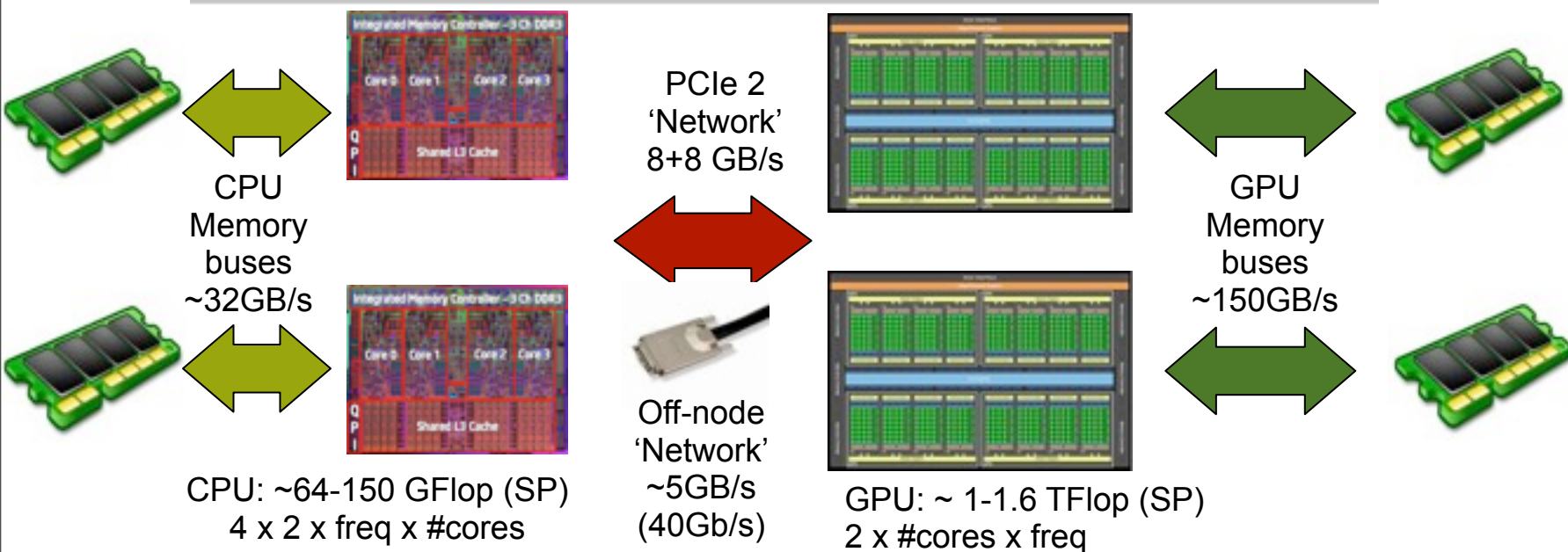


Streaming Multiprocessor (SM)

- NVIDIA GPU consists of Streaming Multiprocessors (SMs)
- SMs provide:
  - registers (32K 32-bit)
  - CUDA cores (32 per SM) – 1 SP mul-add per clock.
  - 64 KB Shared Memory (configured as memory/L1 cache)
  - Special Function units (for fast sin/cos/exp etc)
  - Hardware barrier within SM.
  - texture caches, thread dispatch logic etc.



# What does a full system look like?



- Regular, possibly multi-socket/multi-core CPUs
- One or more GPU devices on PCIe 'network' (bus(es), PLX etc )
- Typically CPU & GPU have separate memories
- Regular networking to other 'nodes'

*NB: 'Speeds and Feeds' shown are approximate*

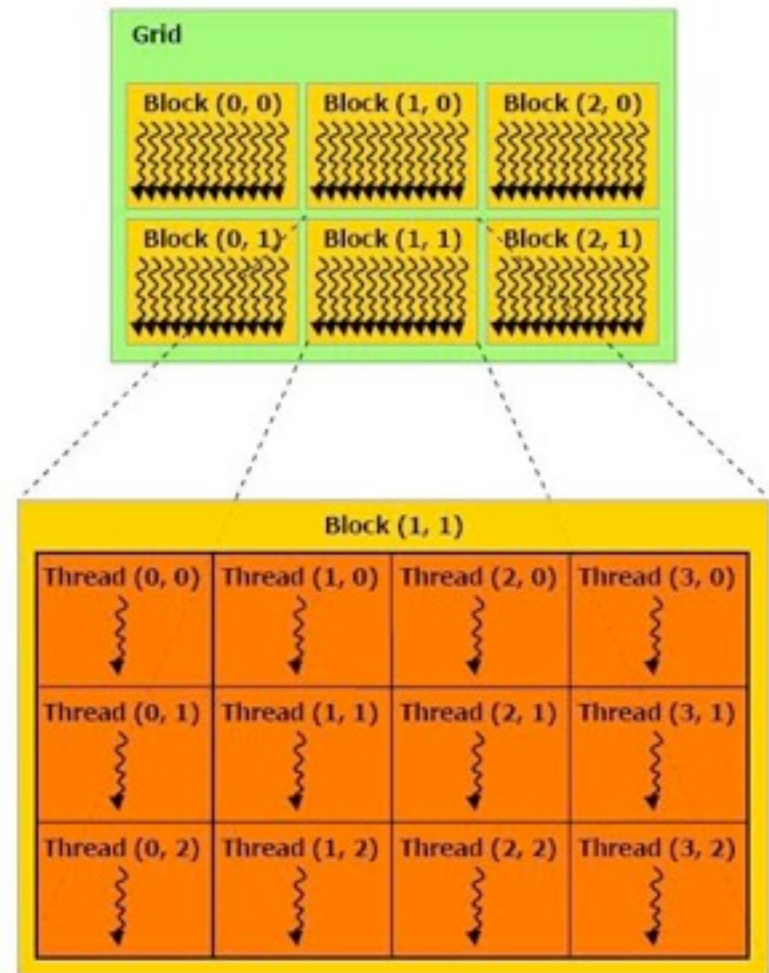


*JLab 10G cluster*



# The CUDA Thread Model

- user 'kernels' execute in a 'grid' of 'blocks' of 'threads'
  - block has ID in the grid
  - thread has ID in the block
- blocks are 'independent'
  - no synchronization between blocks
- threads within a block may cooperate
  - use shared memory
  - fast synchronization
- in H/W, blocks are mapped to SMs



# Example: Kernel to add two vectors

```
#include <cuda.h>
#include <cstdio>
#include <iostream>
```

Include cuda.h to access  
cuda API  
(may also need  
cuda\_runtime.h)

```
#define N 20
```

```
// Kernel to add vectors 'x' and 'y' into 'z'
// vectors are of length N elements
```

```
__global__
void add( float *z, float *x, float *y )
{
    int thread_id = threadIdx.x + blockIdx.x * blockDim.x;
    if( thread_id < N ) {
        z[ thread_id ] = x[ thread_id ] + y[ thread_id ];
    }
}
```

\_\_global\_\_ marks this as a kernel

Generate a global  
thread ID

These are device memory  
accesses

# Example: Host Code

```
int main(int argc, char *argv[])
{
    float host_x[N], host_y[N], host_z[N];
    float* device_x; float* device_y; float* device_z;

    for(int i=0; i < N; i++) {
        host_x[i]=(float)i;  host_y[i]=(float) (2*i);
    }

    cudaMalloc( &device_x, N*sizeof(float) );
    cudaMalloc( &device_y, N*sizeof(float) );
    cudaMalloc( &device_z, N*sizeof(float) );

    cudaMemcpy( device_x, host_x, N*sizeof(float), cudaMemcpyHostToDevice );
    cudaMemcpy( device_y, host_y, N*sizeof(float), cudaMemcpyHostToDevice );

    dim3 n_blocks; dim3 threads_per_block;
    n_blocks.x = 1; threads_per_block.x = N;

    add<<< threads_per_block, n_blocks >>>( device_z, device_x, device_y );

    cudaMemcpy( host_z, device_z, N*sizeof(float), cudaMemcpyDeviceToHost );

    cudaFree( device_x ); cudaFree( device_y ); cudaFree( device_z );
}
```

Allocate space on GPU  
and transfer data to GPU

Launch  
Kernel

Bring Data back from GPU and  
free space on device

# It began with ...

- Egri et al. Comput. Phys. Commun. 177:631-639, 2007
- with a preview at Lattice'06 (Tuscon, AZ)
- predated CUDA, using OpenGL graphics primitives

## Lattice QCD as a video game

Győző I. Egri<sup>a</sup>, Zoltán Fodor<sup>abc</sup>, Christian Hoelbling<sup>b</sup>,  
Sándor D. Katz<sup>ab</sup>, Dániel Nógrádi<sup>b</sup> and Kálmán K. Szabó<sup>b</sup>

<sup>a</sup>*Institute for Theoretical Physics, Eötvös University, Budapest, Hungary*

<sup>b</sup>*Department of Physics, University of Wuppertal, Germany*

<sup>c</sup>*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.

# GPUs and LQCD in the US

- 2008-2009: QUDA Library (QCD with CUDA)
  - GPU/Algorithms program at Boston Univ. led by Brower, Rebbi
  - Mike Clark, Ron Babich lead developers
  - Staggered branch (Gottlieb, Shi), DWF branch (Giedt)
- 2009-2010: Joined by Jefferson Lab
  - JLab deploys 9G/10G ARRA clusters
  - QUDA integration with Chroma (Wilson & Clover solvers)
  - Multi-GPU parallelization (T-direction)
  - Strong Scale QUDA to 8-16 GPUs, Weak Scale to 32 GPUs
- 2011: QUDA Unified diverse branches, actions -> community
  - Parallelize QUDA in any direction - strong scale to 256 GPUs
- Independent GPU code for Overlap Fermions: A. Alexandru (GWU)

# LQCD And GPUs around the world

- Budapest-Wuppertal group: Multi-GPU
  - several actions (staggered, Wilson) and representations (e.g. sextet)
- Pisa Group: Multi-GPU
  - Staggered RHMC on GPUs, CUDA & OpenCL
- Taiwan: DWF solver, Single GPU?
- Portugal: SU(3) & SU(4) pure gauge
  - Multi-GPU in a single node using OpenMP
- The APEnet+ project (Rome)
  - Scalable network to connect GPUs
- Edinburgh: QDP++ for GPUs
- Japan: Multi-GPUs + blocking techniques,  $N_f = 10$  simulations, DD
- ... (apologies to the groups I didn't mention)...



# QUDA Community

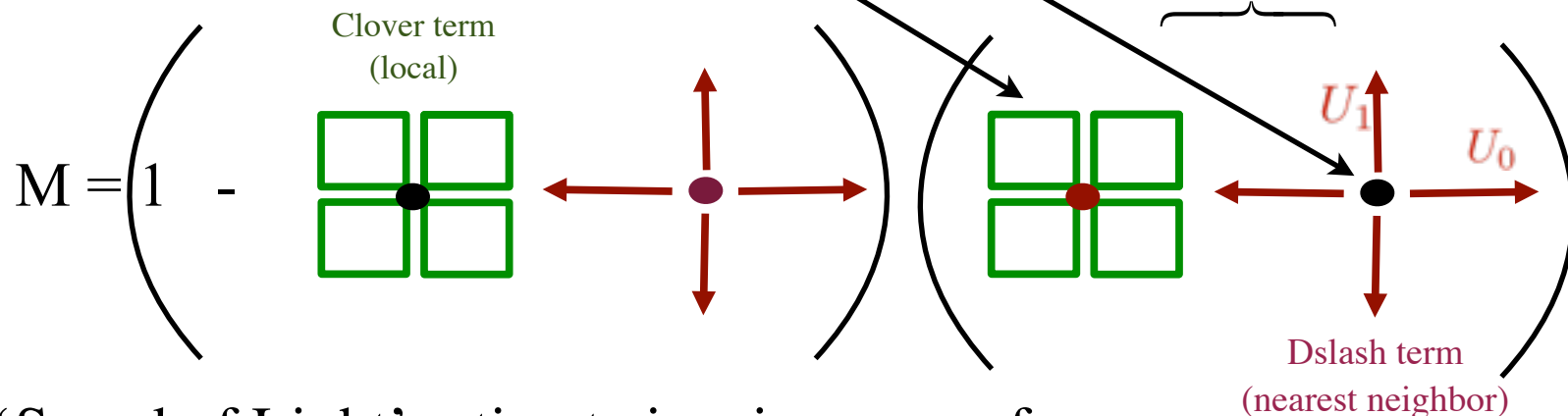
- A group of interested developers coalesced around QUDA
  - Mike Clark (Harvard), Ron Babich (BU) - QUDA leads
  - Bálint Joó (Jefferson Lab) - Chroma integration
  - Guochun Shi (NCSA) - Staggered Fermions, MILC integration
  - Will Detmold, Joel Giedt - previous contributors
  - Rich Brower (BU)
  - Steve Gottlieb, Justin Foley (U. Indiana)
- Source Code: <http://github.com/lattice/quda>

# The Wilson-Clover Fermion Matrix

After even-odd (red-black) preconditioning (Schur style):

$$M = 1 - A_{oo}^{-1} D_{oe} A_{ee}^{-1} D_{eo}$$


total: 1824 flops,  
408 words in + 24 words out  
FLOP/Byte: 1.06 (SP), 0.53 (DP)



- ‘Speed of Light’ estimate is minimum of:
  - multiply/add imbalance: ~75% of peak Flops (Dslash)
  - bandwidth constraint: ~ 1x Mem B/W in Flops (SP) 0.5x (DP)
  - staggered is harder: ~(2/3) x Mem B/W in Flops (SP) 1/3x (DP)

# B/W Trick: Compression

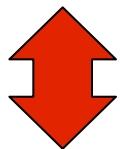
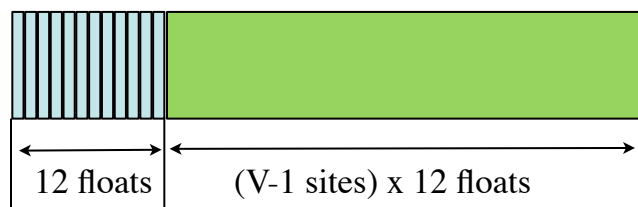
- Store 3x3 SU(3) matrix as 6 complex numbers, or 8 reals
  - 6 (complex) number storage saves loading:  $8 \times 3 \times 2 = 48$  words
  - costs  $8 \times 60 = 480$  flops.
  - New FLOP/Byte ratio for clover: 1.5 (SP), 0.75 (DP)
  - Speed of light now 50% higher from Bandwidth Constraint
  - May not realize all of this (e.g. if Flops are not really free)

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ \mathbf{x} & \mathbf{x} & \mathbf{x} \end{pmatrix} \quad \begin{array}{l} \mathbf{a} = (a_1, a_2, a_3) \\ \mathbf{b} = (b_1, b_2, b_3) \\ \mathbf{c} = (\mathbf{a} \times \mathbf{b})^* \end{array} \quad \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}$$


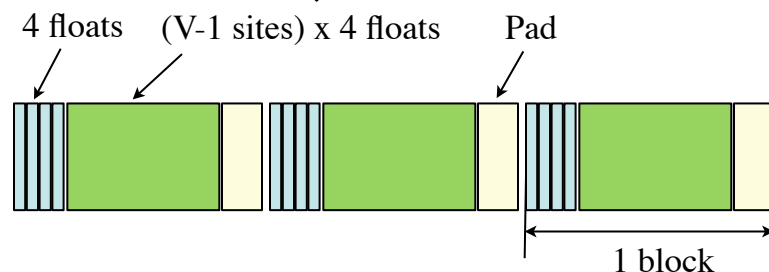
# QUDA Optimizations

- Data Layout tuned for Memory Coalescing
  - 1 thread / lattice site,
  - break up data for site data into chunks (e.g. float4 for SP)

*Host Order:*



*GPU Order:*



## Single Precision Gauge Field Example

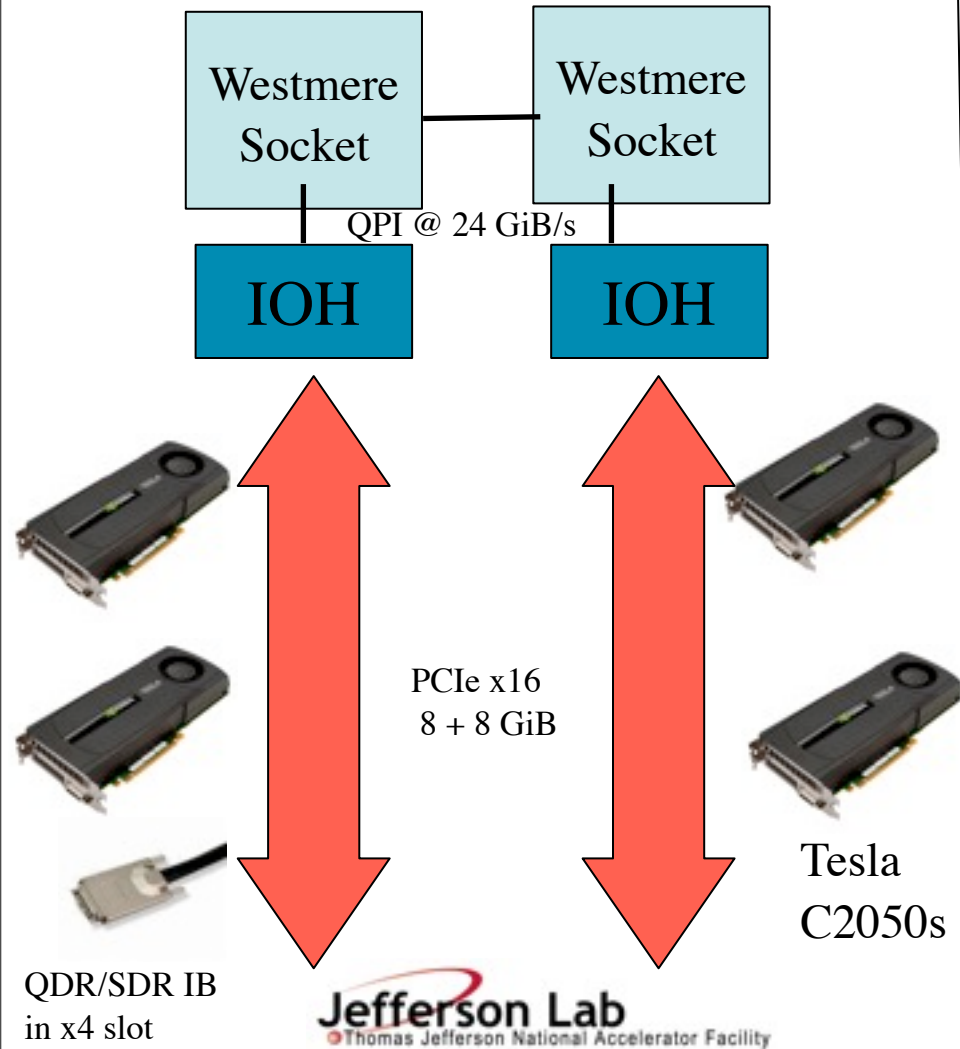
- V sites x 12 floats/site ( 2 row compressed )
- Break 12 into 3 chunks of 4 floats (float4-s)
- 1 block = V float4-s, 3 blocks for full field
- each thread reads a float4 at a time
  - coalesced reads
- Add Pad to avoid 'partition camping'
- Store ghost zones in Pad
- for spinors store ghost zones at end of data.
- similar for other types

# More bandwidth saving tricks

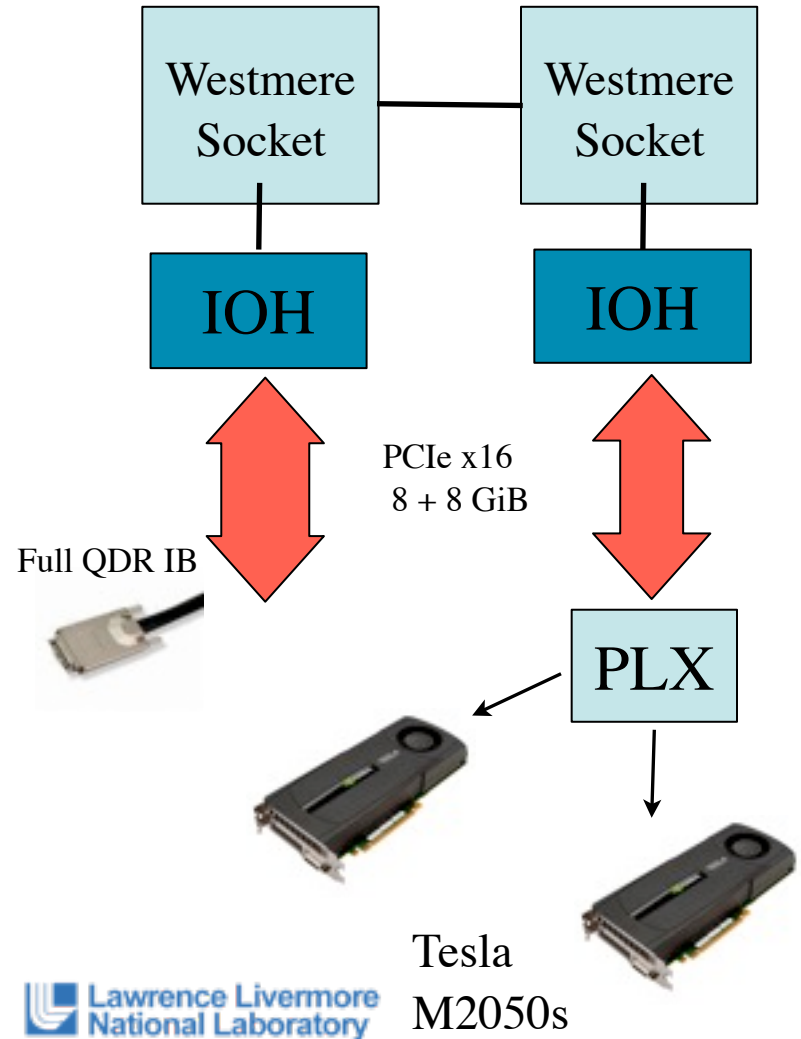
- Use Dirac Basis - only load in 1/2 of temporal neighbours
- Use Axial Gauge - save loading temporal gauge links
  - but caveat: axial gauge transformation may introduce noticeable rounding effects in single precision
- Multiple precision solver algorithms: as low as 16-bit precision
  - Iterative refinement: Inner-Outer schemes
  - Reliable Updates (aka residual replacement) schemes
    - \* solve in reduced precision
    - \* occasionally compute full precision residuum
    - \* ‘group-wise’ update of solution
    - \* equivalent to an iterative refinement scheme in mixed precision
- *Clark, et. al., Comp. Phys. Commun. 181:1517-1528, 2010*

# Test Clusters

## JLab Nodes (Up to 32 in partition)



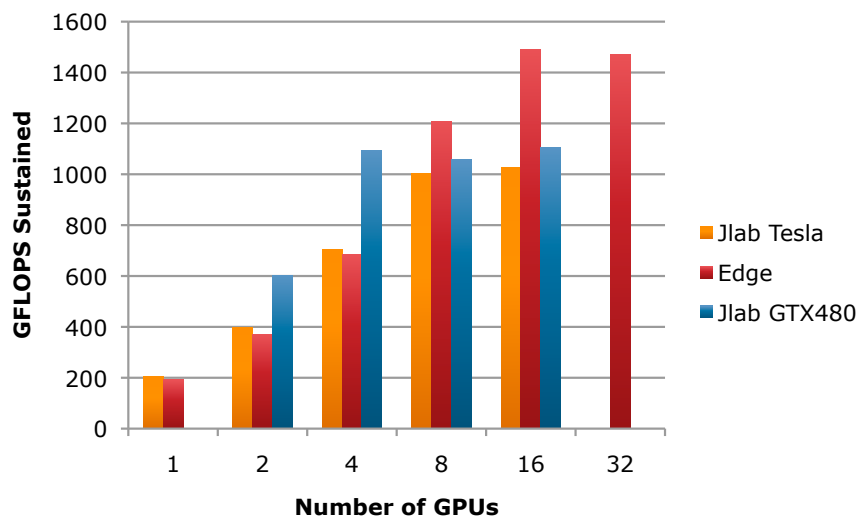
## Edge Nodes (Up to 392 in partition)





# Scaling of BiCGStab Solver

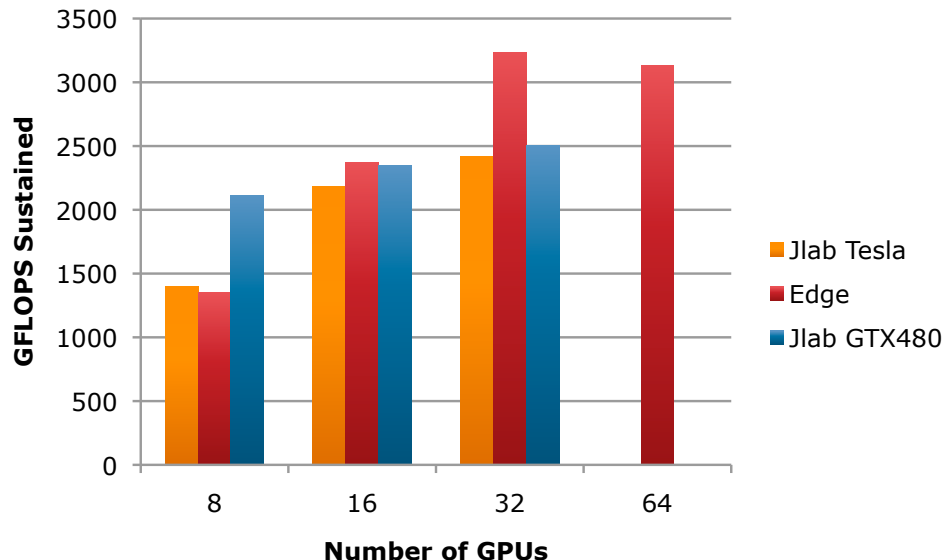
**Single-Half: 24x24x24x128**



- JLab Tesla tops out at 8 GPUs
- JLab GTX480 tops out at 4 GPUs
  - GTX 480 cluster uses SDR
  - JLab Tesla: QDR in x4 slots

- JLab Tesla tops out at 16 GPUs
- GTX480 tops out at 8 GPUs
- Edge can almost make it to 32.

**Single Half: 32x32x32x256**



# Distillation in a nutshell

Meson 2pt function:

*Peardon et al: Phys.Rev.D80:054506,2009*

$$C(t', t) = \text{Tr} [\Phi^B(t') \tau(t', t) \Phi^A(t) \tau(t, t')]$$

Meson Source/Sink  
Operator

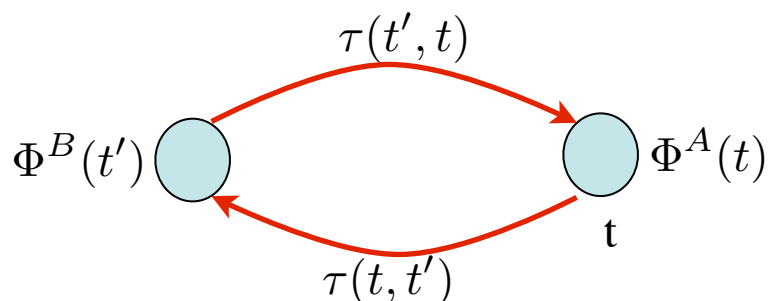
Quark 'perambulator'

$$\tau(t', t) = V^\dagger(t') M_{\alpha, \beta}^{-1}(t', t) V(t)$$

Distillation subspace

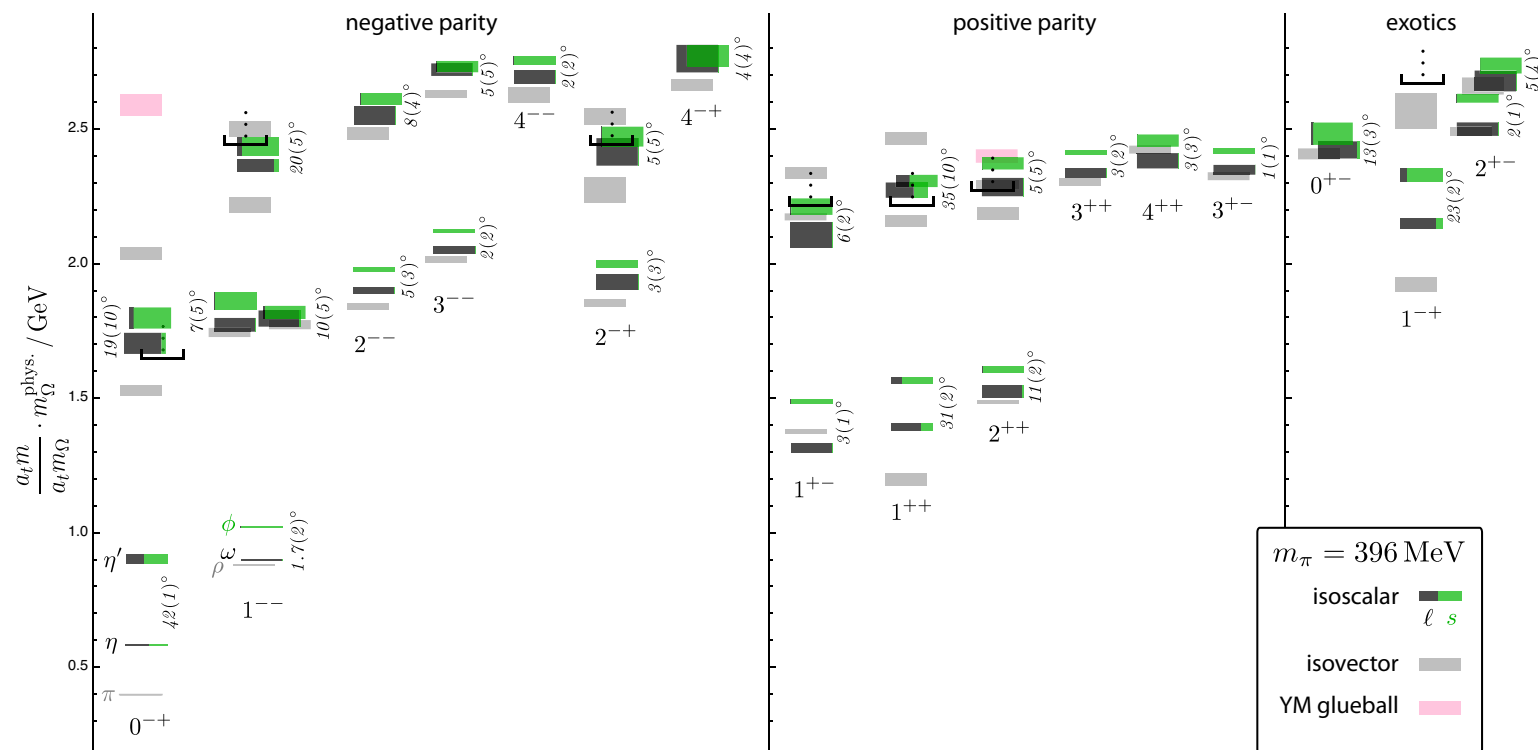
$$V(t) = (v_0 | v_1 | \dots | v_n)$$

N eigenvectors of 3D  $\nabla^2$  smearing  
operator on t-slice t (spin-diagonal)



- Both  $\Phi(t)$  and  $\tau(t', t)$  are  $N \times N_{\text{spin}}$  dimensional dense matrices
- isoscalar disconnected diagram needs  $\tau(t, t)$  on all timeslices
- $N_t \times N_s \times N_{\text{ev}} \times \text{\#quark inversions per cfg.}$  needed for isoscalar mesons

# Isoscalar Meson Spectrum



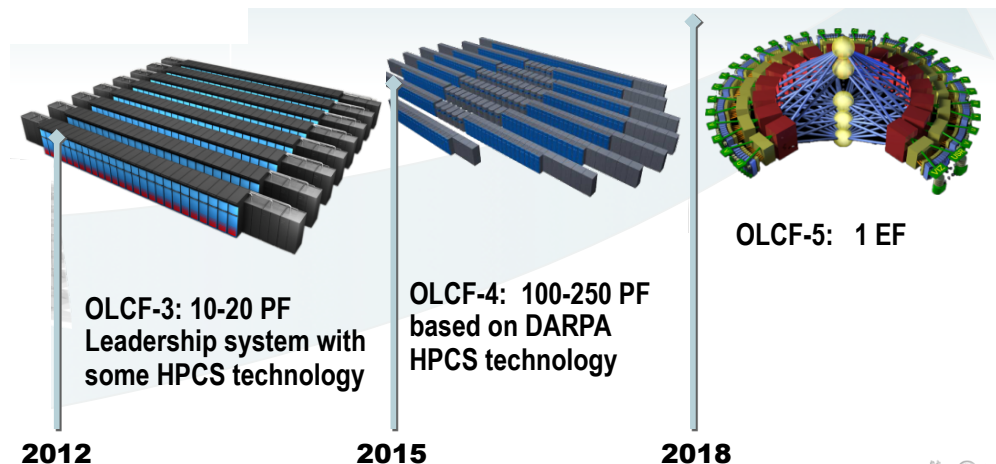
- Dudek et. al. PRD, 83, 111502(R) (2011)
- 31 Million solves + large variational basis + anisotropic lattice
- All T to all T ‘perambulators’ using Distillation method
- Excited States,  $J^{PC}$  identification, light/strange quark content
- Exotics within reach of JLab@12 GeV

# More to Isoscalars than GPUs

- Excited States: anisotropic lattice generation needed
- Disconnected diagrams: needed Distillation, a new analysis method
- $J^{\text{PC}}$  identification
  - Large variational basis of operators
  - Continuum operators subduced onto lattice cubic symmetries - new technique
- Databases of operators/perambulators
  - FILEDB package - new software
  - Lustre filesystem - new infrastructure at JLab
- and the GPUs of course :)
  - which may be able to help further with the dense matrix multiplies needed for constructing the correlation function

# What about Capability Computing?

- Accelerated Capability Machines are on the Way
  - More Clusters like Edge
  - Cray XK6 System
  - Keeneland Phase 2
  - Titan@OLCF
  - Nvidia's Echelon?
- What about Capability GPU Capability?

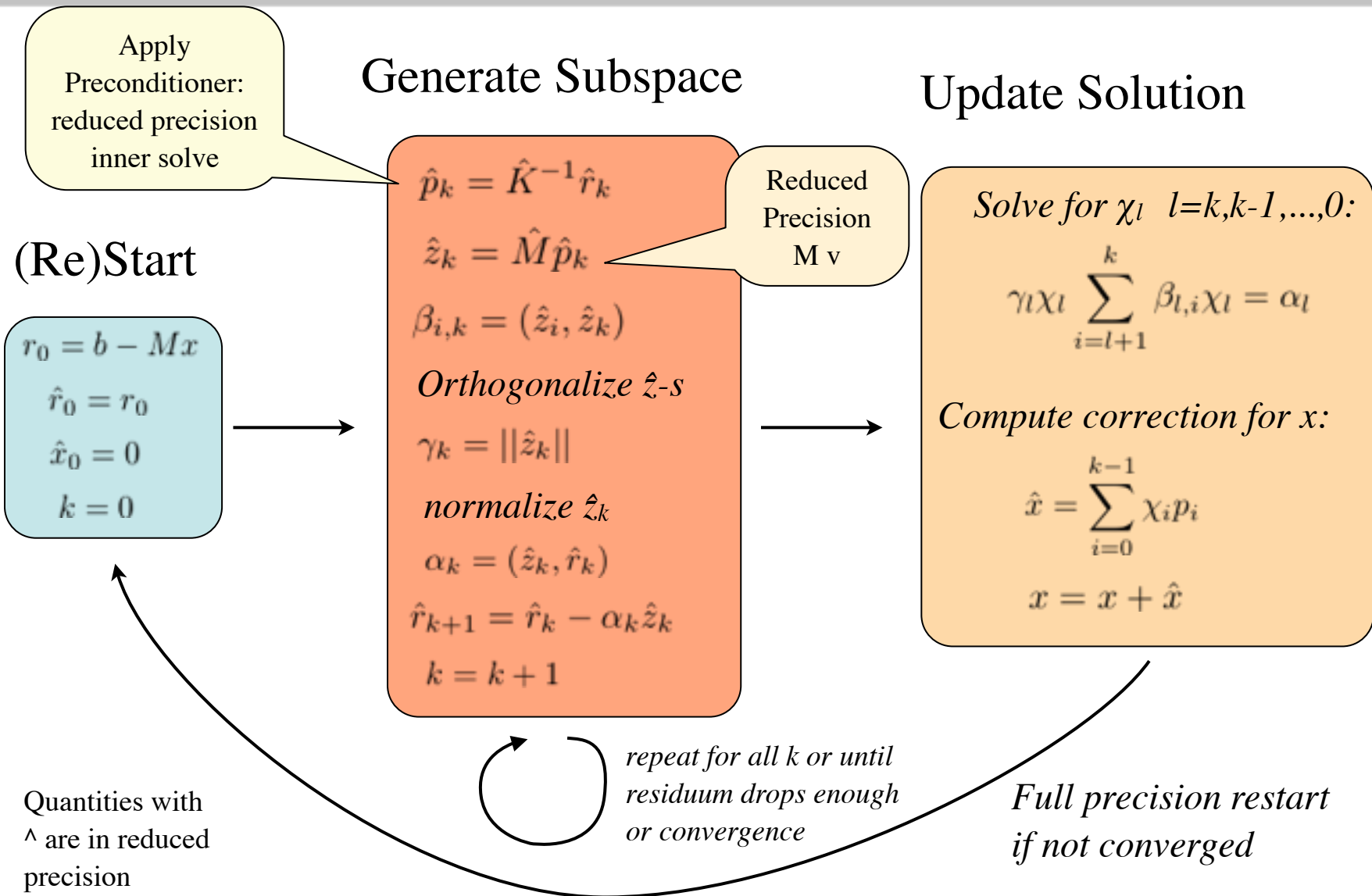


# What about Capability?

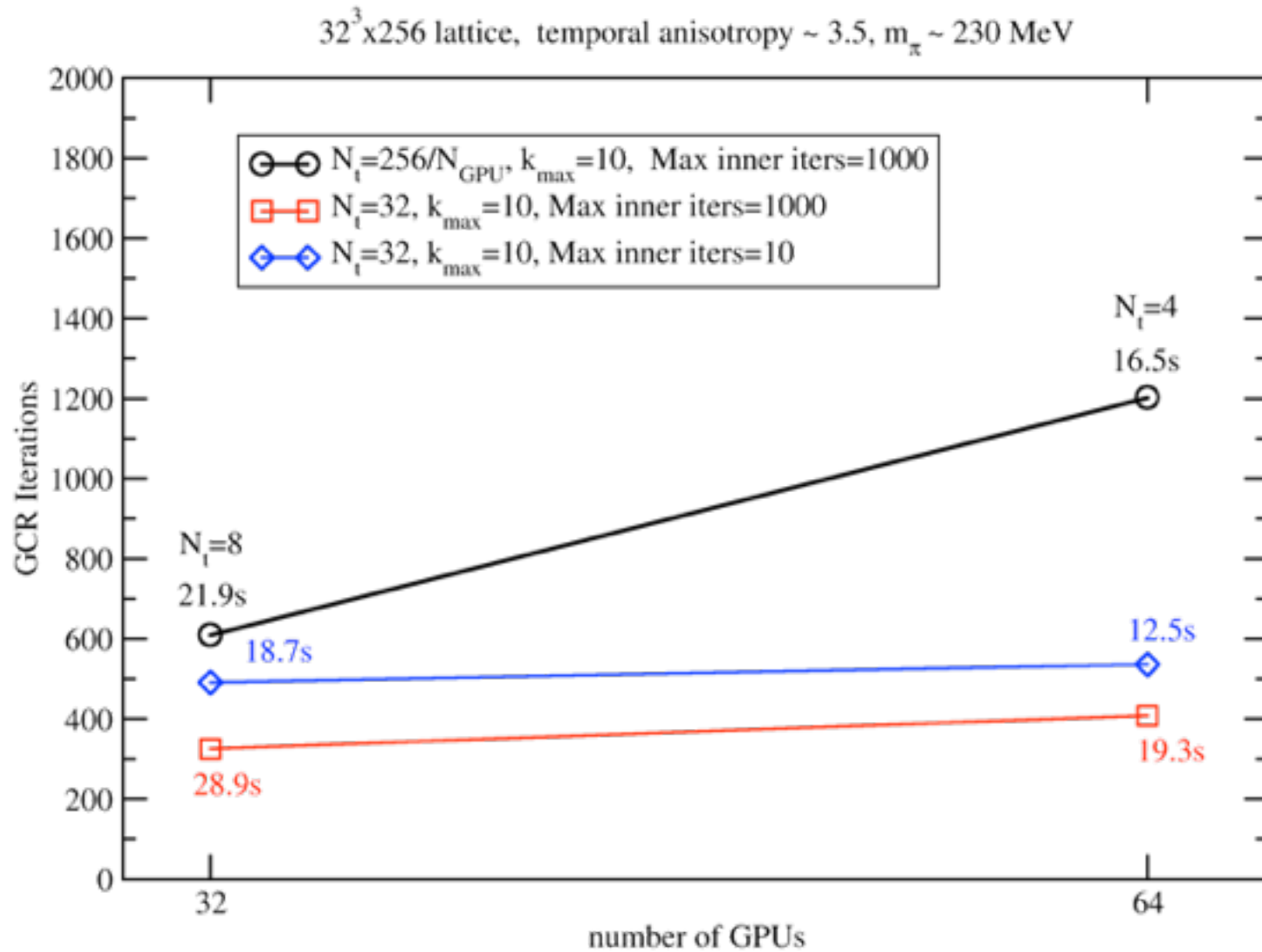
- Communication appears to be scaling bottleneck
- How to ameliorate this:
  - Wait for technology to improve
  - Change Algorithm, do less communication
- Domain Decomposed Preconditioner:
  - Divide lattice into domains, assign 1 domain to 1 GPU
  - No communication between domains (interior kernel only)
  - Apply preconditioner with ‘inner solve’
  - Need a ‘flexible’ solver (variable preconditioner) e.g. GCR, Flexible GMRES etc.



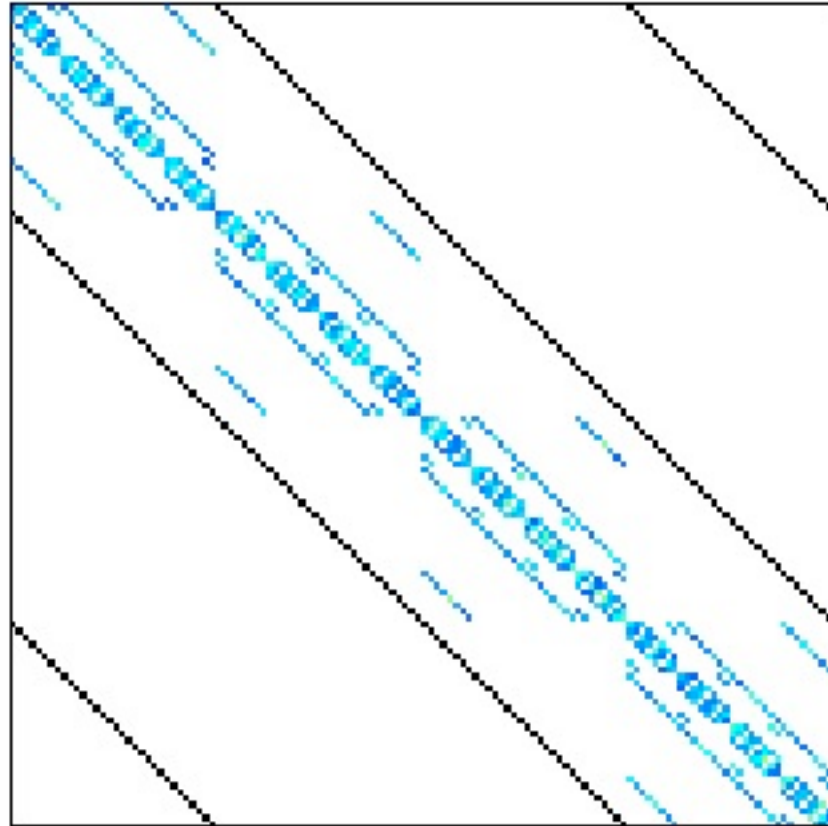
# Preconditioned GCR Algorithm



# Size Matters



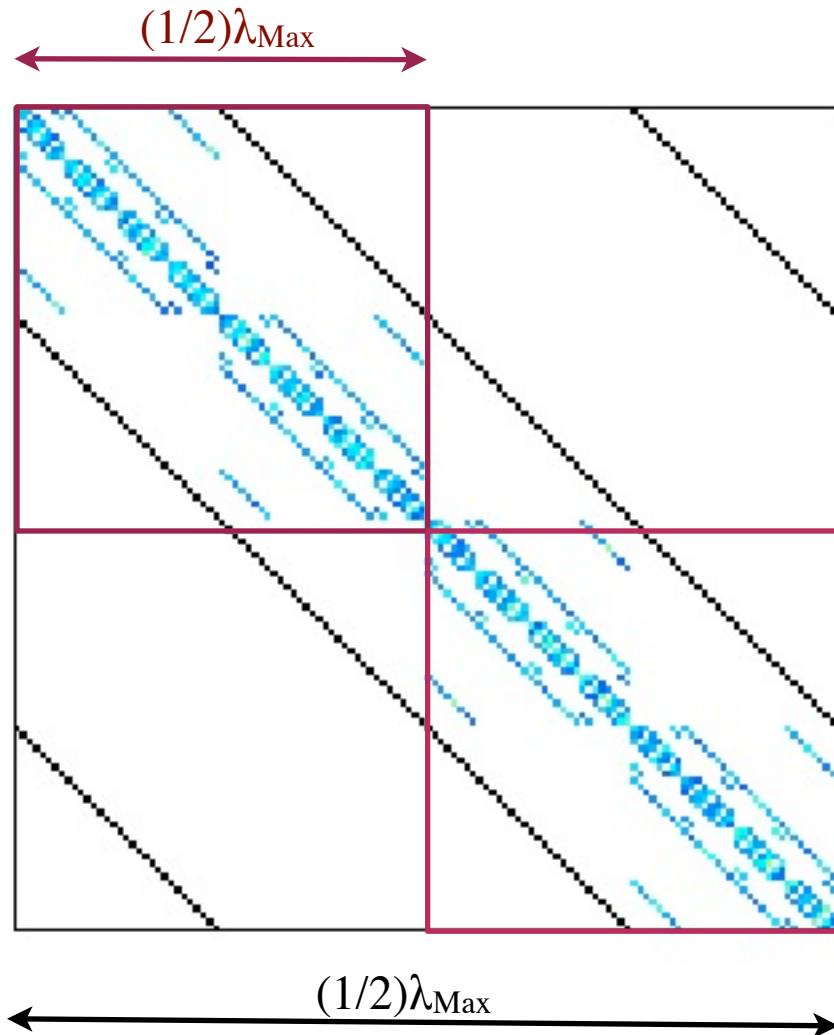
# Size Matters



$(1/2)\lambda_{\text{Max}}$

- No comms between domains
  - Block Diagonal Preconditioner
- Blocks impose  $\lambda$  cutoff
- Finer Blocks
  - lose structure in operator
  - lose long wavelength/low energy modes
- Heuristically (& from Lüscher)
  - keep wavelengths of  $\sim O(\Lambda_{\text{QCD}}^{-1})$
  - $\Lambda_{\text{QCD}}^{-1} \sim 1\text{fm}$
  - Aniso: ( $a_s=0.125\text{fm}$ ,  $a_t=0.035\text{fm}$ )
    - Our case:  $8^3 \times 32$  blocks are ideal
  - Iso:  $1\text{fm} \sim 8\text{-}10$  sites ( $a=0.11\text{fm}$ )
  - Min. blocksize has scaling implications

# Size Matters



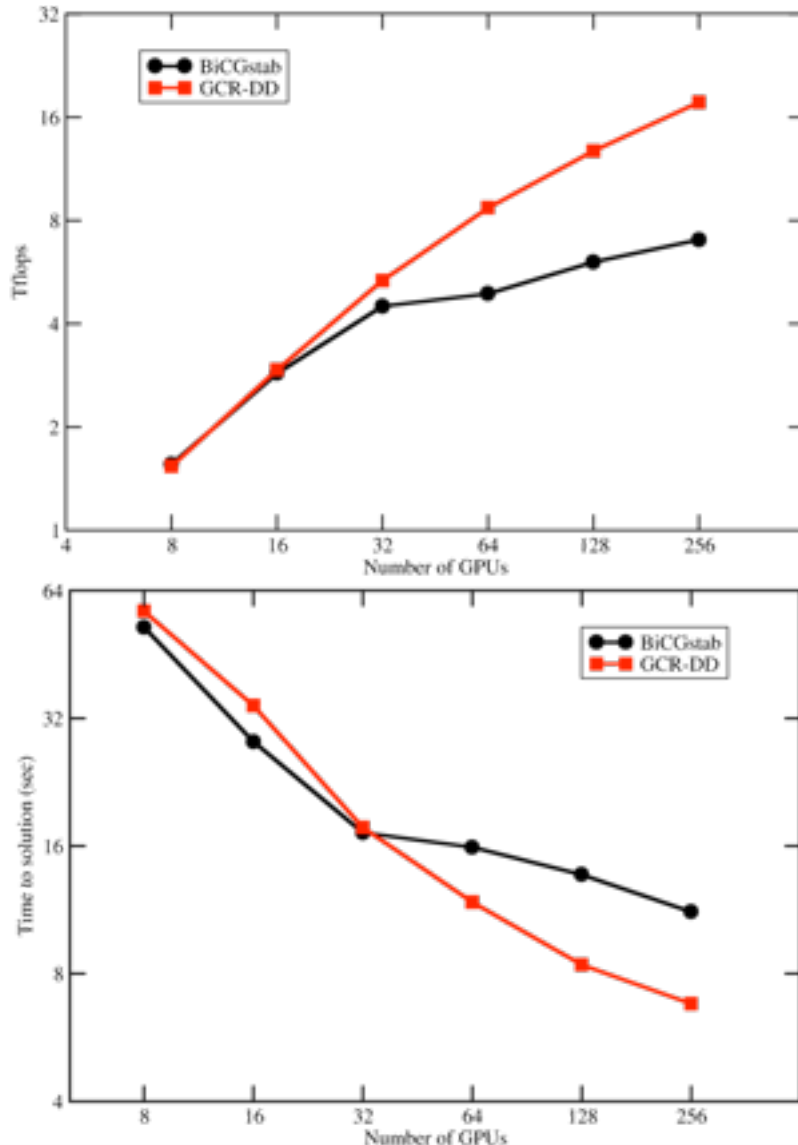
- No comms between domains
  - Block Diagonal Preconditioner
- Blocks impose  $\lambda$  cutoff
- Finer Blocks
  - lose structure in operator
  - lose long wavelength/low energy modes
- Heuristically (& from Lüscher)
  - keep wavelengths of  $\sim O(\Lambda_{\text{QCD}}^{-1})$
  - $\Lambda_{\text{QCD}}^{-1} \sim 1\text{fm}$
  - Aniso: ( $a_s=0.125\text{fm}$ ,  $a_t=0.035\text{fm}$ )
    - Our case:  $8^3 \times 32$  blocks are ideal
  - Iso:  $1\text{fm} \sim 8\text{-}10$  sites ( $a=0.11\text{fm}$ )
  - Min. blocksize has scaling implications

# Size Matters



- No comms between domains
  - Block Diagonal Preconditioner
- Blocks impose  $\lambda$  cutoff
- Finer Blocks
  - lose structure in operator
  - lose long wavelength/low energy modes
- Heuristically (& from Lüscher)
  - keep wavelengths of  $\sim O(\Lambda_{\text{QCD}}^{-1})$
  - $\Lambda_{\text{QCD}}^{-1} \sim 1\text{fm}$
  - Aniso: ( $a_s=0.125\text{fm}$ ,  $a_t=0.035\text{fm}$ )
    - Our case:  $8^3 \times 32$  blocks are ideal
  - Iso:  $1\text{fm} \sim 8\text{-}10$  sites ( $a=0.11\text{fm}$ )
  - Min. blocksize has scaling implications

# Strong Scaling of DD-GCR



- With DD-GCR, we can scale up to 256 GPUs on  $32^3 \times 256$  lattice.
  - $8^3 \times 32$  blocks: 512 GPUs max
- $> 2x$  more FLOPS v.s. BiCGStab
  - but only 1.64x faster walltime
  - trade off fast inner/slow outer iterations
- Scaling drops off at 256 GPUs
  - outer solver + reductions (?)
- This is just the **first step**: need more research on ‘architecture aware’ algorithms

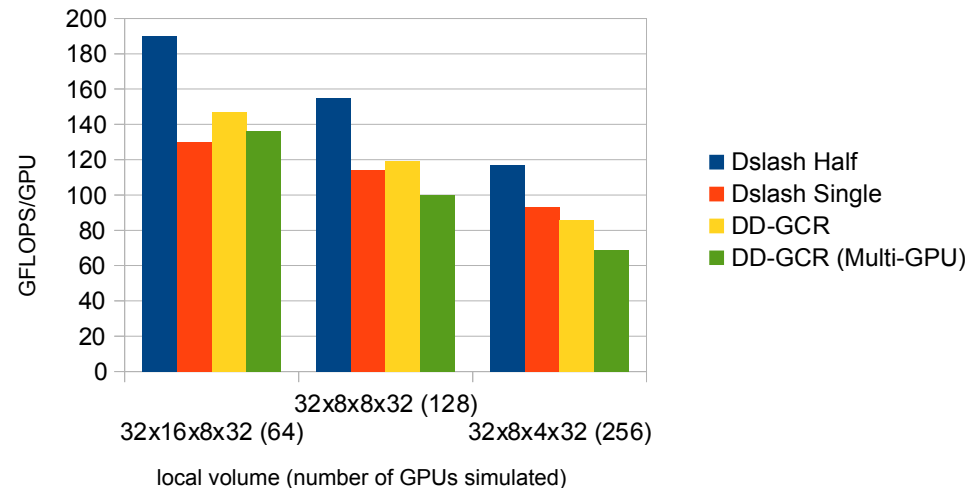
*Babich, Clark, Joó, Shi, Brower, Gottlieb, accepted for SC'11*



# “Please sir, can we have some more?”

- 17.2 Tflop on 256 GPUs = 69 Gflops/GPU
  - using all the precision tricks
  - The local problem is small
    - Low occupancy?
    - Driver overheads?
    - Communications?
  - Single GPU runs
    - with ‘strong scaling’ local volumes
  - Communications seems not the worst bottleneck here.
- Current Multi-GPU sweet spot: 128 GPUs ~ 100Gflop/GPU.
- Ambition:  $40^3 \times 256$  lattice, 1000 GPUs, 50-100 Tflops(?)
- Large algorithmic space to explore

Single GPU using the local volumes from the Multi-GPU running



# QUDA Publications

- Barros, Babich, Brower, Clark, Rebbi, “Blasting through lattice calculations using CUDA”, PoS LATTICE2008:045, 2008
- Clark, Babich, K. Barros, R. C. Brower, C. Rebbi, “Solving Lattice QCD systems of equations using mixed precision solvers”, Comput. Phys. Commun, 181:1517-1528, 2010
- Babich, Clark, Joó, “Parallelizing the QUDA Library for Multi-GPU Calculations in Quantum Chromodynamics”, IEEE/ACM International Conference for High Performance Computing, Networking Storage and Analysis, SC’10, New Orleans, Louisiana, 2010
- G. Shi, S. Gottlieb, A. Torok, V. Kindratenko, “Design of MILC lattice QCD application for GPU clusters,” 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Anchorage(Alaska), May, 2011
- S. Gottlieb, G. Shi, A. Torok, V. Kindratenko, “Quda programming for staggered quarks” In Proc. XXVIII International Symposium on Lattice Field Theory (Lattice 2010), Villasimius, Sardinia, June 2010
- B. Joó, R. Babich, R. Brower, M. Clark, J. Chen, J. Dudek, R. Edwards, M. Peardon, C. Rebbi, D. Richards, G. Shi, C. Thomas, W. Watson III, ”Hadronic Physics from Lattice QCD and GPUs” Scientific Discovery through Advanced Computing (SciDAC), 2010
- G. Shi, S. Gottlieb, A. Torok, V. Kindratenko, “Accelerating Quantum Chromodynamics Calculations with GPUs”, 2010 Symposium on Application Accelerators in High-Performance Computing (SAHPC10)
- Babich, Clark, Joó, Shi, Brower, Gottlieb, “Scaling Lattice QCD Beyond 100 GPUs”, Accepted for Publication in IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis, SC’11, Seattle, 2011

# Related Algorithmic Work

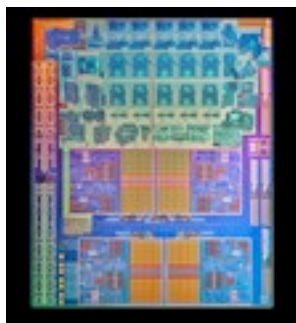
- Schwarz preconditioner
  - (SAP+GCR) Lüscher, Comput.Phys.Commun. 156(2004) 209-220
  - (RAS+ flex. BiCGStab) Osaki, Ishikawa, PoS(Lattice2010), 036
- Domain Decomposed HMC
  - Lüscher, JHEP 0305 (2003) 052
  - Lüscher, Comput.Phys.Commun.165:199-220,2005
- Multi-Grid:
  - Babich et. al., Phys.Rev.Lett.105:201602,2010
  - Osborn et. al., PoS Lattice2010:037,2010
- Deflation:
  - Lüscher, JHEP 0707:081,2007, JHEP 0712:011,2007
  - Stathopoulos & Orginos: SIAM J. Sci. Comput. 32, pp. 439-462

Challenge:  
Updating  
preconditioner/  
deflation space  
in the Gauge  
evolution.

# Will GPUs survive to the exascale?

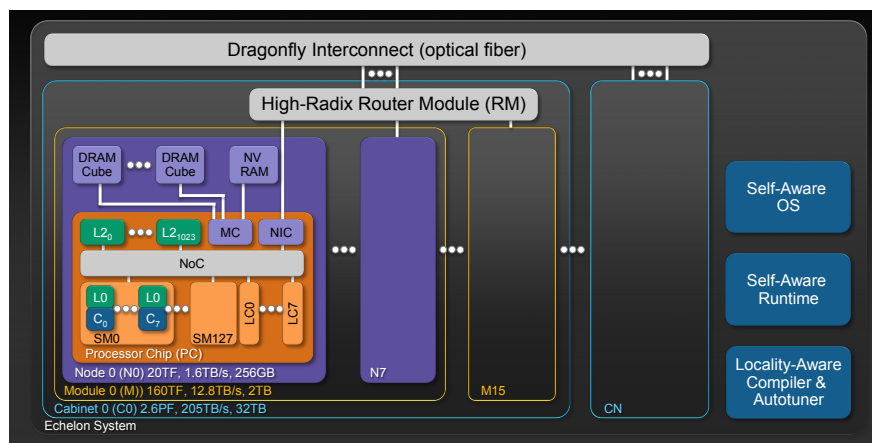
- Perhaps not in the form of ‘PCIe bus based accelerators’
- But very likely as part of massively parallel nodes

Hybrid APU  
for Desktop today



AMD Llano A8 fusion APU:  
4 CPUs + GPU on chip  
figure from [anandtech.com](http://anandtech.com)

Hybrid System for the Exascale



NVIDIA Echelon System Sketch, 20 Tflop node made up of 128 SMs  
figure from [Bill Dally's SC'11 Keynote](#)

# Where do we go from here?

- Forthcoming ‘large’ (sustained petascale) machines in the US
  - Titan: GPU accelerated (eventually)
    - CUDA, OpenCL, Compilers
  - BlueWaters: Petascale multi-core, shared caches, vectors
    - Plain old C/C++, OpenMP/pthreads
  - BlueGene/Q
- Accelerator architectures
  - NVIDIA CUDA line (Kepler, Maxwell)
  - AMD line (OpenCL, Microsoft C++ AMP extensions?)
  - Intel MIC
- Technology is in a state of flux - many programming model options

# What about the exascale?

- It's all very well to say:
  - “Expose parallelism in your problem.”
  - “You'll reap benefits in the long term (exascale and beyond)”
- But we also need to reap benefits in the short run to survive:
  - High Performance code for BlueWaters, BlueGene, AVX
  - Moving gauge generation, correlation functions onto GPUs
  - Developing new algorithms: more Domain Decomposition?
  - Developing new analysis techniques: better scaling Distillation?
  - Tuning current production run parameters
- We should think also about ‘planting trees’ (a la Brad Chamberlain)
  - OpenCL? LQCD in Chapel ?
- and of course maintain Chroma, develop QUDA, and do physics too

# A personal worry...

- Doing the kind of work I just mentioned is resource intensive
  - Bronson Messer's talk: 2 +/- 0.5 FTEs to port codes to GPUs
    - in practice: ~4 member code teams @ ~50% effort
    - including 2 vendor specialists
- Our current resources are oversubscribed
- Risk Management:
  - How would your project cope without its current lead developer?
- Need for resources that can be focused on the software tasks
  - Grad Students typically focus on post graduation jobs
  - Post Docs usually focus on doing science to get long term jobs.
- This issue needs to be addressed soon
  - for both exascale and current generation hardware

# Summary/Conclusions

- GPUs are with us, and provide much needed Flops for QCD.
- Common features between GPU and future Exascale computers
- Successful exploitation by Lattice QCD
  - QUDA Library in the US: strong scaled to 256 GPUs
  - Capacity Use: JLab clusters deliver ~100TFlop/s in aggregate
  - Isoscalar Spectrum calculation made tractable by GPUs
- Need to get the entire LQCD software stack GPU enabled
  - already done by e.g.: Budapest-Wuppertal Group, Pisa group, ...
- Need to get more resources for this here in the U.S.
  - algorithm development
  - code development
  - collaboration



# Acknowledgements

- QUDA Collaborators:
  - Mike Clark, Ron Babich, Guochun Shi, Steve Gottlieb, Rich Brower
- Thanks to Jefferson Lab and LLNL for cluster use.
- B. Joó acknowledges funding through US DOE grants
  - DE-FC02-06ER41440 and DE-FC02-06ER41449 (SciDAC)
  - DE-AC05-06OR23177 under which JSA LLC operates JLab.
- M. Clark acknowledges funding through NSF grant OCI-1060067
- G. Shi acknowledges funding through the Institute for Advanced Computing Applications and Technologies (IACAT) at the University of Illinois at Urbana-Champaign.