The Fat-Link Computation On Large GPU Clusters for Lattice QCD

Guochun Shi

Innovative System Laboratory
NCSA

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

Credits

- Ron Babich (Nvidia)
- Mike Clark (Nvidia)
- Balint Joó(Jefferson Lab)
- Steve Gottlieb (Indiana University)
- Vlad Kindratenko (NCSA)
- Justin Foley (University of Utah)

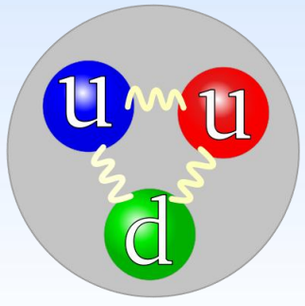
Outline

- **Application Background**
  I. Lattice QCD and MILC
  II. CPU run characteristics
  III. Fatlink computation in CPU

- **GPU Implementation with QUDA library**
  I. Data layout for CPU and GPU
  II. Multi-GPU approach
    - ❑ Standard approach
    - ❑ Extended volume approach
- **Performance**
  I. Single GPU performance with artificial communication
  II. Multi-GPU strong scaling performance

- **Conclusions**

Quantum ChromoDynamics



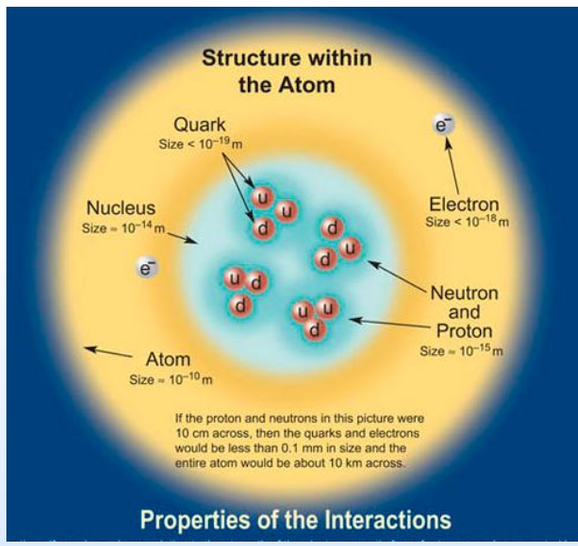
- QCD is the theory of the strong force that binds nucleons
- Impose local SU(3) symmetry on vacuum
- Color charge analogous to electric charge of QED
- Lagrangian of the theory simple to write down

$$\mathcal{L}_{QCD} = \psi_i \left(i\gamma^\mu (D_\mu)_{ij} - m\delta_{ij} \right) \psi_j - G^a_{\mu\nu} G^{\mu\nu}_a$$

- Path integral formulation

$$\langle \Omega \rangle = \frac{1}{Z} \int [dU] e^{-\int d^4x L(U)} \Omega(U)$$

- Infinite dimensional integral
- Theory is strictly non-perturbative at low energies



Slide courtesy of M. A. Clark, Harvard University

Lattice QCD performance on CPU

Time distribution for a run on 2048 XT3 (BigBen) cpus using a $40^3 \times 96$ grid ($5 \times 10^2 \times 6$ per cpu) with $m_l = 0.1 m_s$:

Activity	time(s)	MF/cpu	per cent
CG	2987	530	58.5
FF	1125	579	22.0
GF	489	469	9.5
Fat	442	627	8.7
Long	24	340	<1
Input config.	41		<1
total above	5108		
unaccounted	104		1.9
wallclock	5212		

- Two phases in Lattice QCD computation
  - Configuration generation: compute a snapshot of state of the strong force field, Conjugate Gradient (CG), fermion force (FF), gauge force (GF) and link fattening (FAT).
  - Analysis: the observables of interest are computed over the configurations. Conjugate Gradient (CG) is even more dominant in this phase.
  - In short, CG is the dominant part, but other parts are important.

Lattice QCD performance on CPU

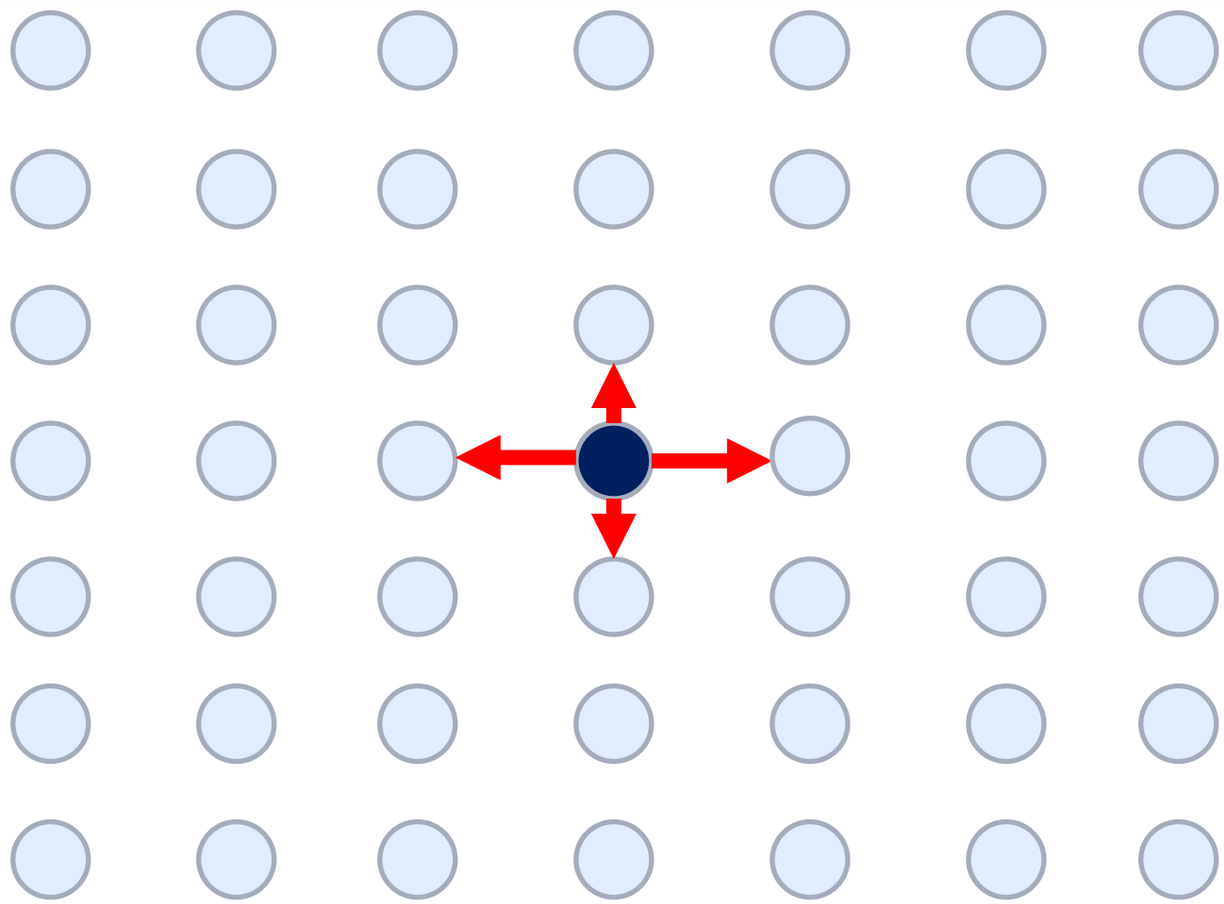
Time distribution for a run on 2048 XT3 (BigBen) cpus using a $40^3 \times 96$ grid ($5 \times 10^2 \times 6$ per cpu) with $m_l = 0.1 m_s$:

target

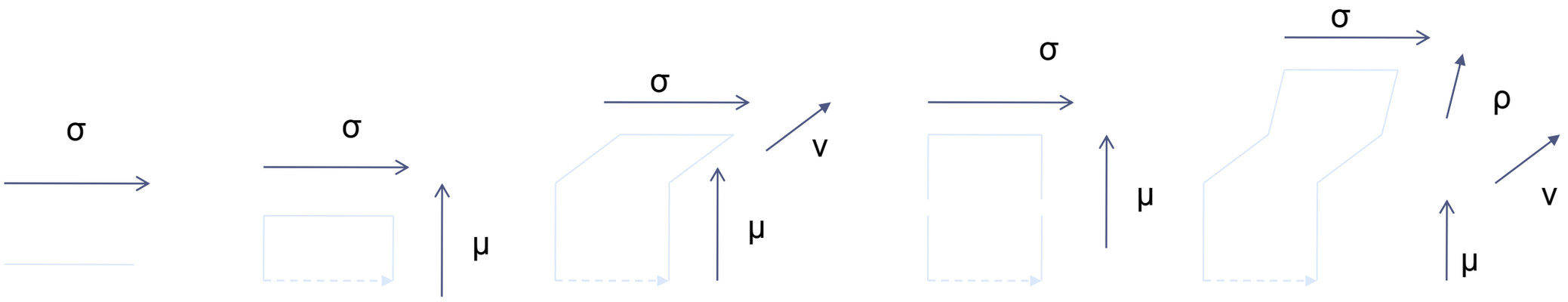
Activity	time(s)	MF/cpu	per cent
CG	2987	530	58.5
FF	1125	579	22.0
GF	489	469	9.5
Fat	442	627	8.7
Long	24	340	<1
Input config.	41		<1
total above	5108		
unaccounted	104		1.9
wallclock	5212		

- Two phases in Lattice QCD computation
  - Configuration generation: compute a snapshot of state of the strong force field, Conjugate Gradient (CG), fermion force (FF), gauge force (GF) and link fattening (FAT).
  - Analysis: the observables of interest are computed over the configurations. Conjugate Gradient (CG) is even more dominant in this phase.
  - In short, CG is the dominant part, but other parts are important.

Site links in 2-D

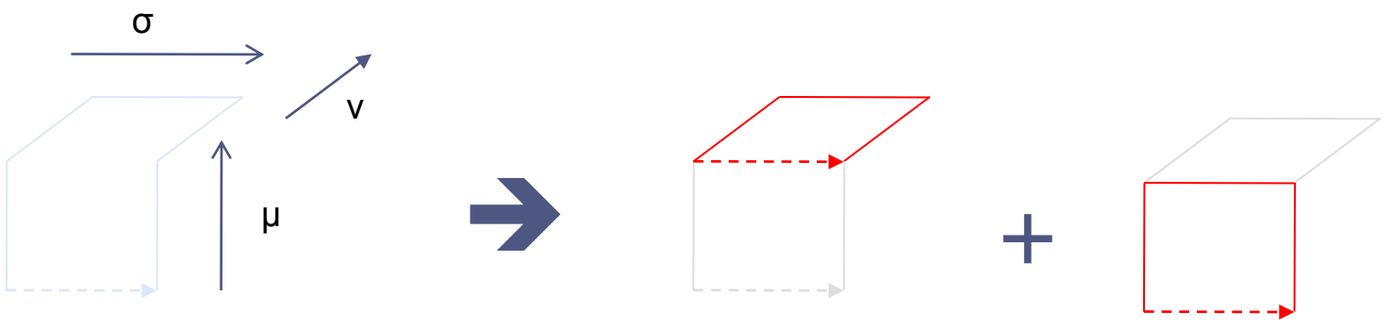


links, 3x3 complex matrix



1-link term 3-link term 5-link term Lepage term 7-link term

- Fatlink is computed using site links



5-link term

Two 3-link computations

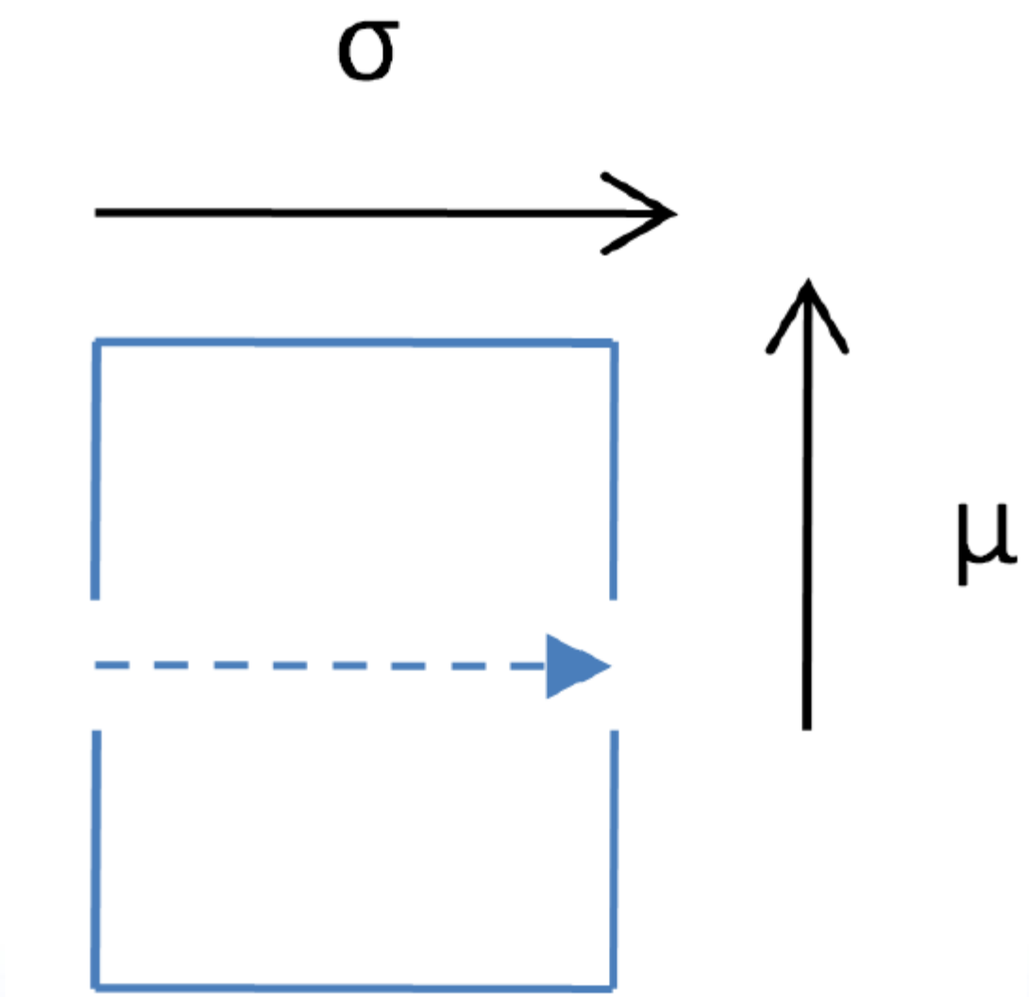
- Fat7 link fattening
- 5-link/Lepage/7-link terms are computed using multiple 3-link term computations
- This also avoid redundant computations

Fatlink CPU implementation code

```
one_link = (act_path_coeff[0] - 6.0*act_path_coeff[5]);
for (dir=XUP; dir<=TUP; dir++){
  FORALLSITES(i,s) {/*Loop through all sites macro */
     fat1 = (*t_fl) +  4*i + dir;
     scalar_mult_su3_matrix(&(s->link[dir]), one_link, fat1 );                    /* 1-link term*/
   }
  for(nu=XUP; nu<=TUP; nu++) if(nu!=dir){
     compute_gen_staple_site(staple,dir,nu,F_OFFSET(link[dir]), *t_fl, act_path_coeff[2]);   /* 3-link term*/
     compute_gen_staple_field(NULL,dir,nu,staple, *t_fl, act_path_coeff[5]);      /* Lepage term */
     for(rho=XUP; rho<=TUP; rho++) if((rho!=dir)&&(rho!=nu)){
        compute_gen_staple_field( tempmat1, dir, rho, staple, *t_fl, act_path_coeff[3]);   /* 5-link term*/
        for(sig=XUP; sig<=TUP; sig++)
          if((sig!=dir)&&(sig!=nu)&&(sig!=rho)){
            compute_gen_staple_field(NULL,dir,sig,tempmat1, *t_fl, act_path_coeff[4]);   /* 7-link term*/
          } /* sig */
       } /* rho */
    } /* nu */
}/* dir */
```

- Each *compute_gen_staple_*()* call is a 3-link term call
- Communications are coded inside the staple computing function

Compute_gen_staple_*()



QUDA library

- A generic lattice QCD library using NVIDIA GPUs written in CUDA C/C++, supporting various fermion actions.

- Work started at Boston University, now multiple institutions collaborate to develop the library

- Open Source, available at http://lattice.github.com/quda/

- The fatlink implementation uses QUDA library

Fatlink Kernels: Bandwidth Bound

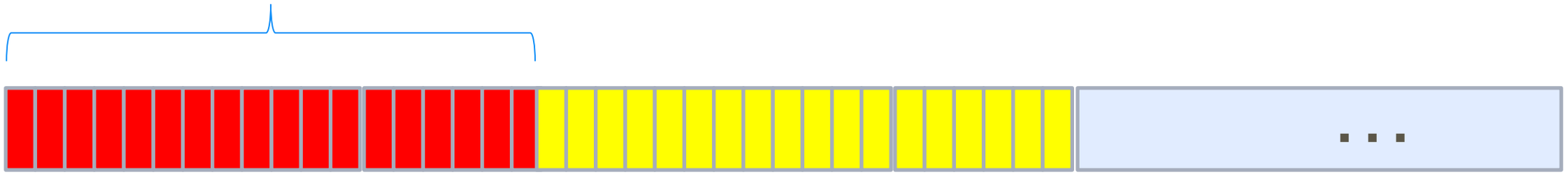
- Total bytes (single precision)
  - Each link is 3x3 complex matrix: 72 bytes
  - 7 link load and 2 link write
  - Total bytes= (7+2)*72 = 648 bytes

- Total flops
  - Matrix matrix multiplication: 198 flops
  - Matrix scaling/add/sub: 18 flops
  - 4 matrix matrix multiplication and 3 matrix scaling/add/sub
  - Total flops= 198*4+18*3 =846

- Flop/Byte Ratio
  - 1.30 (single precision)
  - 0.65 (double precision)

Gauge field layout in host memory

one gauge field



Gauge field layout in GPU memory

Vh *Float2 + pad

pad/ghost gauge fields

```
#define LOAD_MATRIX_18_SINGLE(gauge, dir, idx, var, stride)       \
  float2 var##0 = gauge[idx + dir*9*stride];                      \
  float2 var##1 = gauge[idx + dir*9*stride + stride];             \
  float2 var##2 = gauge[idx + dir*9*stride + 2*stride];           \
  float2 var##3 = gauge[idx + dir*9*stride + 3*stride];           \
  float2 var##4 = gauge[idx + dir*9*stride + 4*stride];           \
  float2 var##5 = gauge[idx + dir*9*stride + 5*stride];           \
  float2 var##6 = gauge[idx + dir*9*stride + 6*stride];           \
  float2 var##7 = gauge[idx + dir*9*stride + 7*stride];           \
  float2 var##8 = gauge[idx + dir*9*stride + 8*stride];
```

Optimization techniques summary

- Data arrangement in device memory to enable coalesced read and write
- Using textures to read data improves performance
- Use padding memory to avoid partition camping
- Use constant memory to store commonly used constant variables (X1,X2,X3,X4, X1m1, X2X1mX1, …)
- 12 or 8 reconstructing of gauge field matrix to save bandwidth in the expense of more computation

Optimization techniques summary -- cont

- Auto tuning
- Use shared memory to reduce register pressure
- Use 48 KB L1 cache (Fermi)
- Automatically find the best NUMA cores

Multi-GPU Implementation: Standard Approach

- Each *compute_gen_staple_*()* is split into multiple kernels, with data communication from exterior kernels

- Multiple exterior kernels
  - Computed boundary nodes

- One interior kernel
  - Interior nodes computed

- The idea is to overlap computation with communication

Standard Approach

Total 9 cuda Streams

0: kernels

X+ exterior kernel

Interior kernel

1: X-backward

2: X-forward

.
.
.

7: T-backward

8: T-forward

T+ exterior kernel

gather kernel

GPU kernel

cudaMemcpy

MPI send/recv

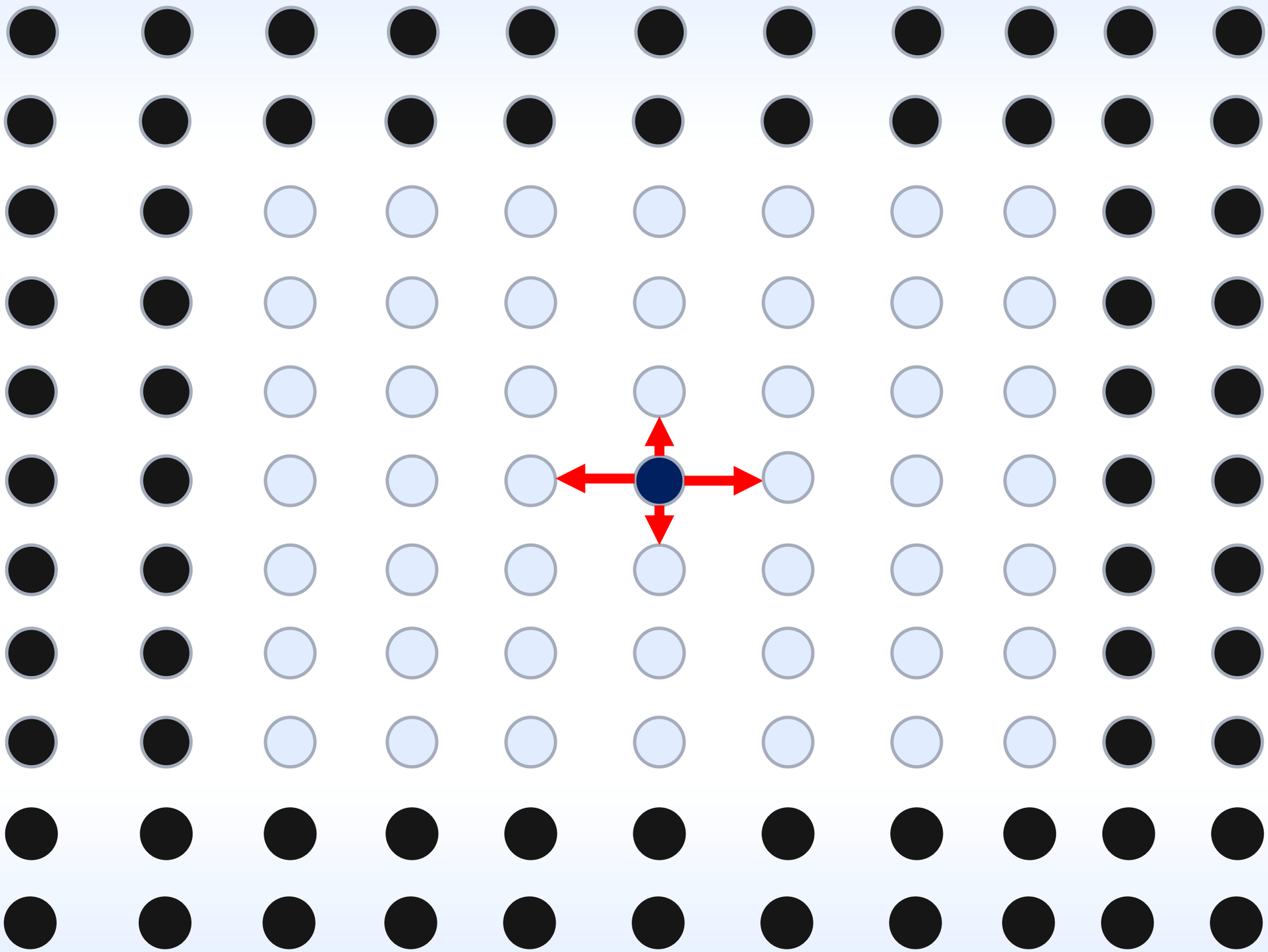
Multi-GPU Implementation: Extended Volume Approach

- We notice that all link data we need from neighbors are 2 slices of ghost data from each direction

- After that we can compute all intermediate staple data without communication

- This means more communication at the beginning of fatlink computation, redundant computation inside *compute_gen_staple_*()* calls, but there is no communication inside *compute_gen_staple_*()* calls

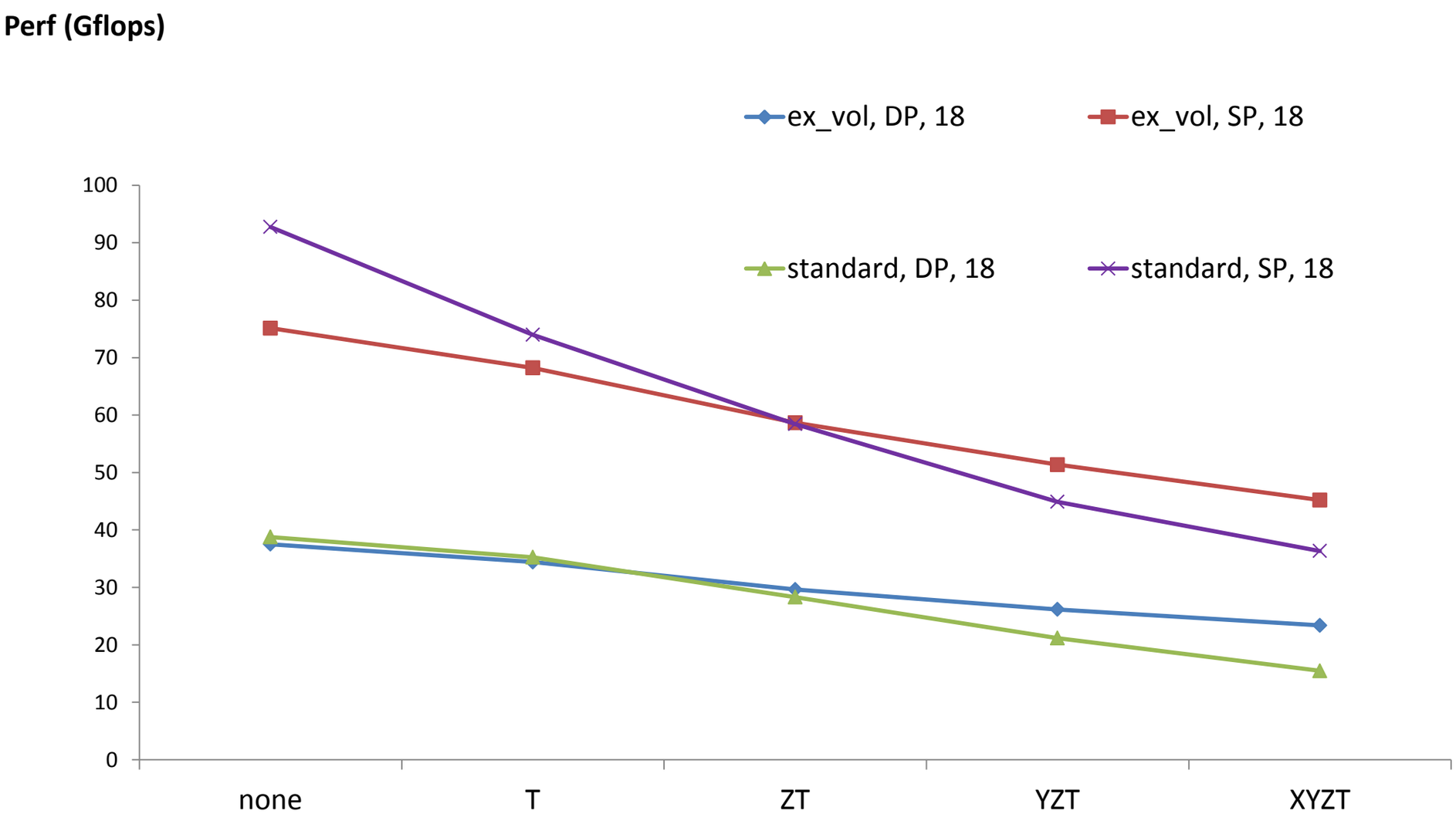
Site links in 2-D with ghost cells



links, 3x3 complex matrix

Single GPU performance running on multi-gpu code path

Performance in GTX480

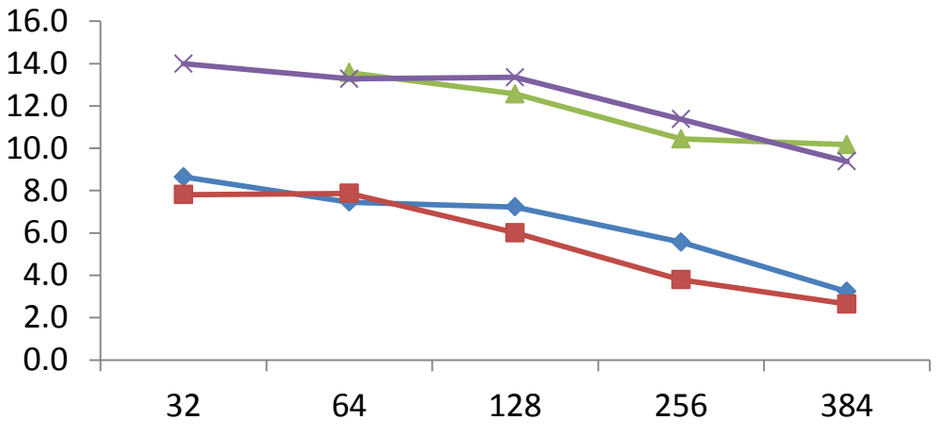


Longhorn @ TACC

- Each node
  - 8 Intel Nehalem cores @ 2.5 Ghz
  - 2 Nvidia Quadro FX5800 GPUs
  - GPU:CPU_core ratio = 1:4

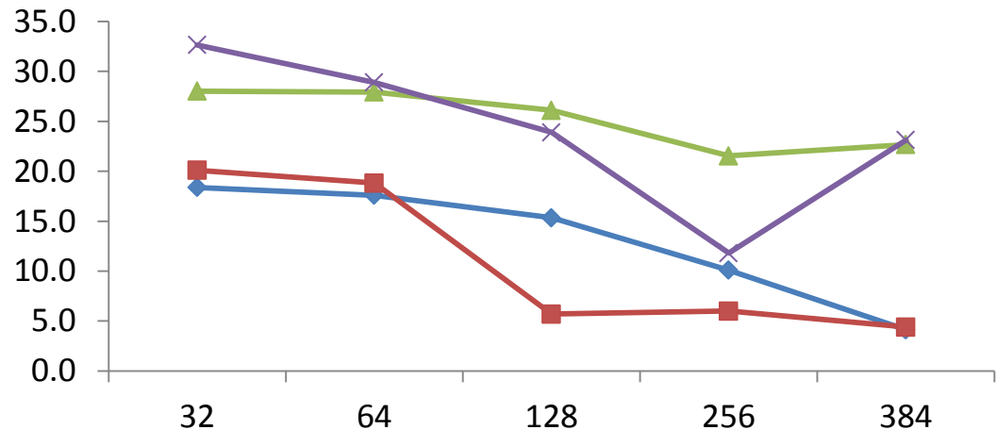
Gflops/GPU

DP_XYZT



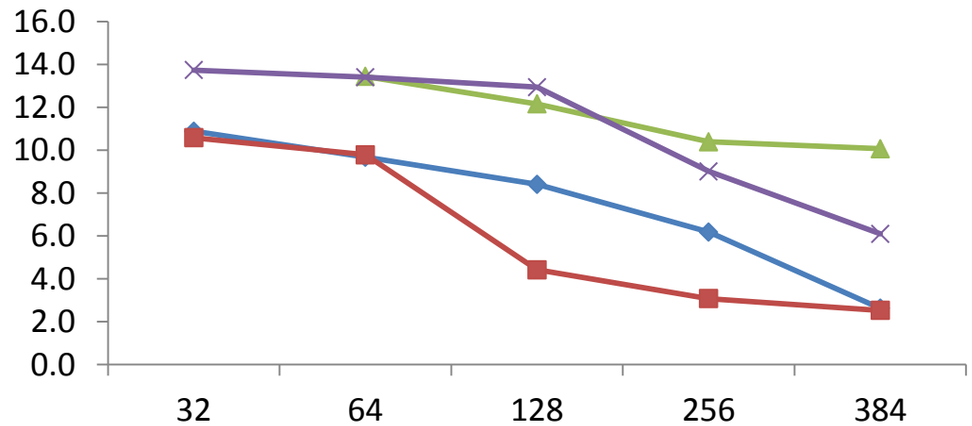
Gflops/GPU

SP_XYZT

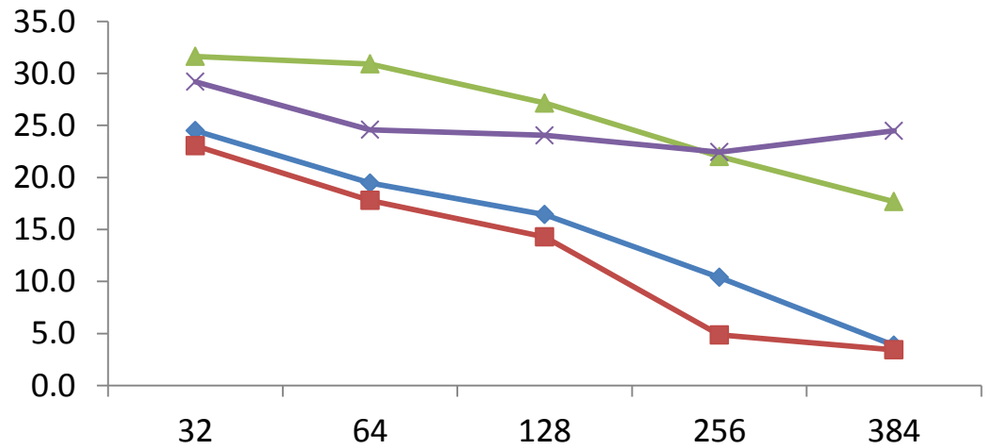


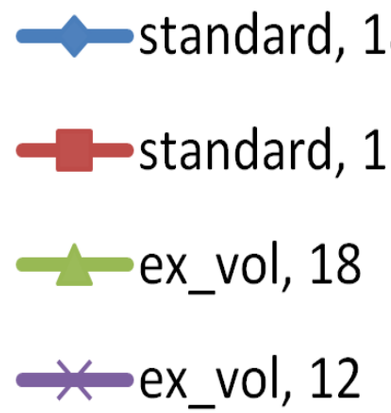
Longhorn

DP_YZT

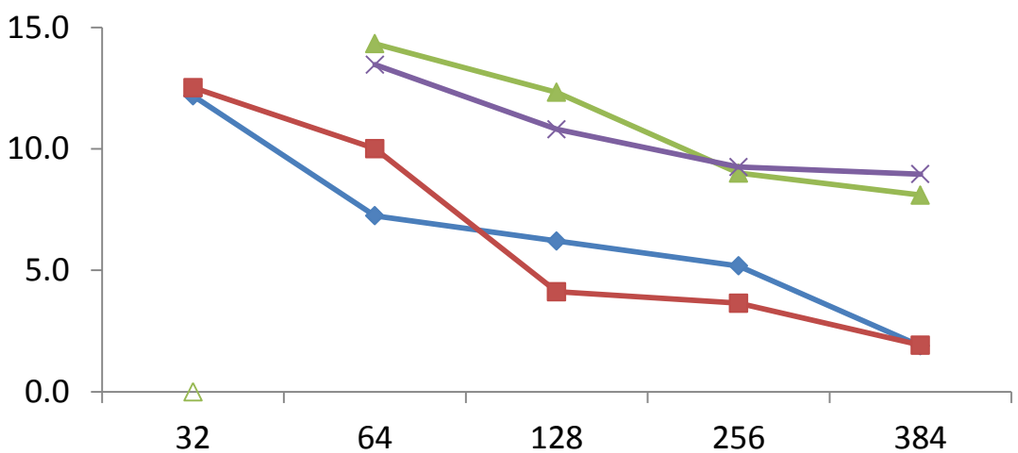


SP_YZT

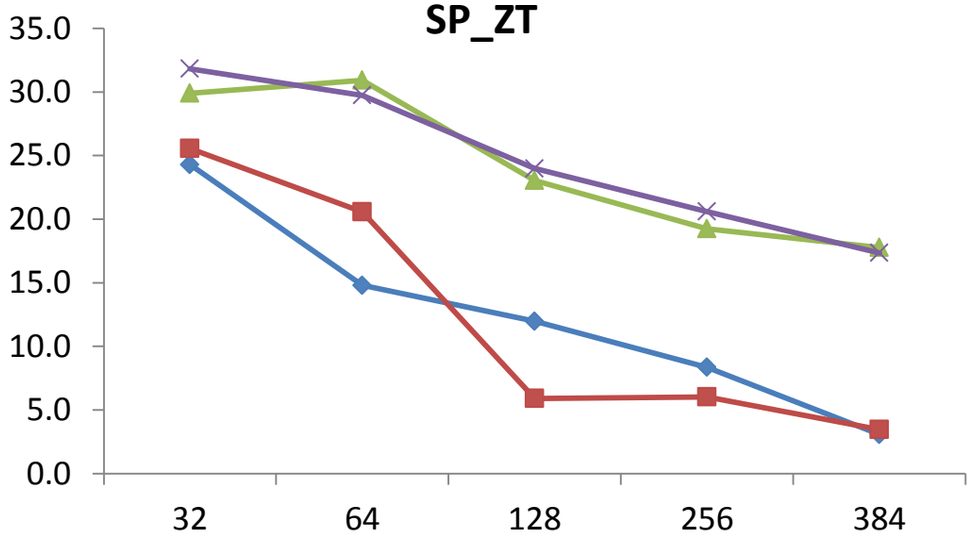




DP_ZT



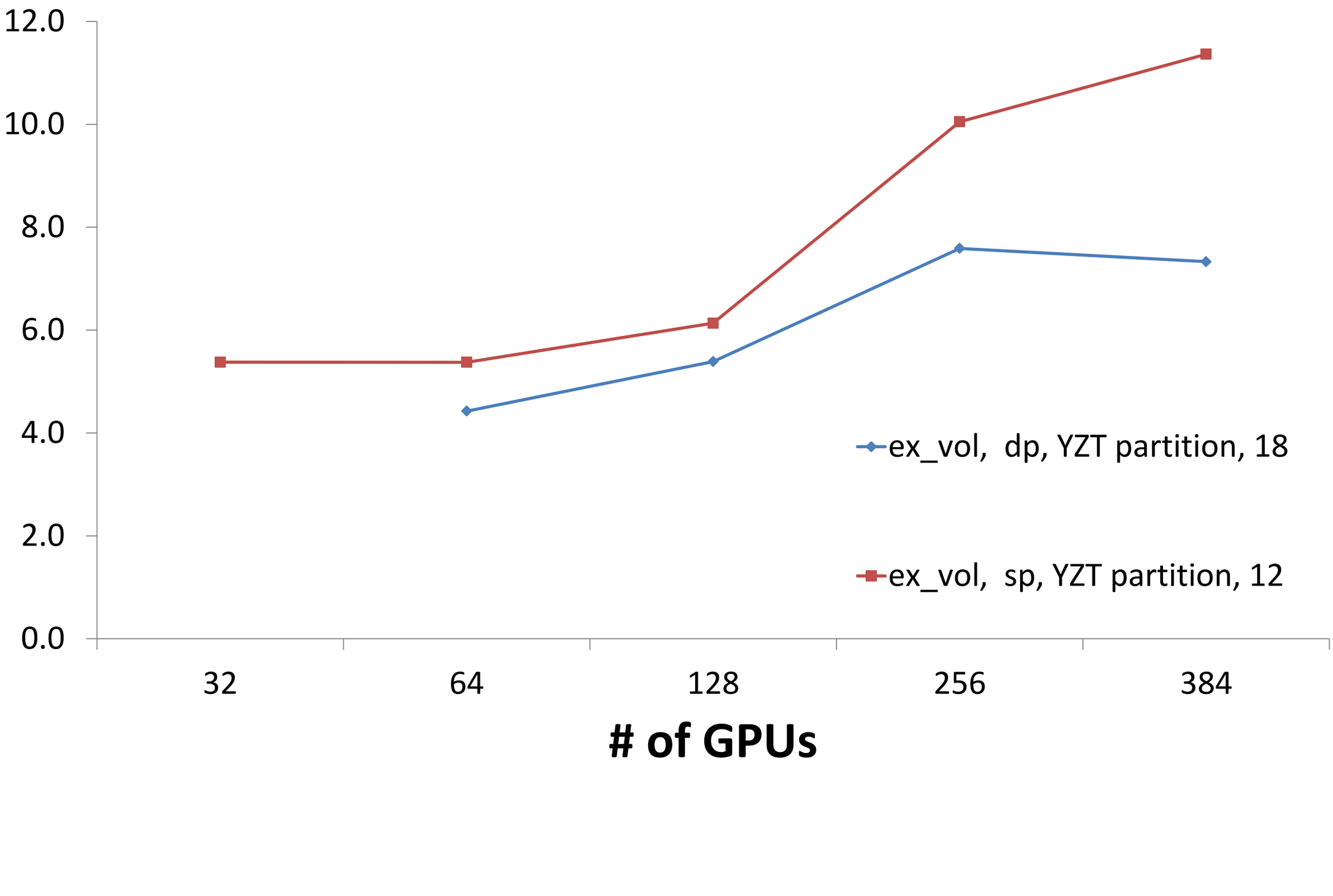
SP_ZT



# of GPUs

# of GPUs

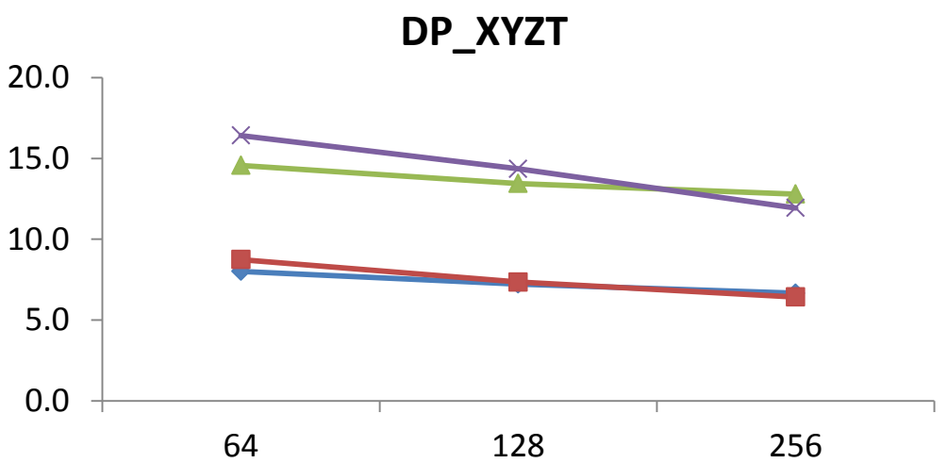
Longhorn: Cluster Speedup with GPUs



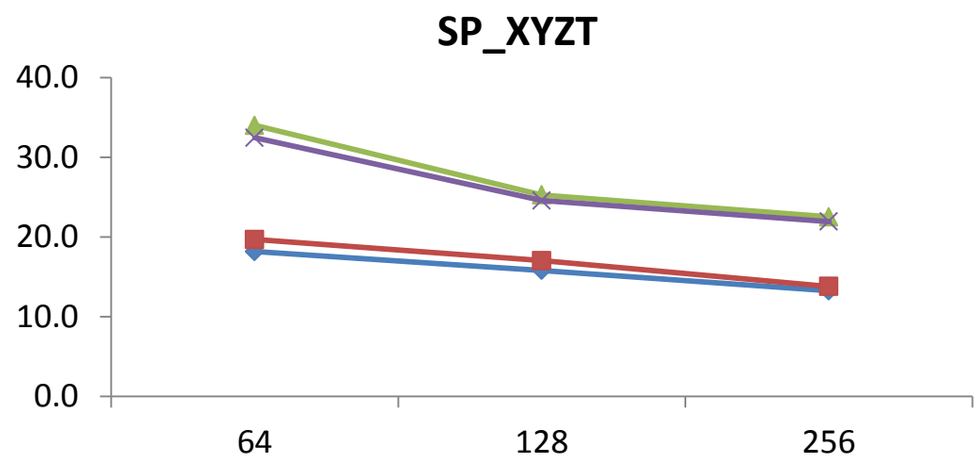
Keeneland@ NICS

- Each node
  - ❑ Dual hex-core Intel Westmere CPUs
  - ❑ 3 Nvidia M2070 GPUs
  - ❑ GPU:CPU_core ratio = 1:4

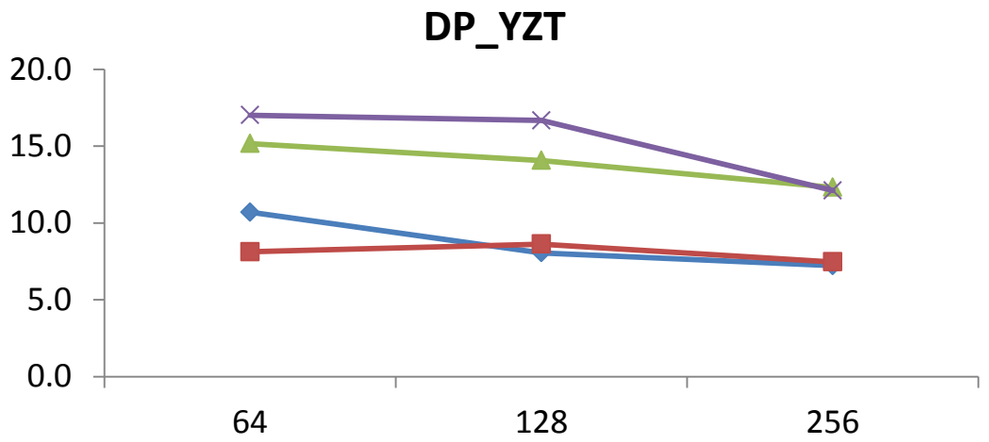
Gflops/GPU

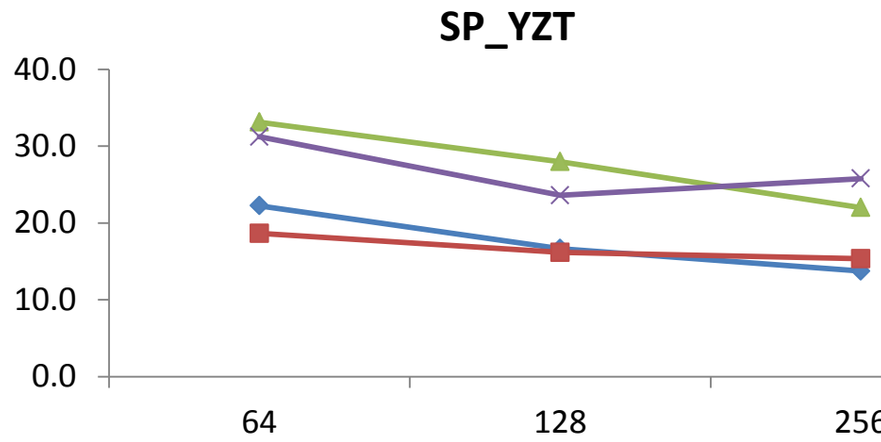


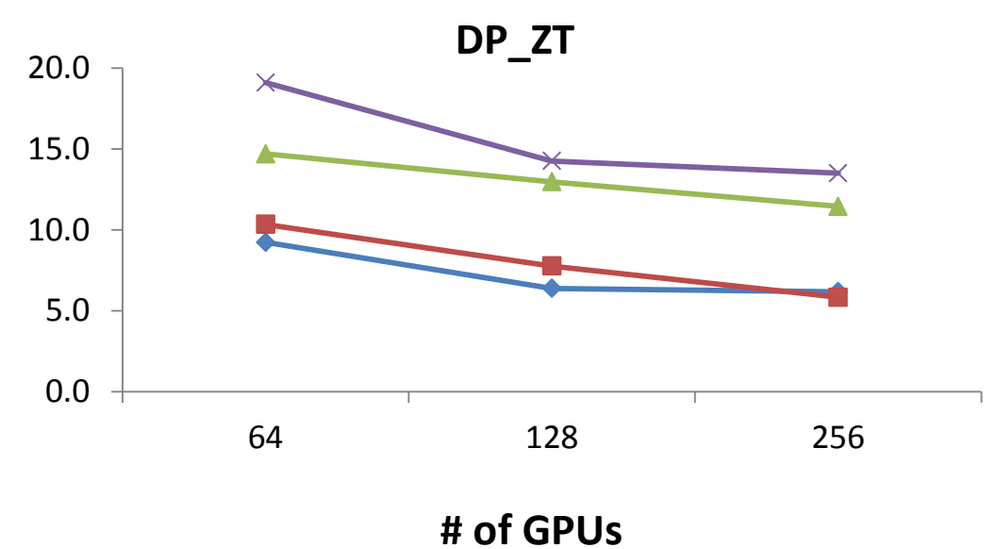
Gflops/GPU

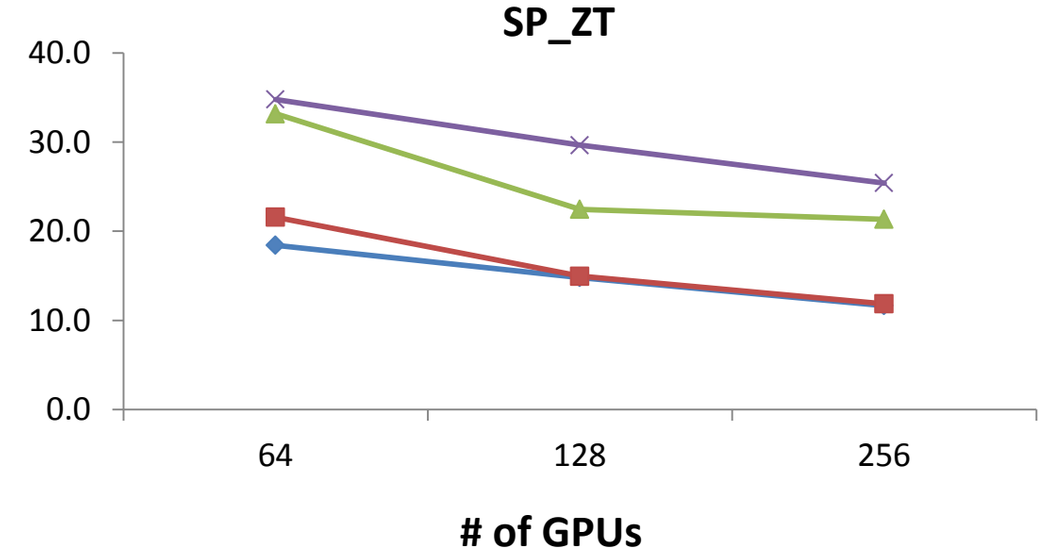


Keeneland

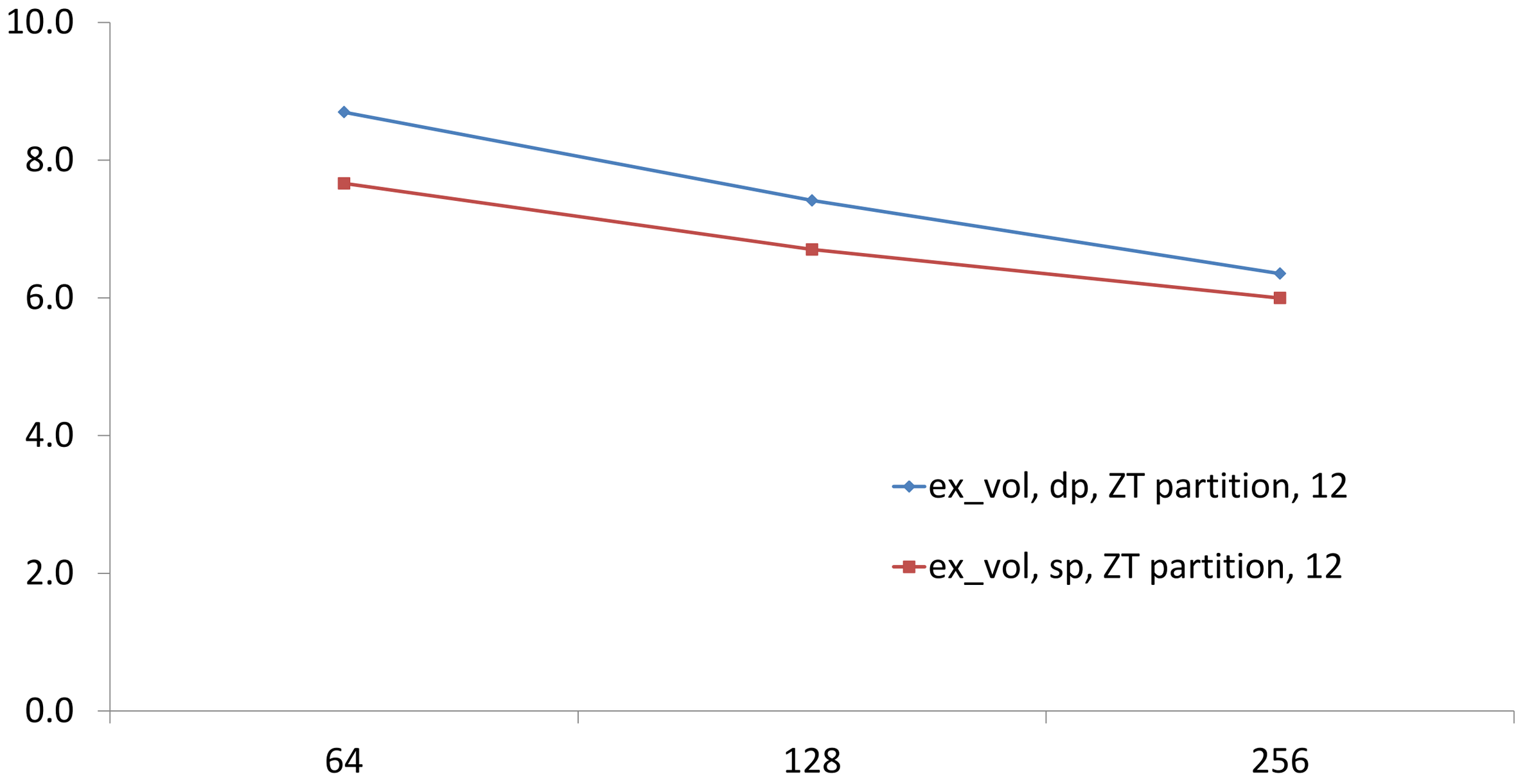








Keeneland: Cluster Speedup with GPUs



Fatlink routine and kernel perf

Machine	Precision	Perf w/ overhead (Gflops)	Kernel Perf only (Gflops)
Longhorn	DP	13.4	28.9
	SP	30.9	79.0
Keeneland	DP	15.2	43.8
	SP	33.1	93.8

64-GPU, extended-volume approach, YZT partitioning scheme, 18-reconstruct

- Kernel performance is up to 3x faster

Other components

	Description	Multi-GPU Status
CG	Update spinors using the neighboring links and spinors through Conjugate Gradient process	Production runs
FAT	Update fatlink using the neighboring links	Production runs
GF	Update the momentum using the neighboring links	Performance tuning
FF	Update the momentum using the neighboring links and spinors	Performance tuning

The current production run

- Jefferson Lab
  - Over 500 GT200 and Fermi GPUs for LQCD calculations

- Fermi Lab
  - In the process of acquiring a 128 Fermi GPU cluster for LQCD calculations

- LQCD GPUs jobs are in production in various GPU cluster
  - Lincoln/Ecog/AC @NCSA, dirac@NERSC, longhorn@TACC, etc
  - The upcoming forge @NCSA, titan@ ORNL

Future work

- Make use of new features
  - ~~GPU direct 1.0: Infiniband and CUDA share the same pinned memory~~
  - GPU direct 2.0: GPU peer to peer communication – difficult to use in our application given the current constraint
  - GPU direct 3.0: Use MPI's new GPU capability to send GPU data directly over the wire

- Integrate CG/Fat/GF/FF codes together to do gauge generation runs in large GPU clusters
  - Blue Waters: 3000 GPUs
  - Titan: 20,000 GPUs

- Make full use of all CPU cores to battle Amdahl's Law

Some of the works are already underway

Conclusion

- The fatlink computation code is optimized for multi-gpu in large GPU cluster, achieving up over 11x and 8x speedup in Longhorn and Keeneland cluster (both with 1:4 cpu core to gpu ratio)

- Two different approaches are implemented for multi-gpu and we find the extended volume approach to be much faster in large GPU runs.

- The current GPU-accelerated code used in production, but more work remains to be done

Questions?

References

- G.I. Egri, Z. Fodor, C. Hoelbling, S.D. Katz, D. Nogradi and K.K. Szabo, Lattice QCD as a video game, *Comput. Phys. Comm.* **177** (2007), p. 631 arXiv:hep-lat/0611022

- K. Barros, R. Babich, R. Brower, M.A. Clark and C. Rebbi, Blasting through lattice calculations using CUDA, LATTICE2008, *PoS* (2008), p. 045 arXiv:0810.5365 [hep-lat]

- M.A. Clark et al., ‘‘Solving Lattice QCD Systems of Equations Using Mixed Precision Solvers on GPUs,’’ Computer Physics Comm., 2009; http://arxiv.org/abs/0911. 3191v2.

- R. Babich, M. A. Clark, B, Joo, “Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice Quantum Chromodynamics,” Proc. International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing 2010), New Orleans, LA, November 2010

- 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*

- G. Shi, S. Gottlieb, A. Torok, and V. V. Kindratenko, “Design of MILC lattice QCD application for GPU clusters," in IPDPS. IEEE, 2011

- R. Babich, M. A. Clark, B, Joo, G. Shi, R.C. Brower, S. Gottlieb, “Scaling Lattice QCD beyond 100 GPUs,” Proc. International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing 2011), Seattle, WA, November 2011

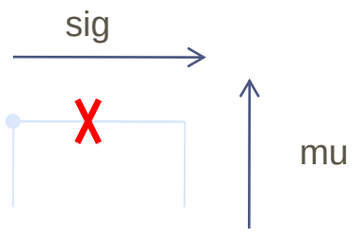
What GPU is suited for LQCD computation

- LQCD inverter can detect errors ➔ Consumers cards can work
  - Consumer cards are cheap and faster
  - Tesla cards are more reliable, have larger memory, ECC and better customer service
  - JLAB has a mixed collection of both types. The new 128-GPU cluster in Fermi Lab is planning on using Telsa cards

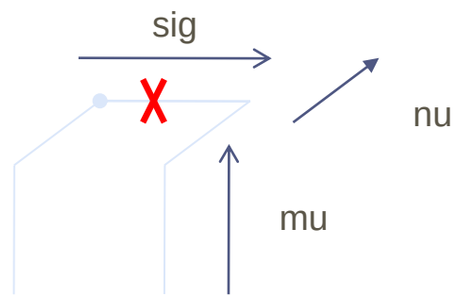
- Other parts of the LQCD, when ported, must use ECC-protected Tesla GPUs

Backup slides

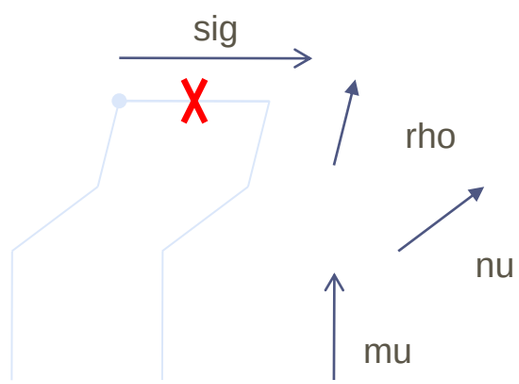
Fermion Force contribution terms



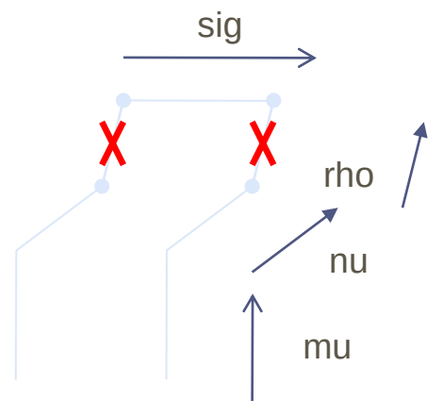
a) 1st link in 3-link path



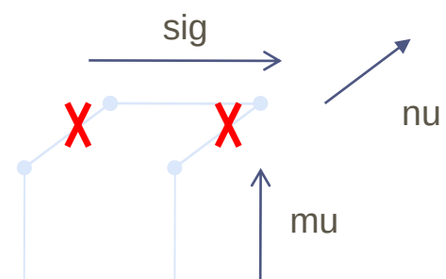
b) 2nd link in 5-link path



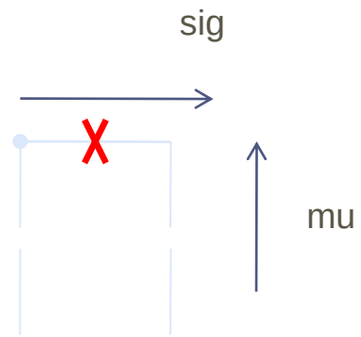
c) 3rd link in 7-link path



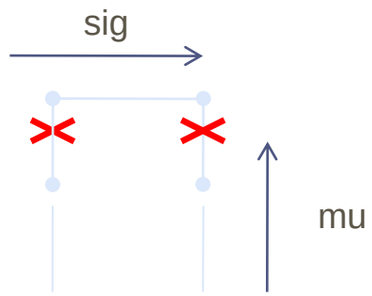
d) 2nd and 4th links in 7-link path



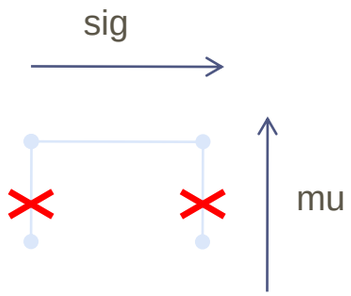
e) 1st and 3rd links in 5-link path



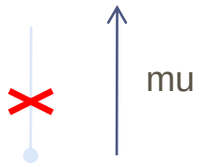
f) 2nd link in Lepage path



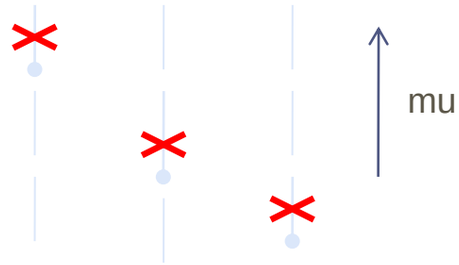
g) 1st and 3rd links in Lepage path



h) 0th and 2rd links in 3-link path



i) 1 link path



j) 3 link path

GF and FF Performance table

	Standalone (Gflops)	Performance MILC sees (with PCIe overhead) (Gflops)	Flags
Gauge Force	186	112	-DUSE_GF_GPU
Fermion Force	107	94	-DUSE_FF_GPU

- Single precision, 12-reconstruct
- The experiment is done in GTX280
- Gauge Field is relatively easy to port to GPU
- Fermion Field is much more complicated