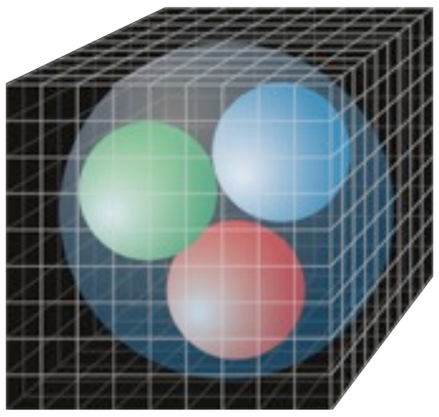
Blasting through lattice QCD computations using GPUs



Michael Clark

Harvard University

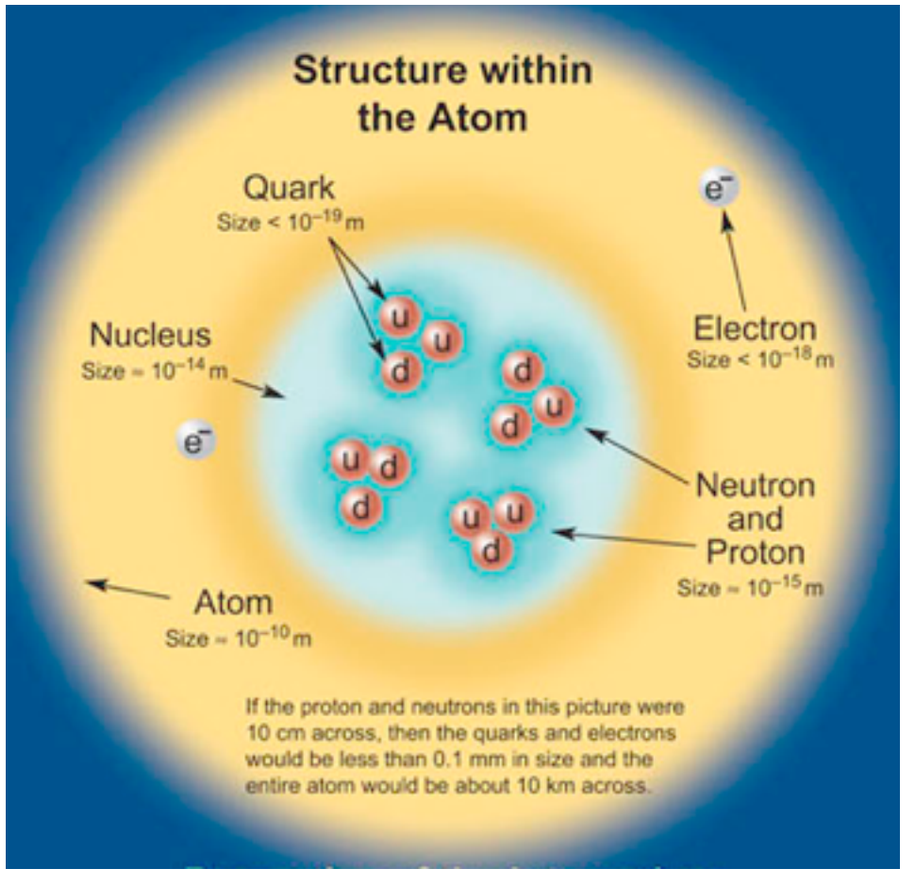
Outline

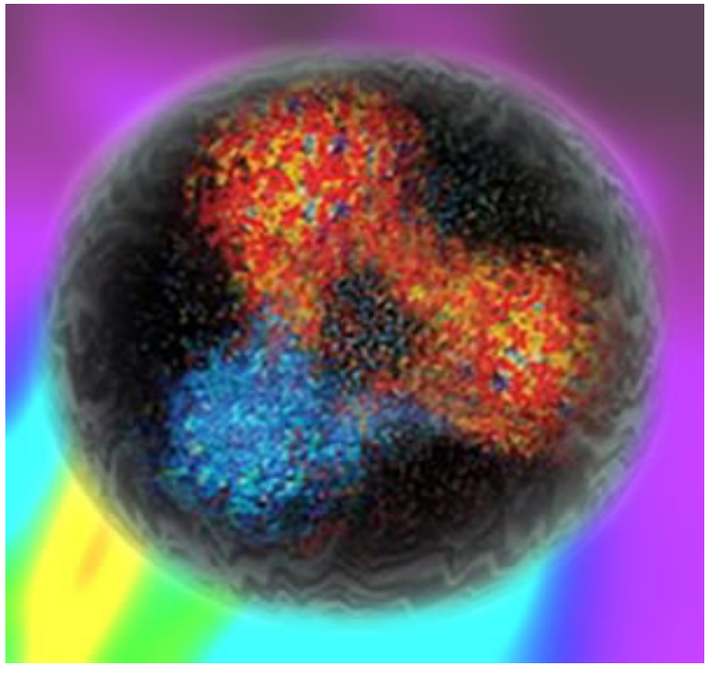
- Introduction to lattice QCD
- Introduction to GPUs
- QUDA
- Multiple GPUs
- Summary and conclusions

Quarks and Gluons

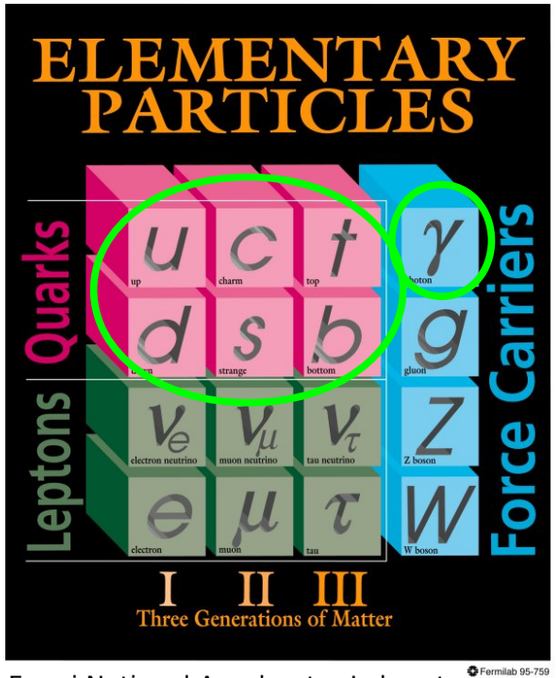
- The strong force is one of the basic forces of nature (along with gravity, electromagnetism and the weak force)

- It's what binds together the quarks and gluons in the proton (and the neutron, as well as hundreds of other particles seen in accelerator experiments).





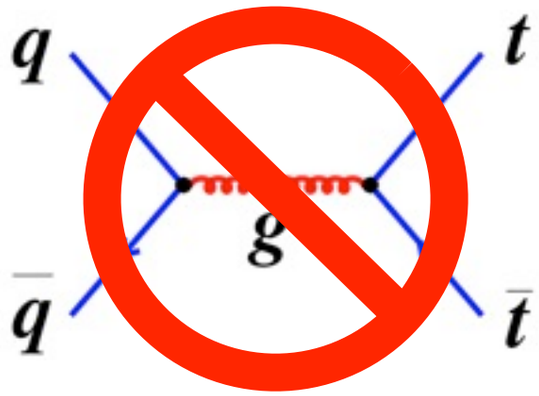
Thomas Jefferson National Accelerator Facility



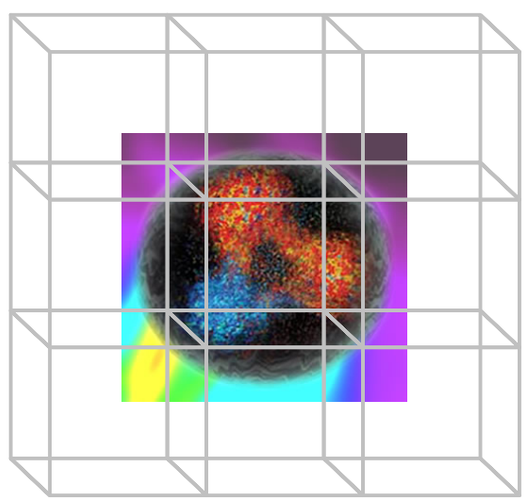
Fermi National Accelerator Laboratory

Quantum Chromodynamics

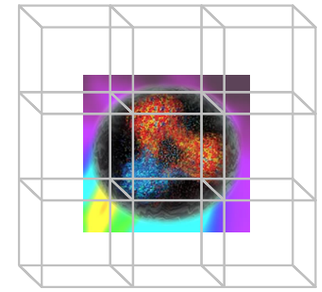
- The theory of the strong force is called Quantum Chromodynamics (QCD)
- Similar to QED (but electric charge replaced by 3 “color” charges)
- Unlike QED, QCD at everyday energies cannot be treated with perturbation theory



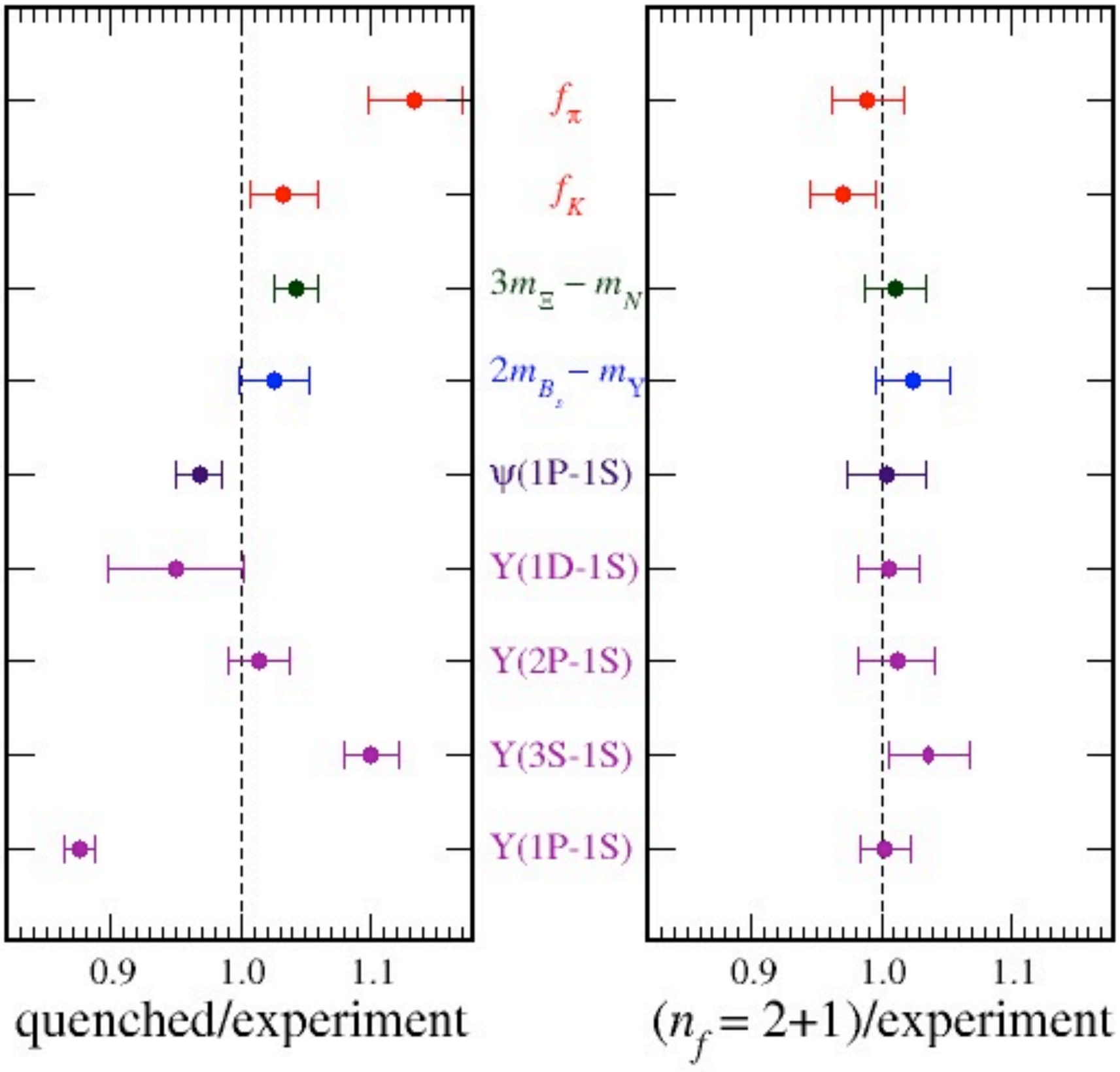
- Instead evaluate the QCD path integral numerically, sampling all possible configurations of the quark and gluon fields in a region of spacetime
- Continuous and infinite spacetime
  ⇒4-d grid (lattice), hence **lattice QCD**
- Ab initio calculation to verify QCD

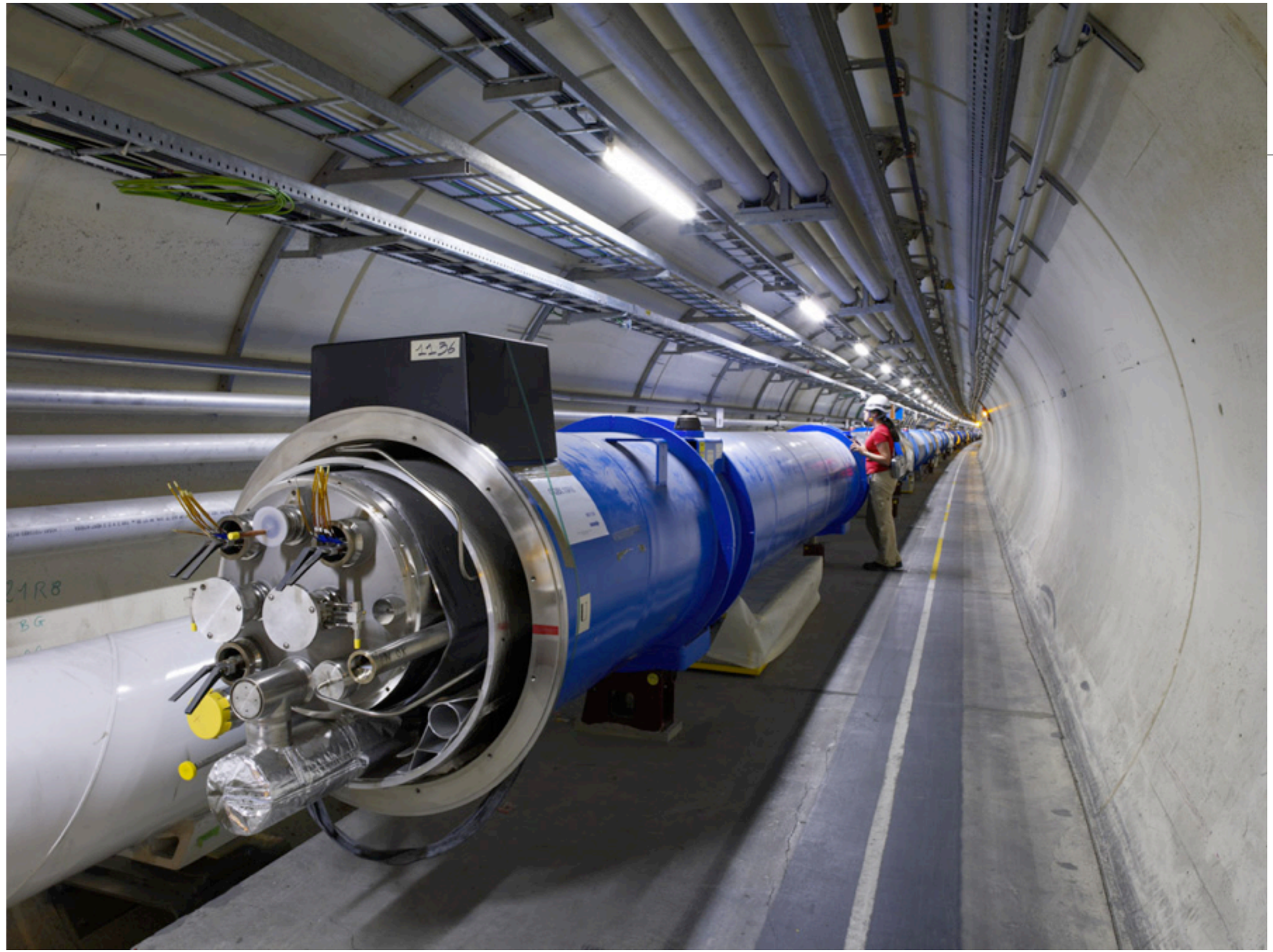


Lattice Quantum Chromodynamics



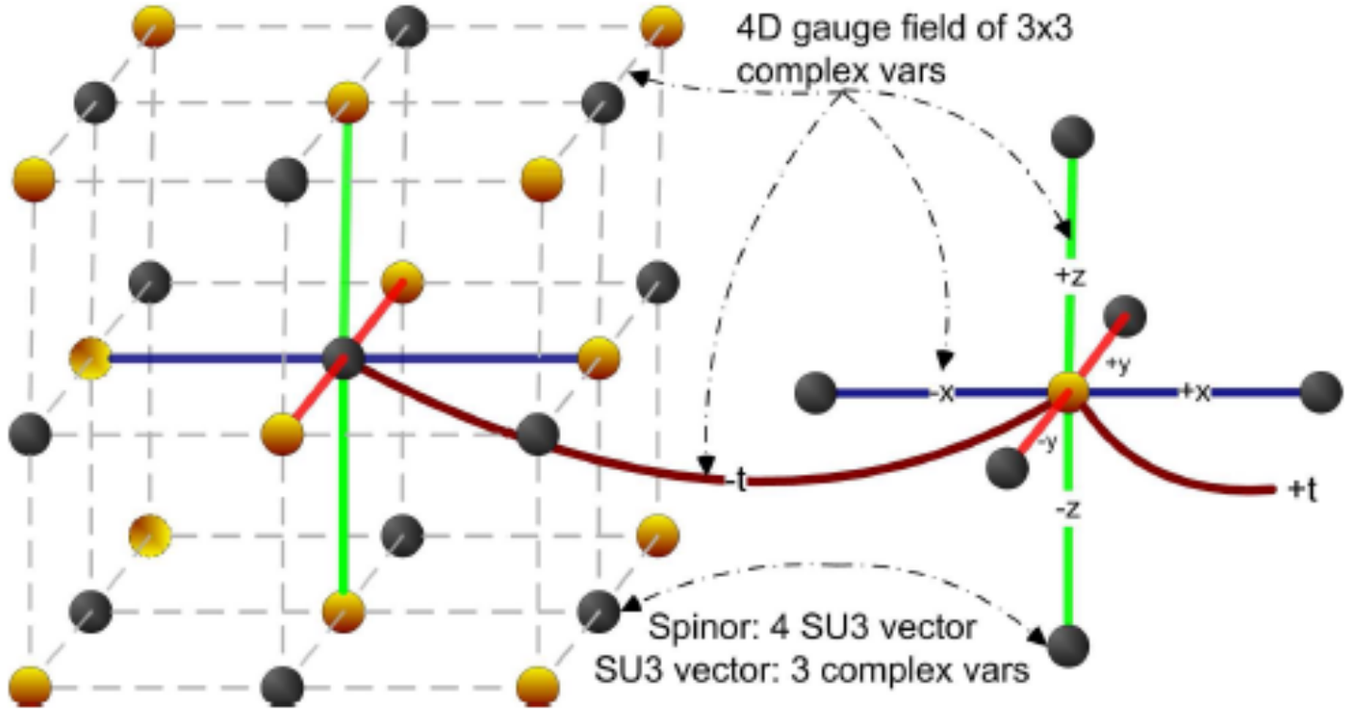
- LQCD underpins many current calculations in nuclear and high energy physics
  - Spectrum of Hadrons (resonances, hybrids)
  - Nuclear Interactions (binding of quarks into light nuclei)
  - Test of Standard Model (e.g CKM matrix elements)
  - Beyond the Standard Model (BSM) physics


f_π
f_K
$3m_\Xi - m_N$
$2m_{B_s} - m_\Upsilon$
ψ(1P-1S)
Υ(1D-1S)
Υ(2P-1S)
Υ(3S-1S)
Υ(1P-1S)
0.9
1.0
1.1
quenched/experiment
0.9
1.0
1.1
$(n_f = 2+1)$/experiment


1236
21R8
BG

Formulating Lattice QCD

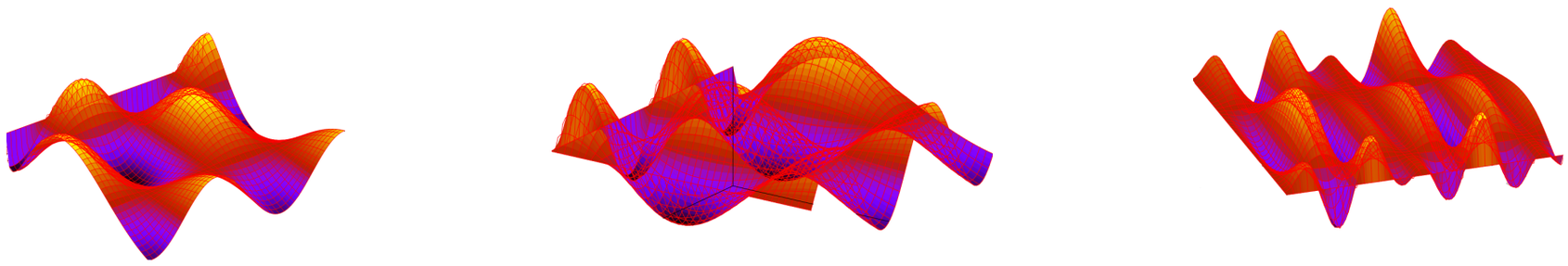
- Quark fields live on the lattice sites
  - “spinors”
  - 12 complex numbers
- Gluon fields live on the links
  - SU(3) “Color matrices”
  - 18 complex numbers



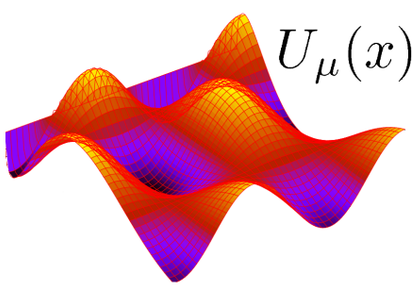
Four dimensional space-time Lattice QCD.

Steps in a Lattice QCD calculation

1. Generate an ensemble of gluon field configurations, $\{U_\mu(x)\}$



2. Compute quark propagators in these fixed backgrounds by solving the Dirac equation for various right-hand sides

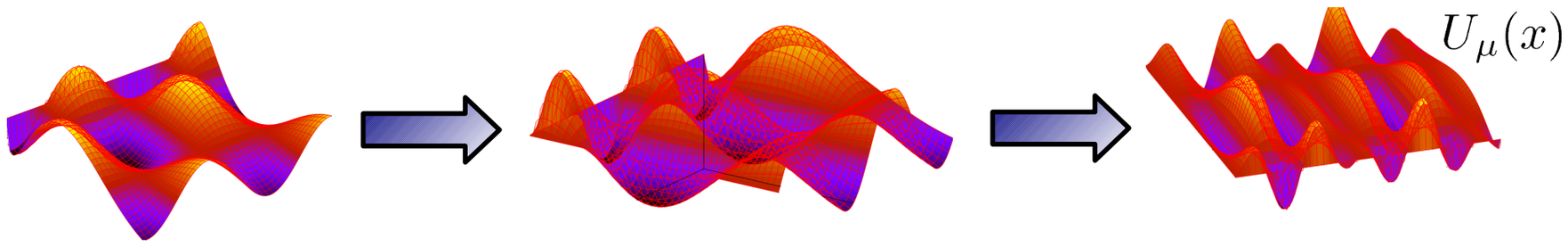


$$D_{ij}^{\alpha\beta}(x,x';U)\psi_j^\beta(x) = \eta_i^\alpha(x')$$

or “Ax=b”

Configuration generation

- Markov process (sequential)



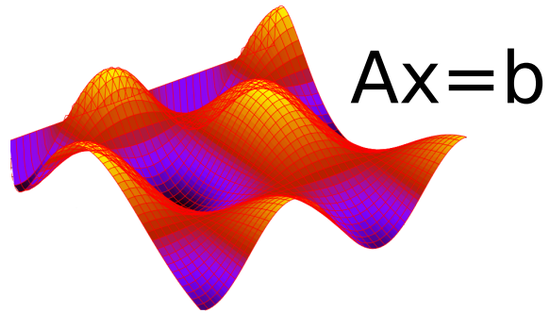
- Requires >O(10 TFLOPS) = BlueGene, Cray etc. **Capability Computing**

“Intrepid” - Argonne Leadership Computing Facility

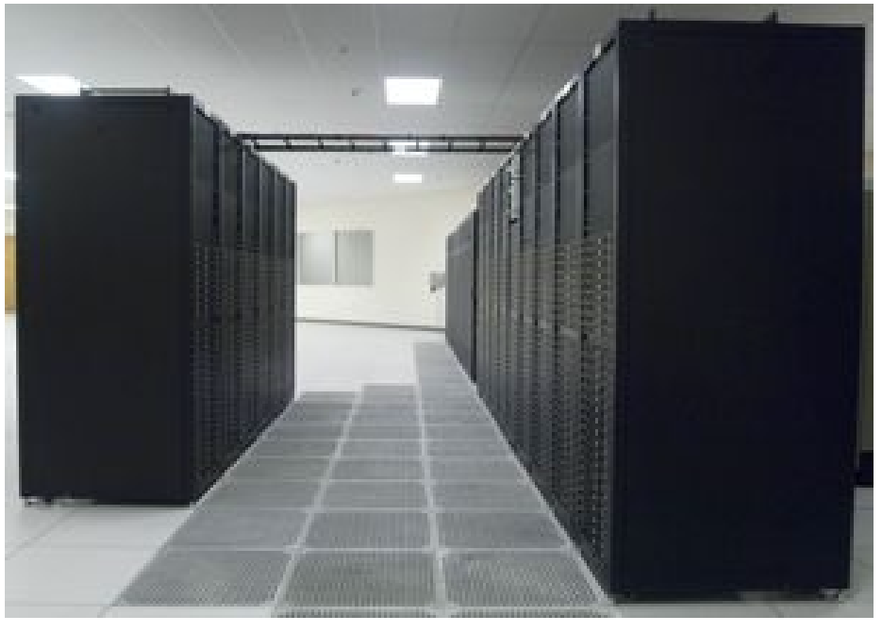
“Jaguar” - Oak Ridge Leadership Computing Facility

Computing Propagators

- Analysis stage suitable for capacity type machines
- Accounts for ~50% of total cycles
- Each job requires tens of cluster nodes



Capacity Computing



(Clusters dedicated to lattice QCD at Fermilab and Jefferson Lab)

The Dirac operator

- Quark interactions described by the Dirac operator
- On the lattice this becomes a large sparse matrix M
- Quark physics reduced to solving linear equations

$$Mx = b$$

- Many discretisations of M possible
  - We focus on the Wilson

Wilson Matrix

Dirac spin projector matrices
(4x4 *spin* space)

SU(3) QCD gauge field
(3x3 *color* space)

A is the clover matrix
(12x12 *spin* $\otimes$ *color* space)

$$M_{x,x'} = -\frac{1}{2}\sum_{\mu=1}^{4}\left(P^{-\mu}\otimes U_{x}^{\mu}\,\delta_{x+\hat{\mu},x'} + P^{+\mu}\otimes U_{x-\hat{\mu}}^{\mu\dagger}\,\delta_{x-\hat{\mu},x'}\right) + (4+m+A_x)\delta_{x,x'}$$

$$\equiv -\frac{1}{2}D_{x,x'} + (4+m+A_x)\delta_{x,x'}$$

m quark mass parameter

Nearest neighbor Local

Wilson Matrix

Dirac spin projector matrices
(4x4 *spin* space)

SU(3) QCD gauge field
(3x3 *color* space)

A is the clover matrix
(12x12 *spin* $\otimes$ *color* space)

$$M_{x,x'} = -\frac{1}{2}\sum_{\mu=1}^{4}\left(P^{-\mu}\otimes U_{x}^{\mu}\,\delta_{x+\hat{\mu},x'} + P^{+\mu}\otimes U_{x-\hat{\mu}}^{\mu\dagger}\,\delta_{x-\hat{\mu},x'}\right) + (4+m+A_x)\delta_{x,x'}$$

$$\equiv -\frac{1}{2}D_{x,x'} + (4+m+A_x)\delta_{x,x'}$$

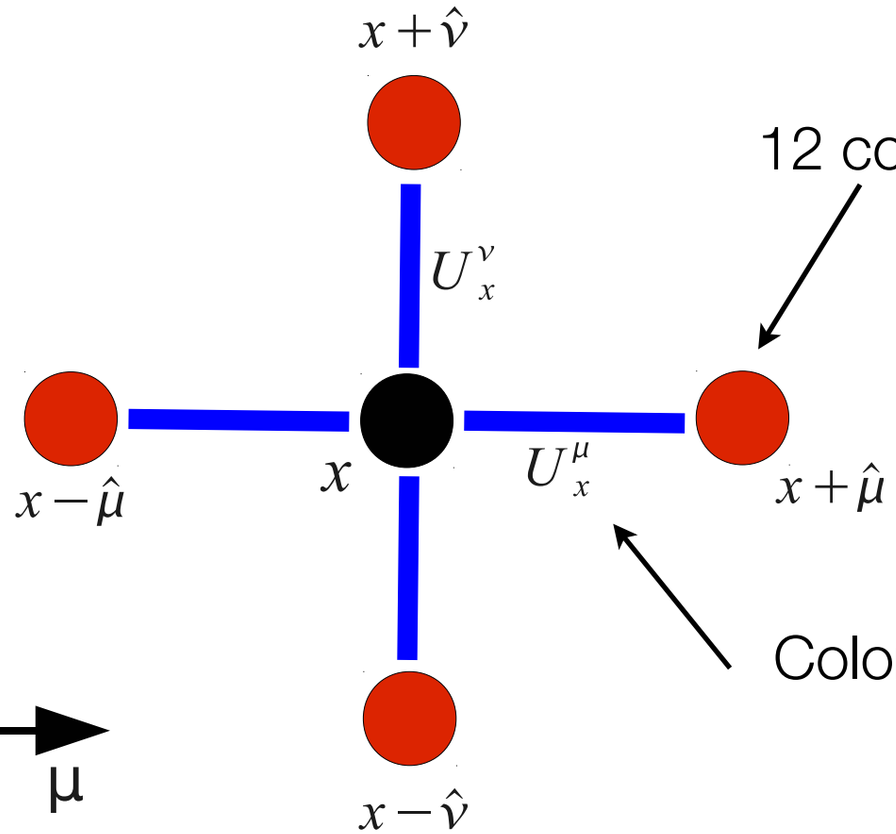
m quark mass parameter

Nearest neighbor Local

4d nearest neighbor stencil operator acting on a vector field

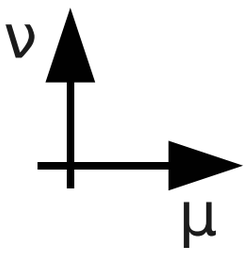
Wilson Matrix

$$D_{x,x'} =$$



12 complex component vector (spinor)

"dslash"



Color (gauge) matrices live between sites

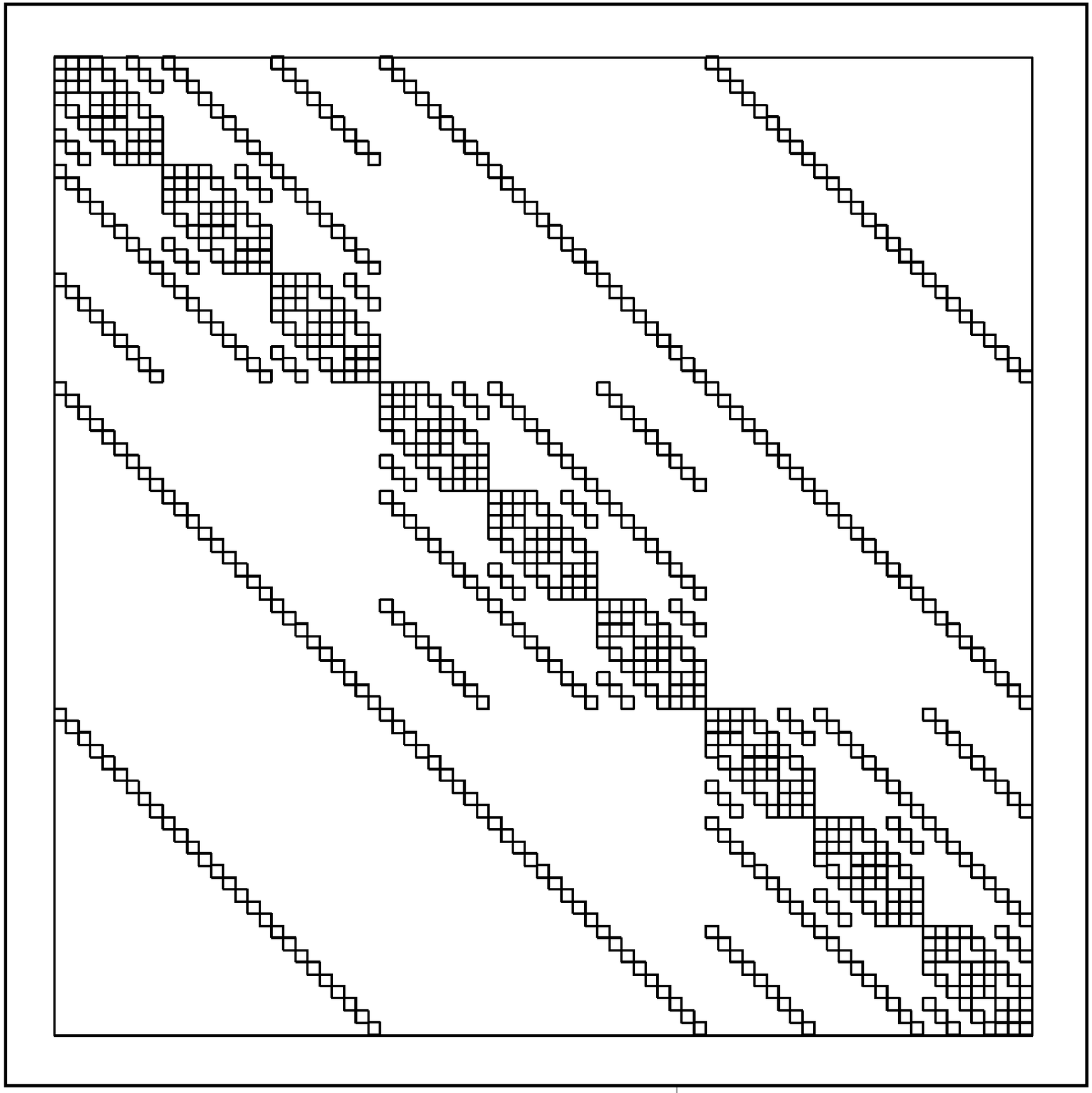
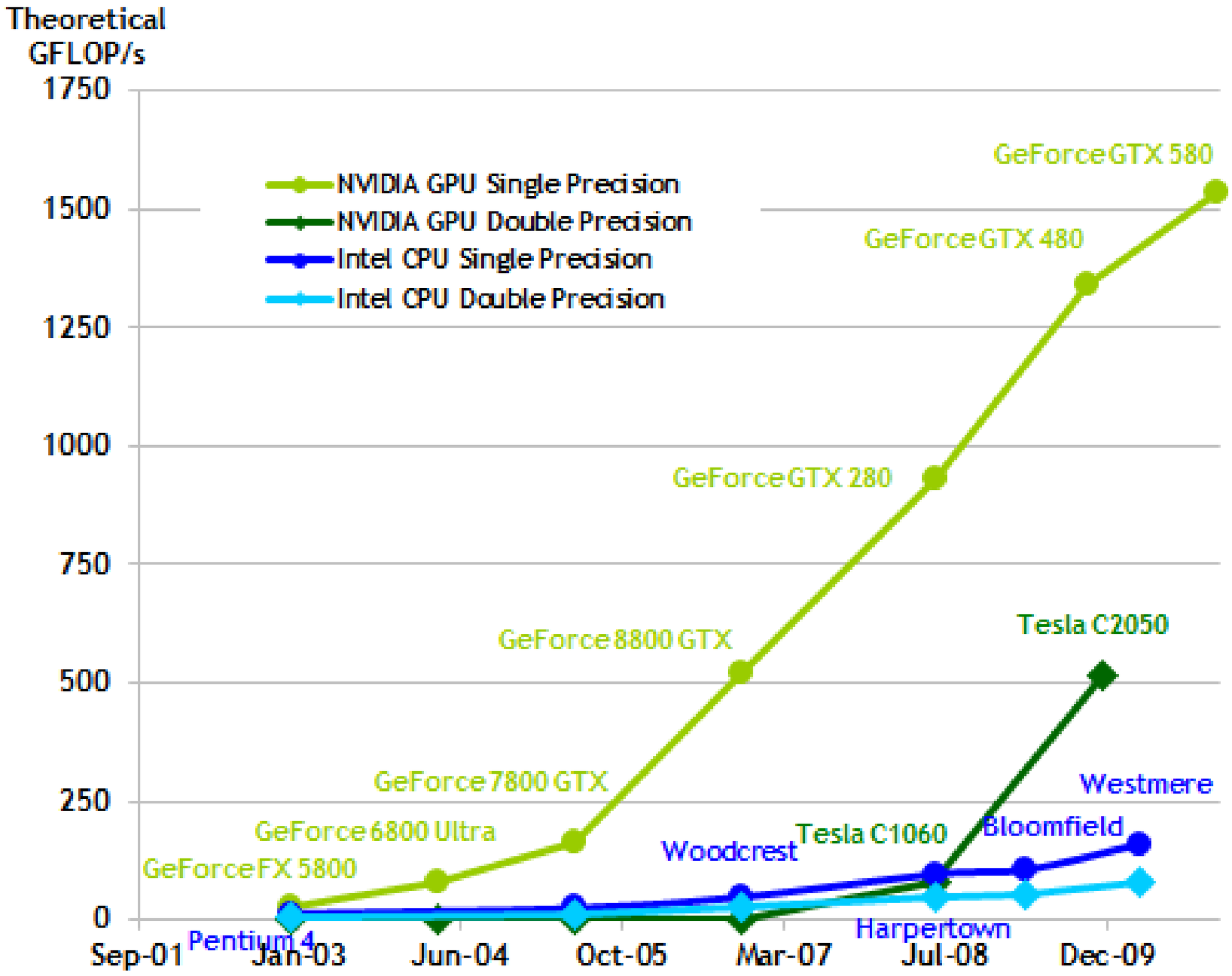
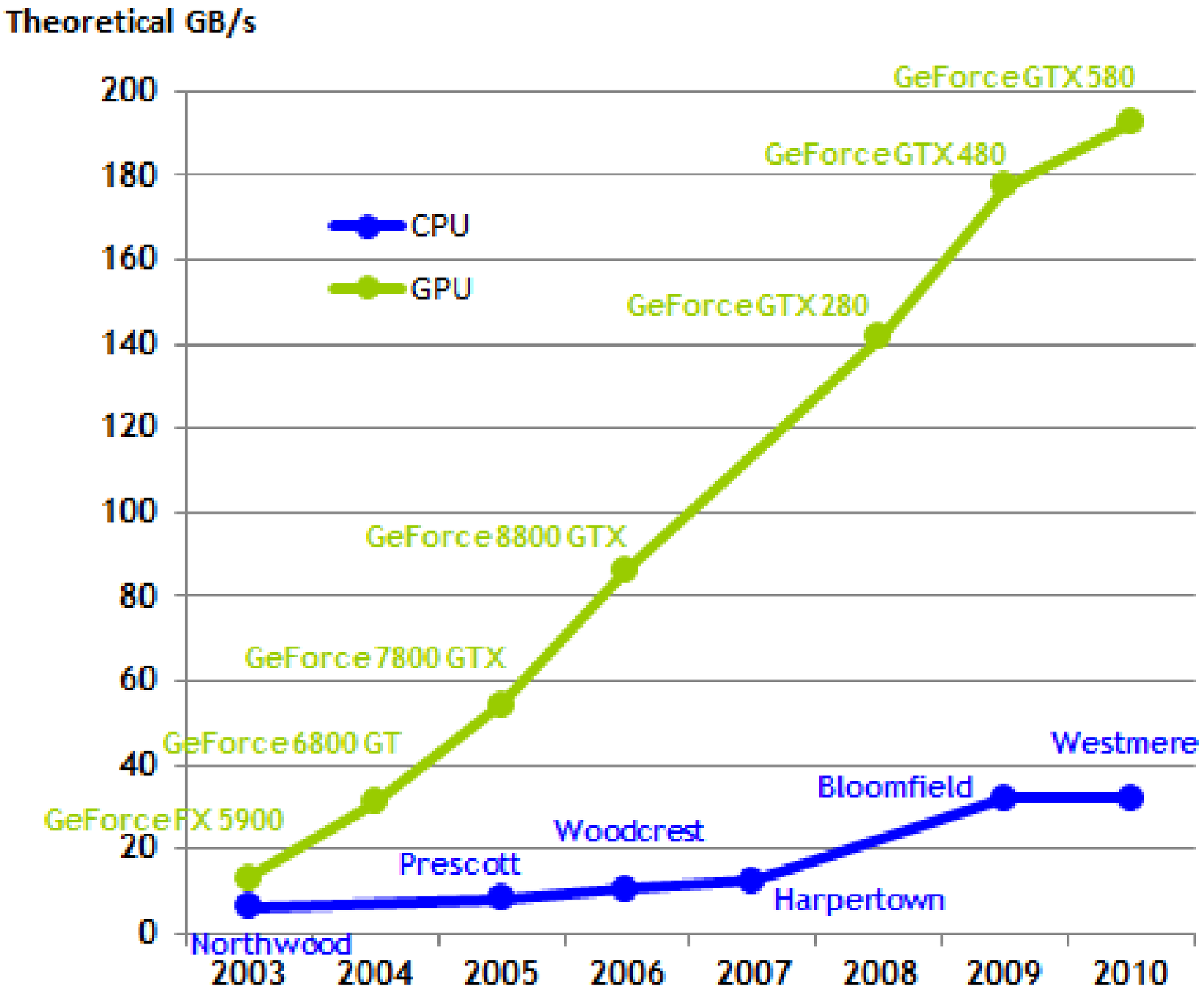


figure by C. Pickles

Solving $Mx=b$

- Iterative Krylov solvers standard method
  - CG, BiCGstab, etc.
- Condition number given by ~(quark mass)$^{-1}$
  - Up / down quark masses are light
  - 10^8 DOF x 100,000 right hand sides - Computationally expensive
  - 50-99.9% of QCD execution time (generation or observables)
- QCD huge consumer of super computer cycles
- Current parameters too far from reality
  - Lattice spacing too coarse
  - Volumes too small
  - Quarks too heavy
  - Full QCD verification requires Peta/Exascale computing
- BSM applications require even more cycles


Theoretical GFLOP/s
1750
1500
1250
1000
750
500
250
0
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Single Precision
Intel CPU Double Precision
GeForce GTX 580
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Tesla C2050
Tesla C1060
Westmere
Bloomfield
Woodcrest
Harpertown
Pentium 4
Sep-01
Jan-03
Jun-04
Oct-05
Mar-07
Jul-08
Dec-09


Theoretical GB/s
200
180
160
140
120
100
80
60
40
20
0
CPU
GPU
GeForceGTX580
GeForceGTX480
GeForceGTX280
GeForce8800GTX
GeForce7800GTX
GeForce6800GT
GeForceFX5900
Westmere
Bloomfield
Woodcrest
Prescott
Harpertown
Northwood
2003
2004
2005
2006
2007
2008
2009
2010

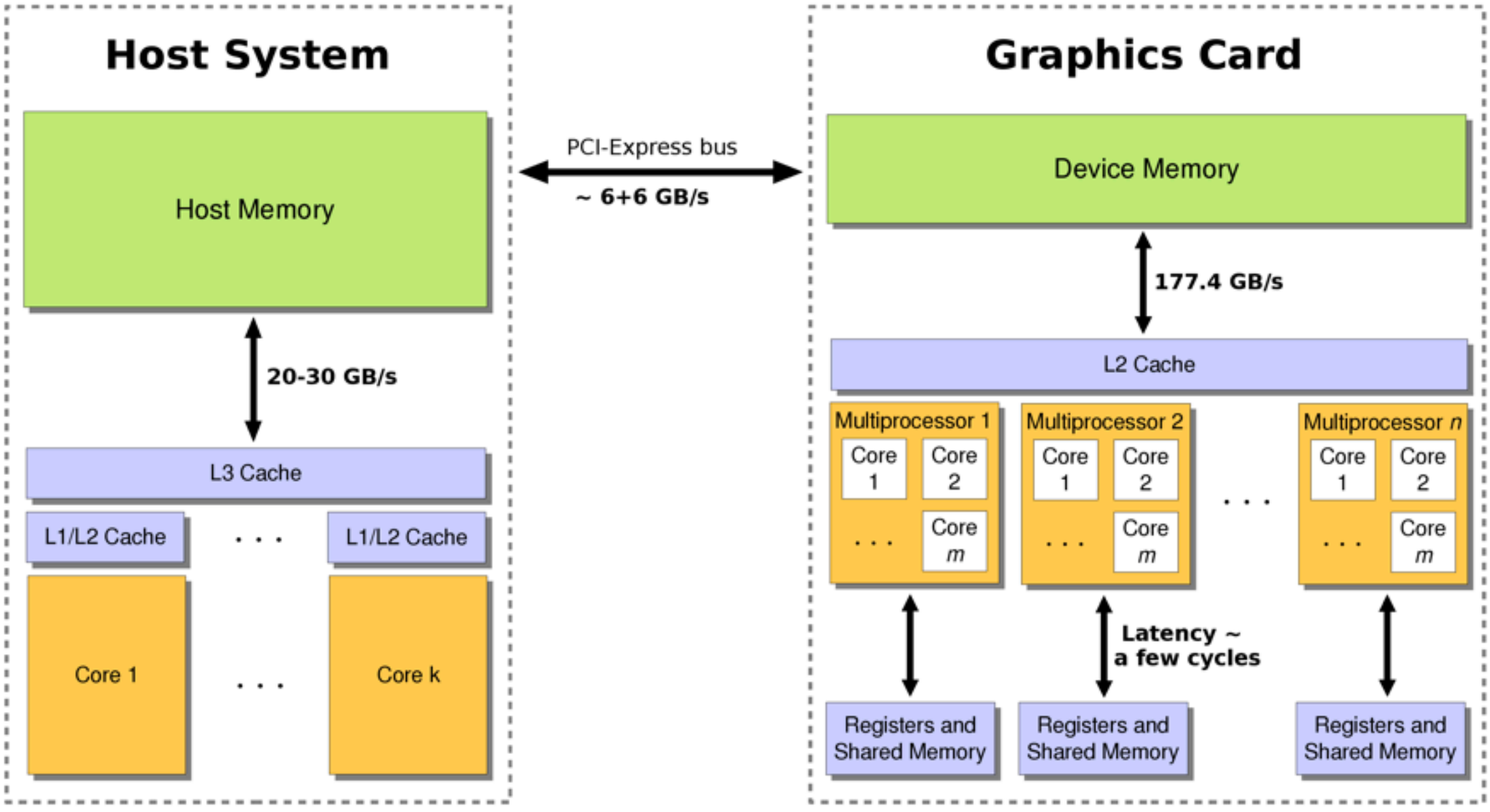
GPUs

- GPUs are now ubiquitous in HPC
- Independent highly parallel processors
  - Very high peak flop rate
  - High bandwidth device memory
  - Narrow channel to host (PCIe)

		GB/s	Gflops		GiB
Card	Cores	Bandwidth	32-bit	64-bit	RAM
GeForce 8800 GTX	128	86.4	518	N/A	0.75
Tesla C870	128	76.8	518	N/A	1.5
GeForce GTX 285	240	159	1062	88	1.0 - 2.0
Tesla C1060	240	102	933	78	4.0
GeForce GTX 480	480	177	1345	168	1.5
Tesla C2050	448	144	1030	515	3.0

- Can be programmed using C-like languages
  - Nvidia's C for CUDA
  - OpenCL

GPU Memory Hierarchy



Who's using GPUs for QCD?

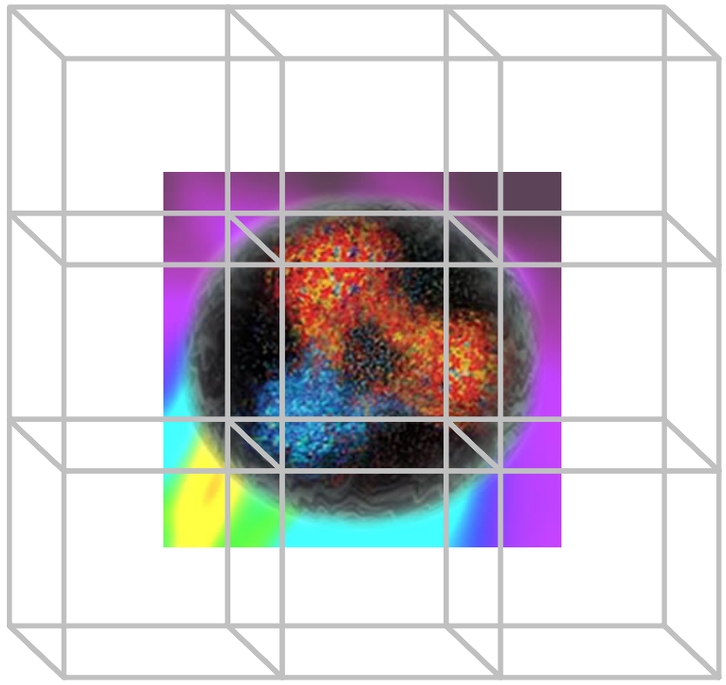
- First report: "Lattice QCD as a Video Game", (Egri et. al.) in 2006
  - Coded in OpenGL
- In the U.S.
  - QUDA Library
  - Wuppertal Group used for BSM Studies (Kuti et. al.)
  - Alexandru et. al. @ GWU
- Worldwide (not in any particular order, not complete)
  - Wuppertal Group (Z. Fodor et. al.)
  - IRISA - France (F. Bodin et. al.)
  - Pisa - Italy (A. Di Giacomo et. al.)
  - Japan (KEK) (Hayakawa et. al.)
  - Taiwan (T-w Chiu et. al.)
  - Brazil (Attilio Cucchieri)

The QUDA library

QUDA - QCD on CUDA

- QUDA is a highly optimized open source QCD library for CUDA GPUs
  - Primary developers Babich, Clark, Joó, Shi (and others)
  - Interfaces to major software physics packages (Chroma, MILC, CPS)
  - Project webpage http://lattice.github.com/quda
- Provides optimized Dirac operator matrix-vector products (and more)
  - Wilson, clover, improved staggered, domain wall, twisted mass
- Mixed-precision linear solvers
  - CG, BiCGstab, GCR
  - Supports multiple GPUs

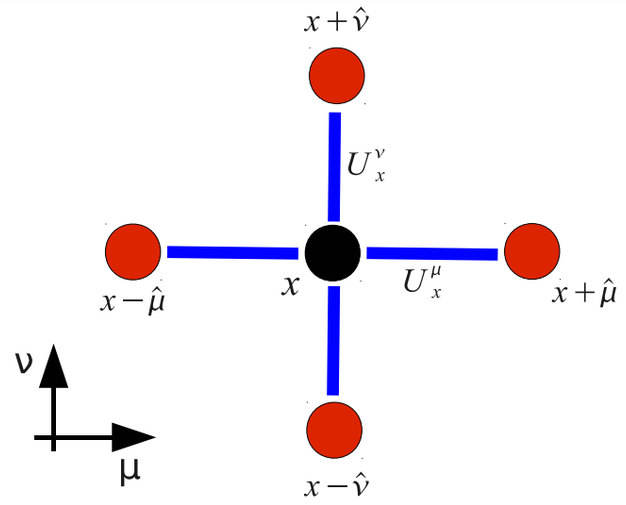
General Strategy



- Assign a single spacetime point to each thread -> V = XYZT threads
  - Map 4-d spacetime index to a 1-d thread index
  - $V = 24^4$ => 3.3×10^6 threads
  - Fine grained parallelization
- Minimize memory traffic even at the expense of additional computation

Mapping the Wilson operator to CUDA

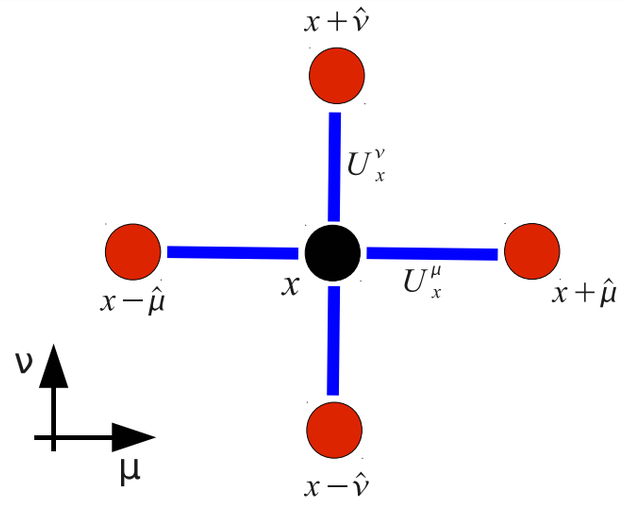
$$D_{x,x'} =$$



- Each thread must
  - Load the neighboring spinor (24 numbers x8)
  - Load the color matrix connecting the sites (18 numbers x8)
  - Do the computation
  - Save the result (24 numbers)
- Arithmetic intensity
  - 1320 floating point operations per site
  - 1440 bytes per site (single precision)
  - 0.92 naive arithmetic intensity

Mapping the Wilson operator to CUDA

$$D_{x,x'} =$$



- Each thread must
  - Load the neighboring spinor (24 numbers x8)
  - Load the color matrix connecting the sites (18 numbers x8)
  - Do the computation
  - Save the result (24 numbers)
- Arithmetic intensity
  - 1320 floating point operations per site
  - 1440 bytes per site (single precision)
  - 0.92 naive arithmetic intensity

		GB/s	Gflops		GiB
Card	Cores	Bandwidth	32-bit	64-bit	RAM
GeForce 8800 GTX	128	86.4	518	N/A	0.75
Tesla C870	128	76.8	518	N/A	1.5
GeForce GTX 285	240	159	1062	88	1.0 - 2.0
Tesla C1060	240	102	933	78	4.0
GeForce GTX 480	480	177	1345	168	1.5
Tesla C2050	448	144	1030	515	3.0

bandwidth bound

Reducing Memory Traffic

- SU(3) matrices are all unitary complex matrices with det = 1
  - 12 number parameterization $\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \longrightarrow \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix}$ **c** = (**a**x**b**)*
    - Reconstruct full matrix on the fly in registers (flops are free)

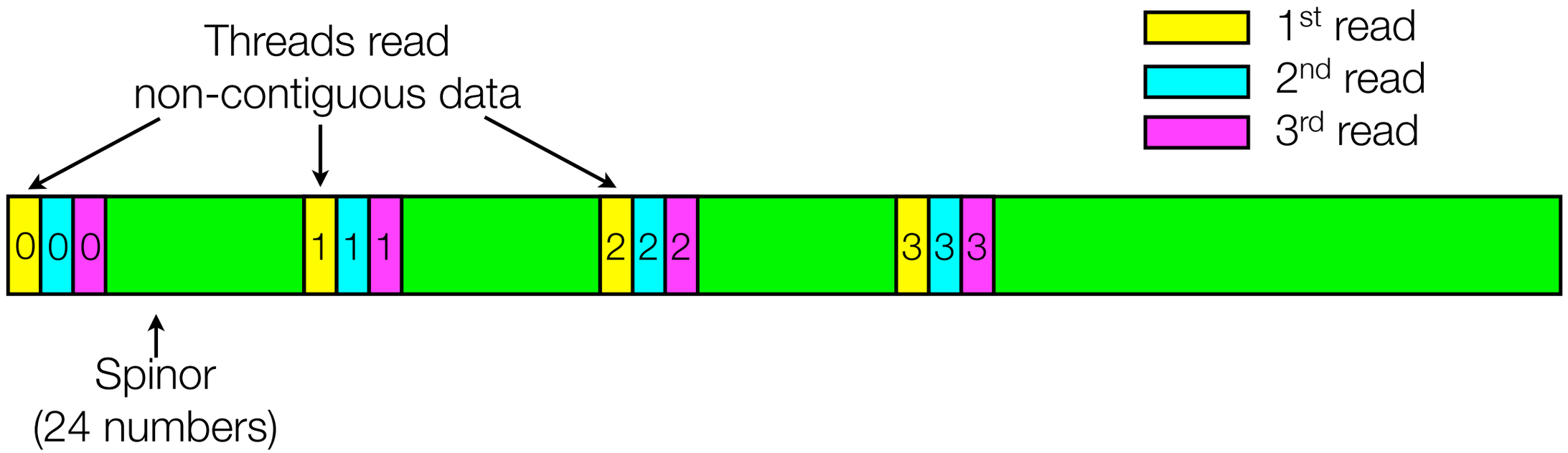
- Impose similarity transforms to increase sparsity
  - Globally change Dirac matrix basis

$$P^{\pm 4} = \begin{pmatrix} 1 & 0 & \pm 1 & 0 \\ 0 & 1 & 0 & \pm 1 \\ \pm 1 & 0 & 1 & 0 \\ 0 & \pm 1 & 0 & 1 \end{pmatrix} \longrightarrow P^{+4} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} P^{-4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

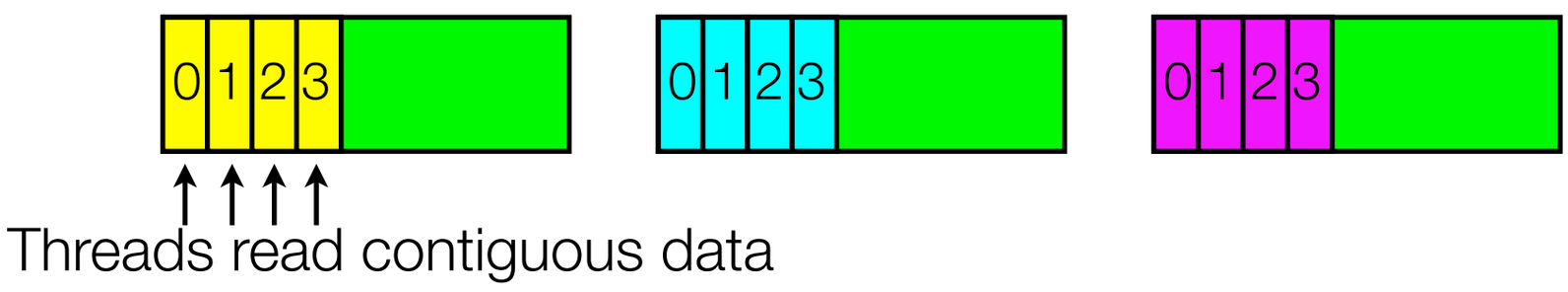
- Use 16-bit fixed point representation

Field Ordering

- Typical CPU spinor field ordering: array of spinors (V x 24 floats)



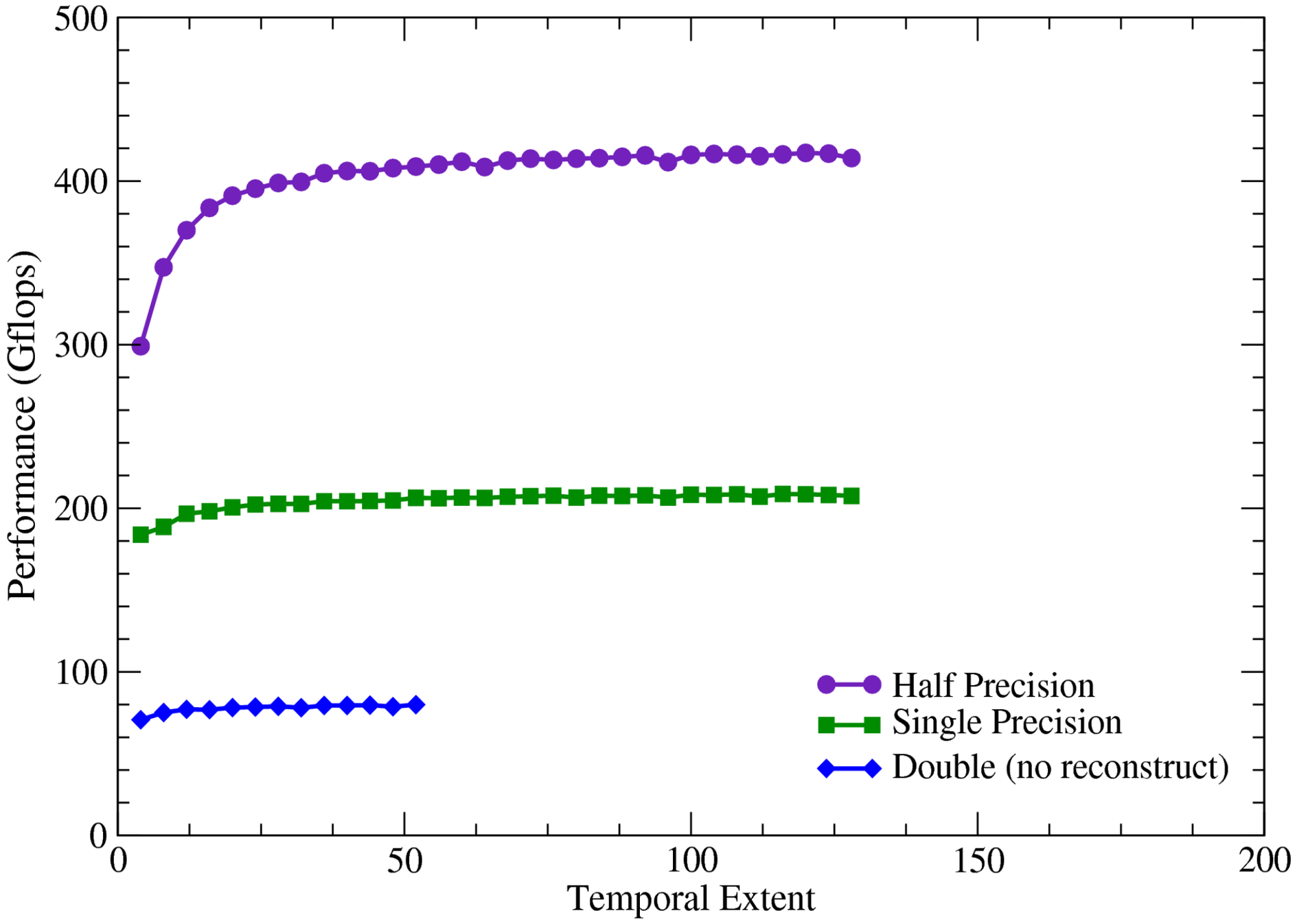
- Reorder fields for coalescing: 6V x float4



- Similar reordering required for color matrices: 3V x float4

- 16-bit uses short4, 64-bit uses double2

Dslash performance



Krylov Solver Implementation

- Complete solver **must** be on GPU
  - Transfer b to GPU
  - Solve Mx=b
  - Transfer x to CPU
- Require BLAS level 1 type operations
  - AXPY operations: b += ax
  - NORM operations: c = (b,b)
- Uses mixed precision for high accuracy and high speed
  - Double-Half or Single-Half solvers usually used

while ($|r_k| > \varepsilon$) {

$\beta_k = (\mathbf{r}_k, \mathbf{r}_k)/(\mathbf{r}_{k-1}, \mathbf{r}_{k-1})$

$\mathbf{p}_{k+1} = \mathbf{r}_k - \beta_k \mathbf{p}_k$

$\alpha = (\mathbf{r}_k, \mathbf{r}_k)/(\mathbf{p}_{k+1}, A\mathbf{p}_{k+1})$

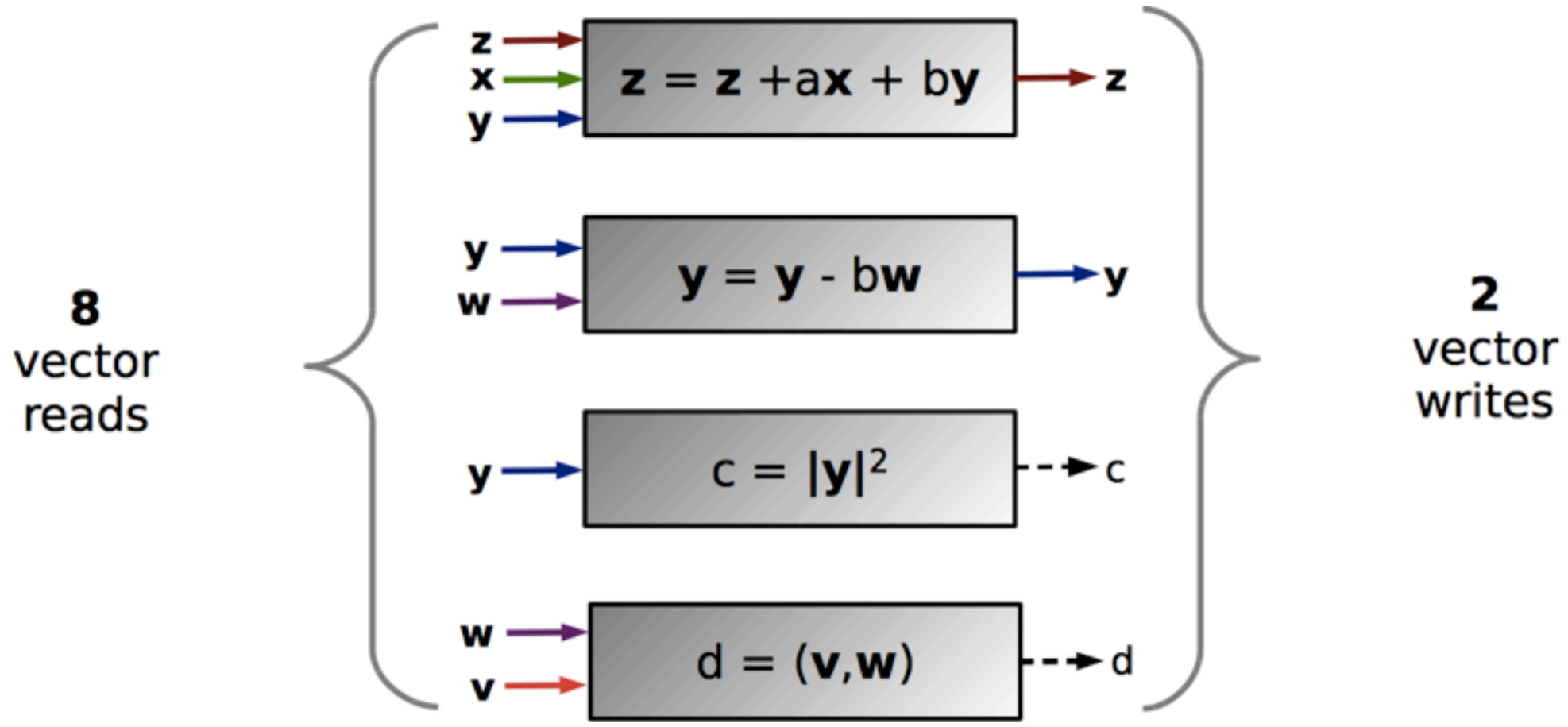
$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha A\mathbf{p}_{k+1}$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{p}_{k+1}$

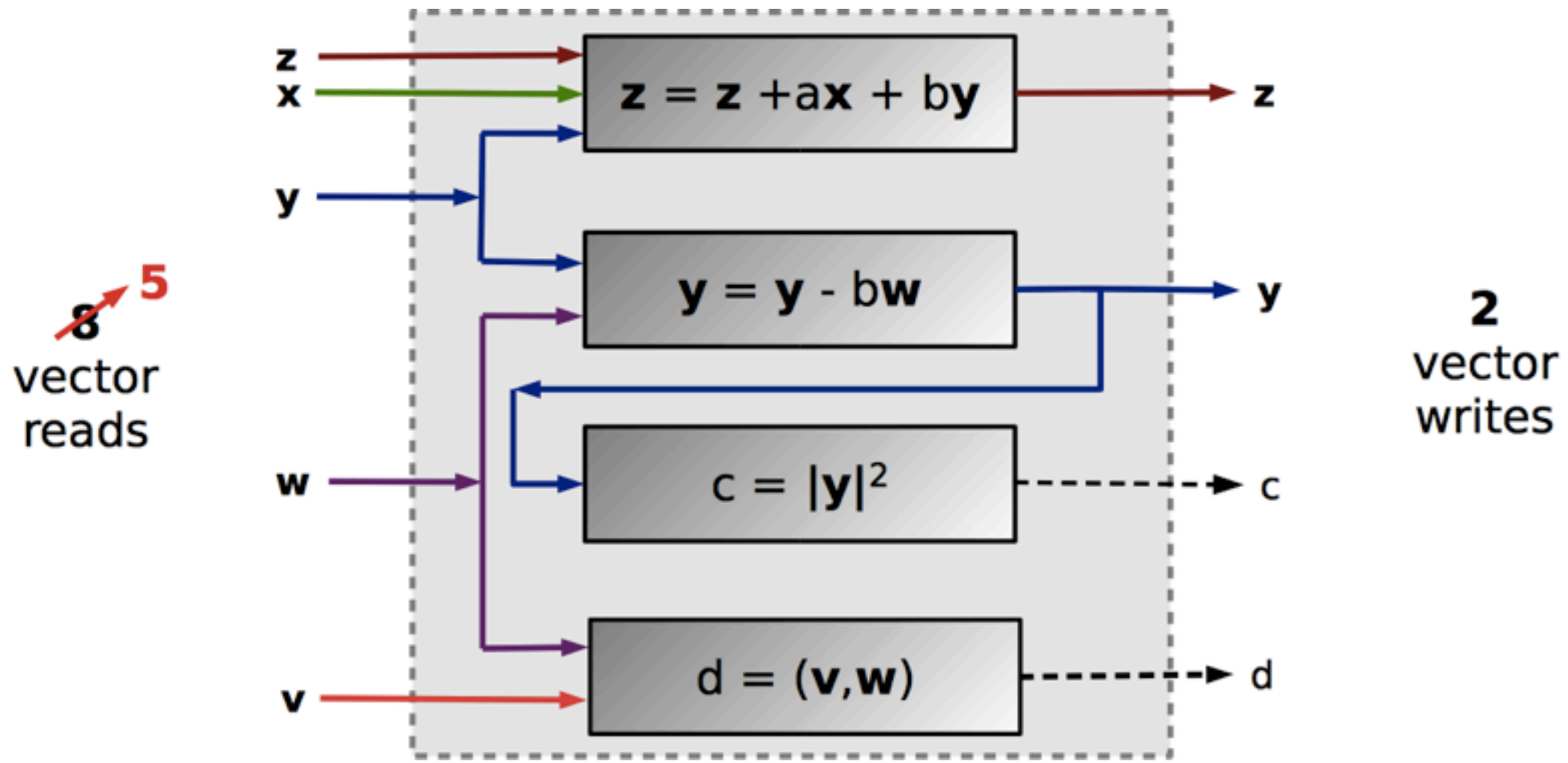
$k = k+1$

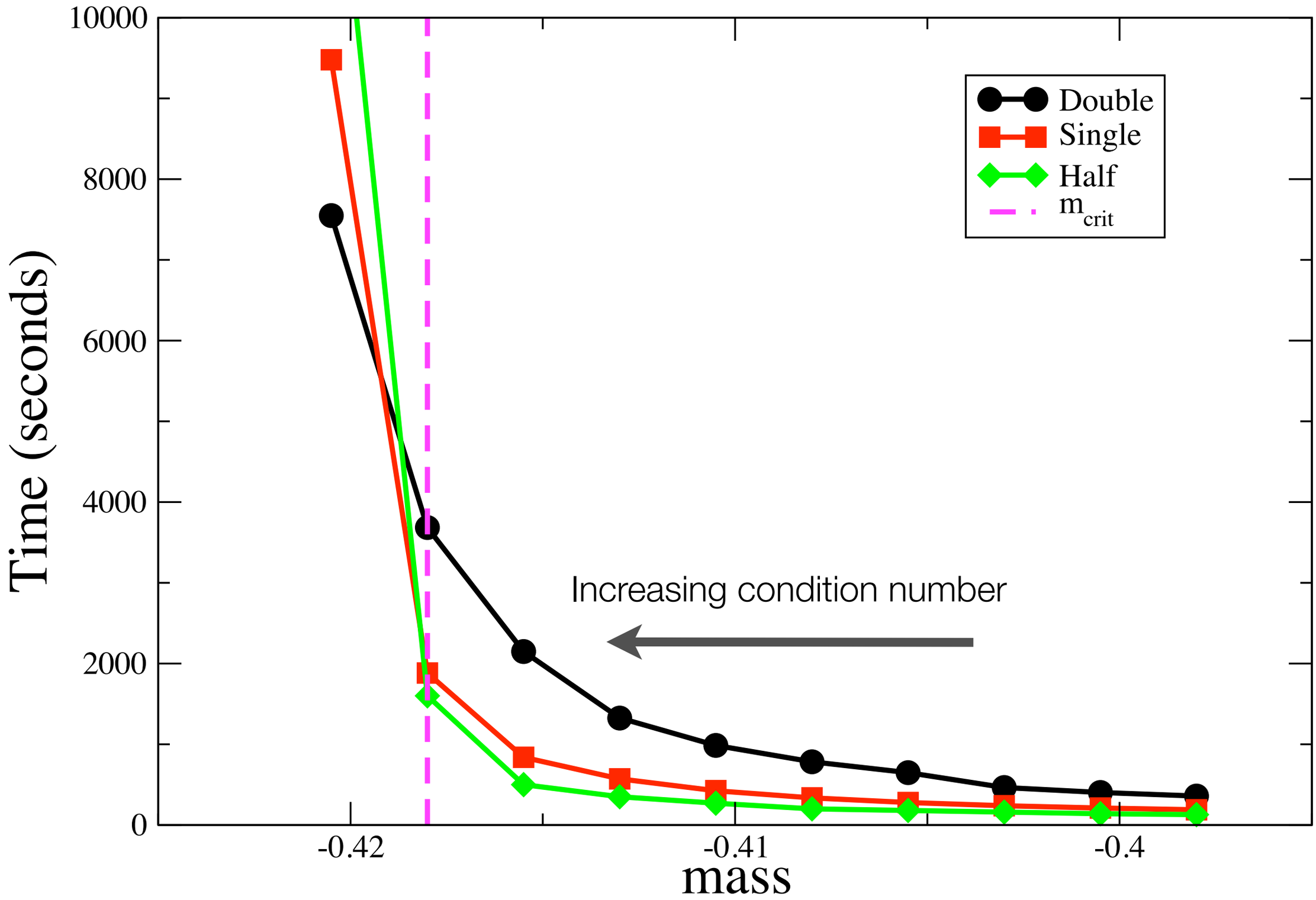
}

Optimizing the Solver: Kernel Fusion



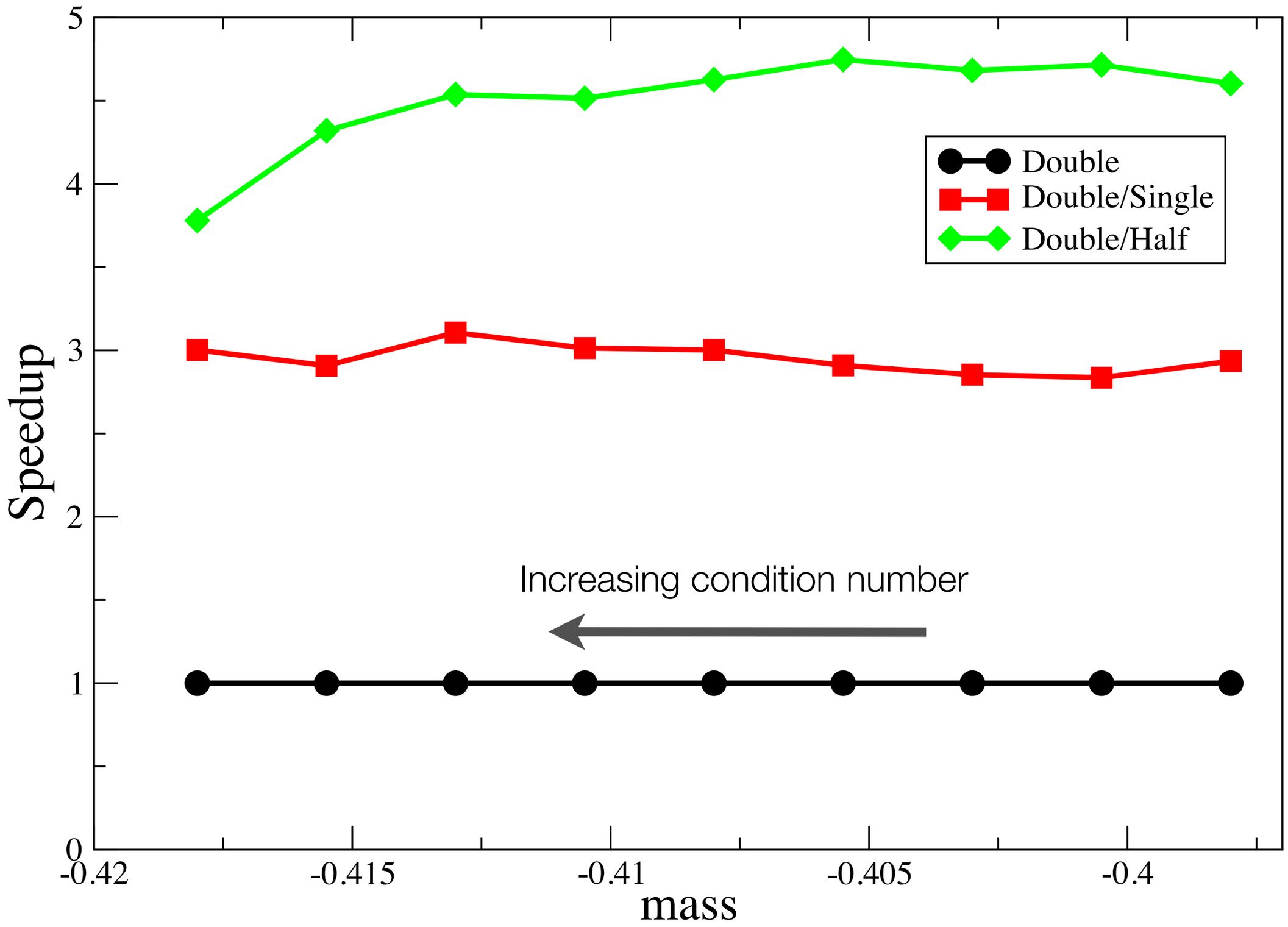
Optimizing the Solver: Kernel Fusion





Inverter Time to Solution

($\varepsilon=10^{-8}$, $V=32^3x96$, GTX 280)

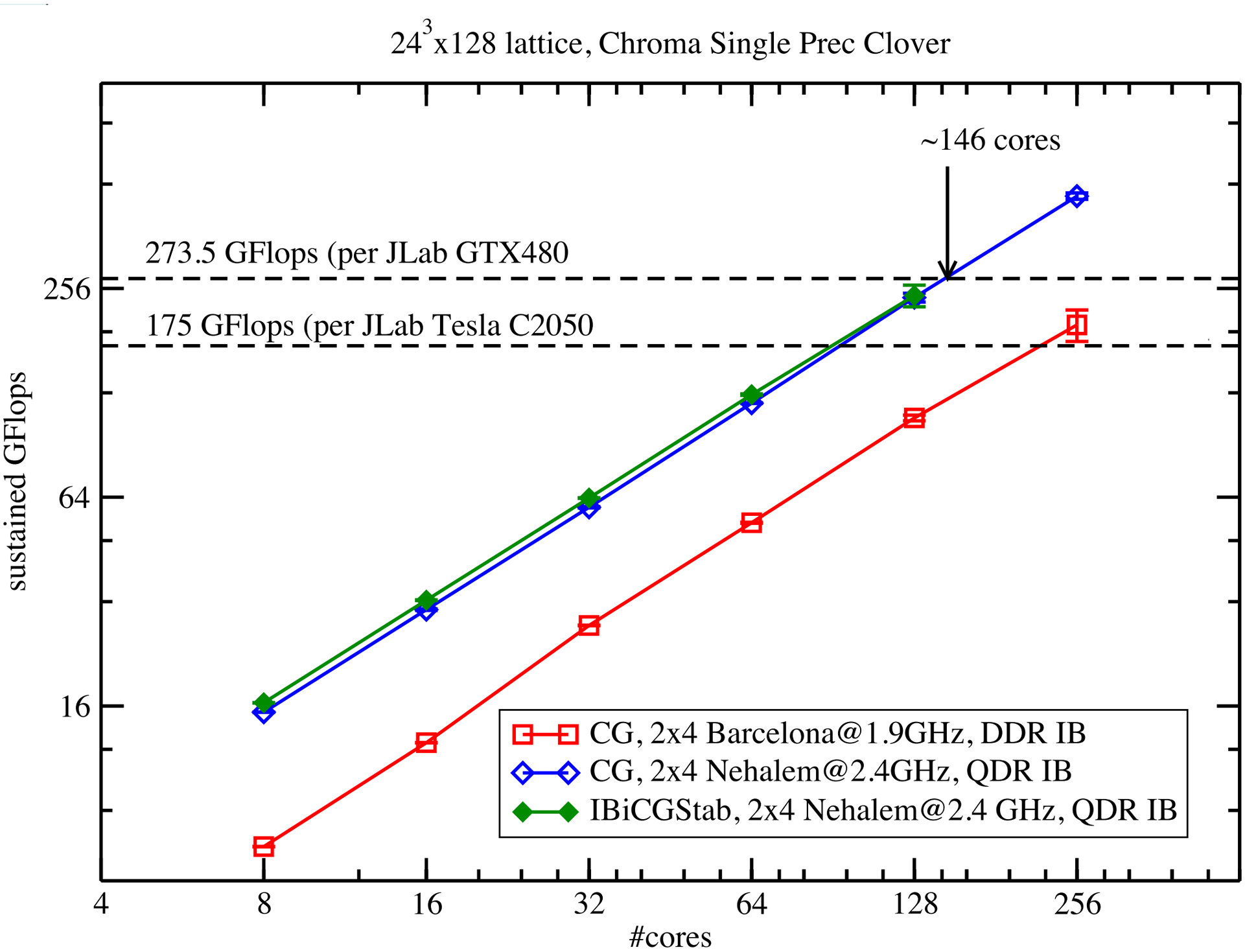


- 16-bit Mat-Vec 400 GFLOPS
- Solver 300-320 GFLOPS

Inverter Speedup

($\varepsilon=10^{-8}$, $V=32^3 \times 96$, GTX 280)

GPUs vs. CPUs


24^3x128 lattice, Chroma Single Prec Clover
~146 cores
273.5 GFlops (per JLab GTX480
175 GFlops (per JLab Tesla C2050
sustained GFlops
256
64
16
4
8
16
32
64
128
256
#cores
CG, 2x4 Barcelona@1.9GHz, DDR IB
CG, 2x4 Nehalem@2.4GHz, QDR IB
IBiCGStab, 2x4 Nehalem@2.4 GHz, QDR IB

Multiple GPUs

The need for multiple GPUs

- Only yesterday's lattice volumes fit on a single GPU
- More cost effective to build multi-GPU nodes
    - Better use of resources if parallelized
- Gauge generation requires strong scaling
    - Can GPUs replace traditional super computers?

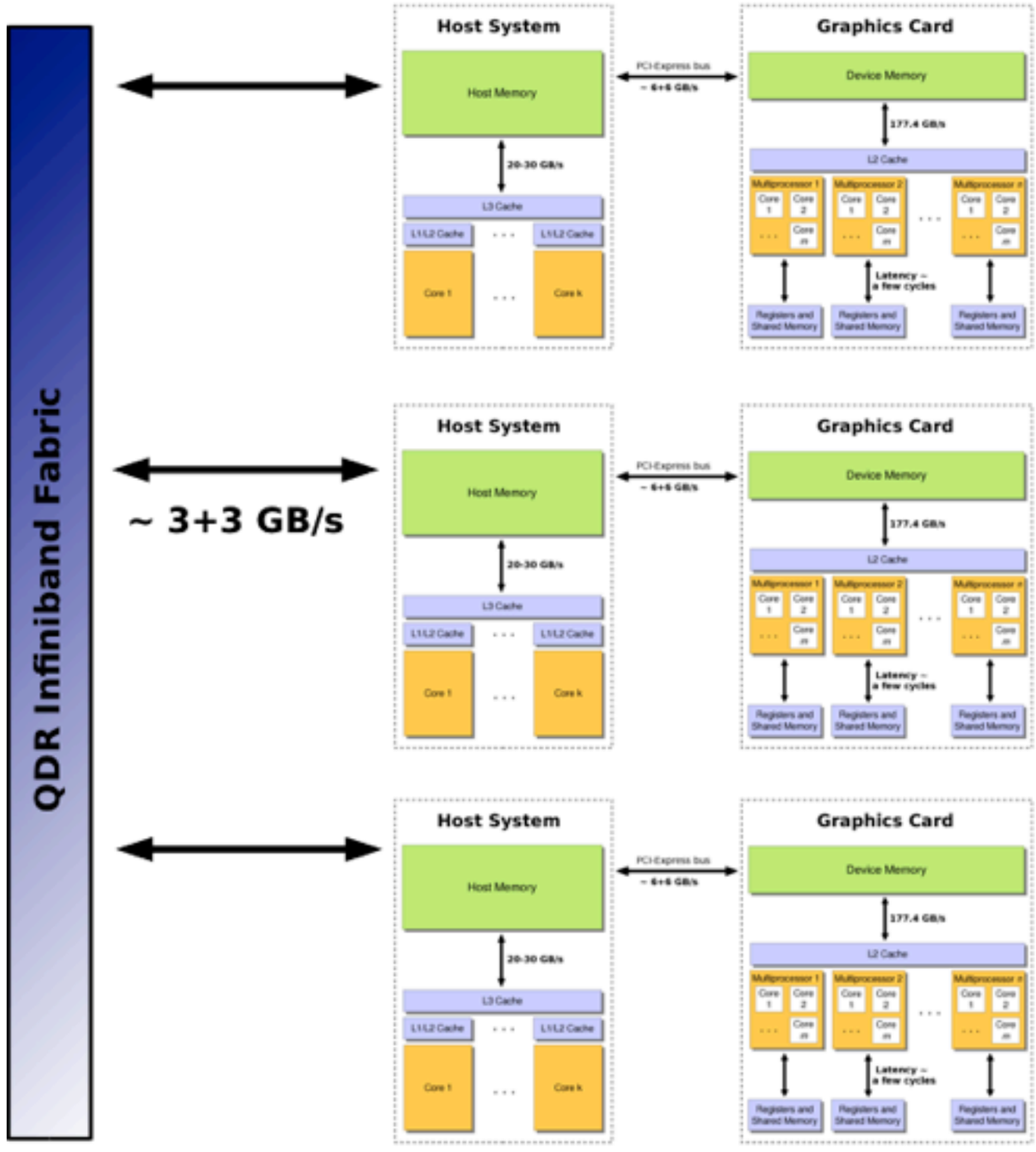
The need for multiple GPUs

- Only yesterday's lattice volumes fit on a single GPU
- More cost effective to build multi-GPU nodes
    - Better use of resources if parallelized
- Gauge generation requires strong scaling
    - Can GPUs replace traditional super computers?

The need for multiple GPUs

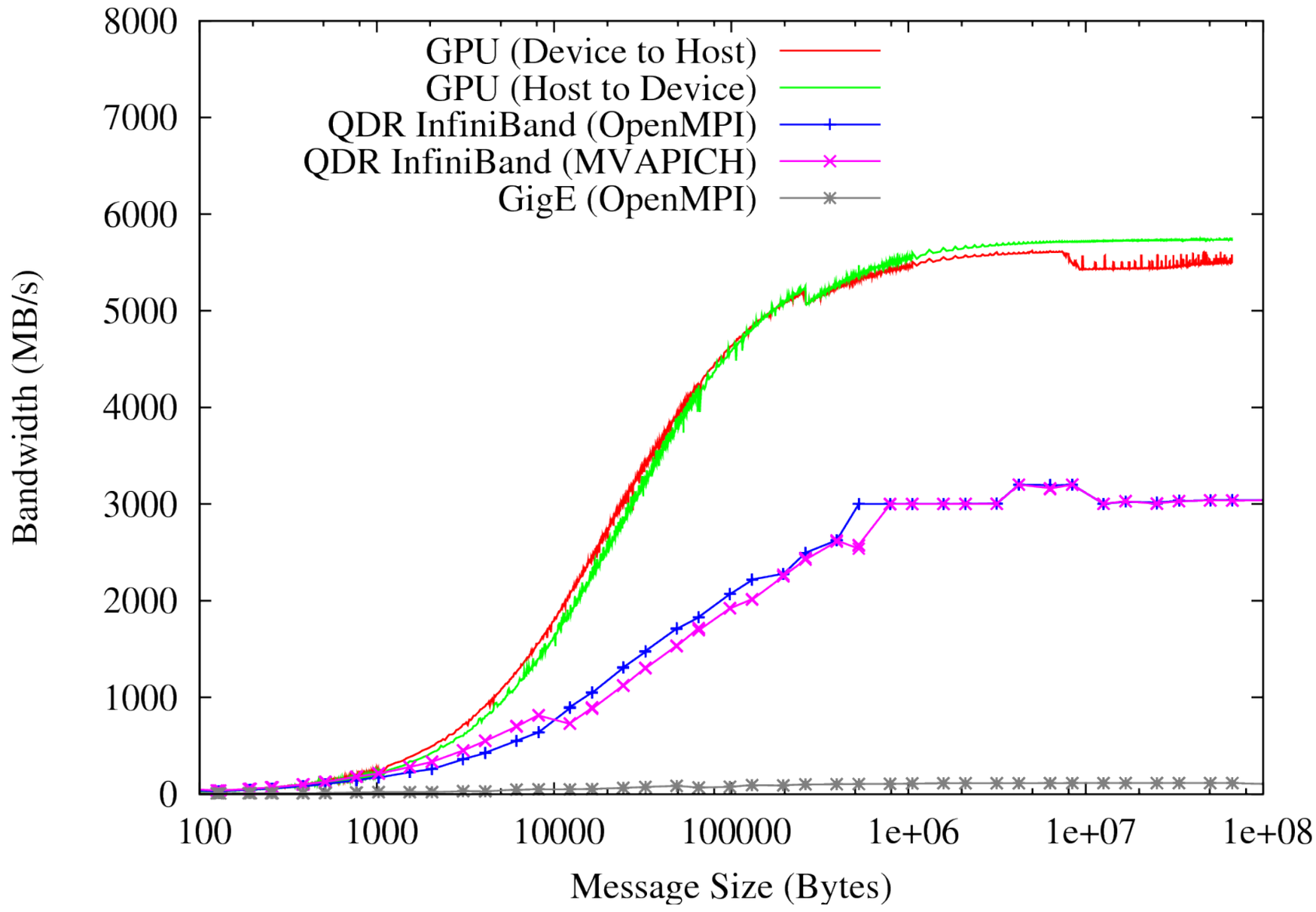
- Only yesterday's lattice volumes fit on a single GPU
- More cost effective to build multi-GPU nodes
    - Better use of resources if parallelized
- Gauge generation requires strong scaling
    - Can GPUs replace traditional super computers?

Multiple GPU Memory Hierarchy



Modeling the multi-GPU performance

- Given the bandwidth of the inter-connects,



Modeling the multi-GPU performance

- the amount of the data to be sent

$$(1320 \text{ flops/site}) \times (L^4/2 \text{ flops}) = 660L^4 \text{ flops}$$

$$(24/2 \times 4 \text{ bytes/boundary site}) \times (8L^3/2 \text{ sites}) = 192L^3 \text{ bytes}$$

$$\frac{660L^4}{\text{Perf}} = \frac{192L^3}{\text{Bandwidth}}$$

$$\text{Bandwidth [MB/s]} = \frac{0.29(\text{Perf [Mflop/s]})}{L}$$

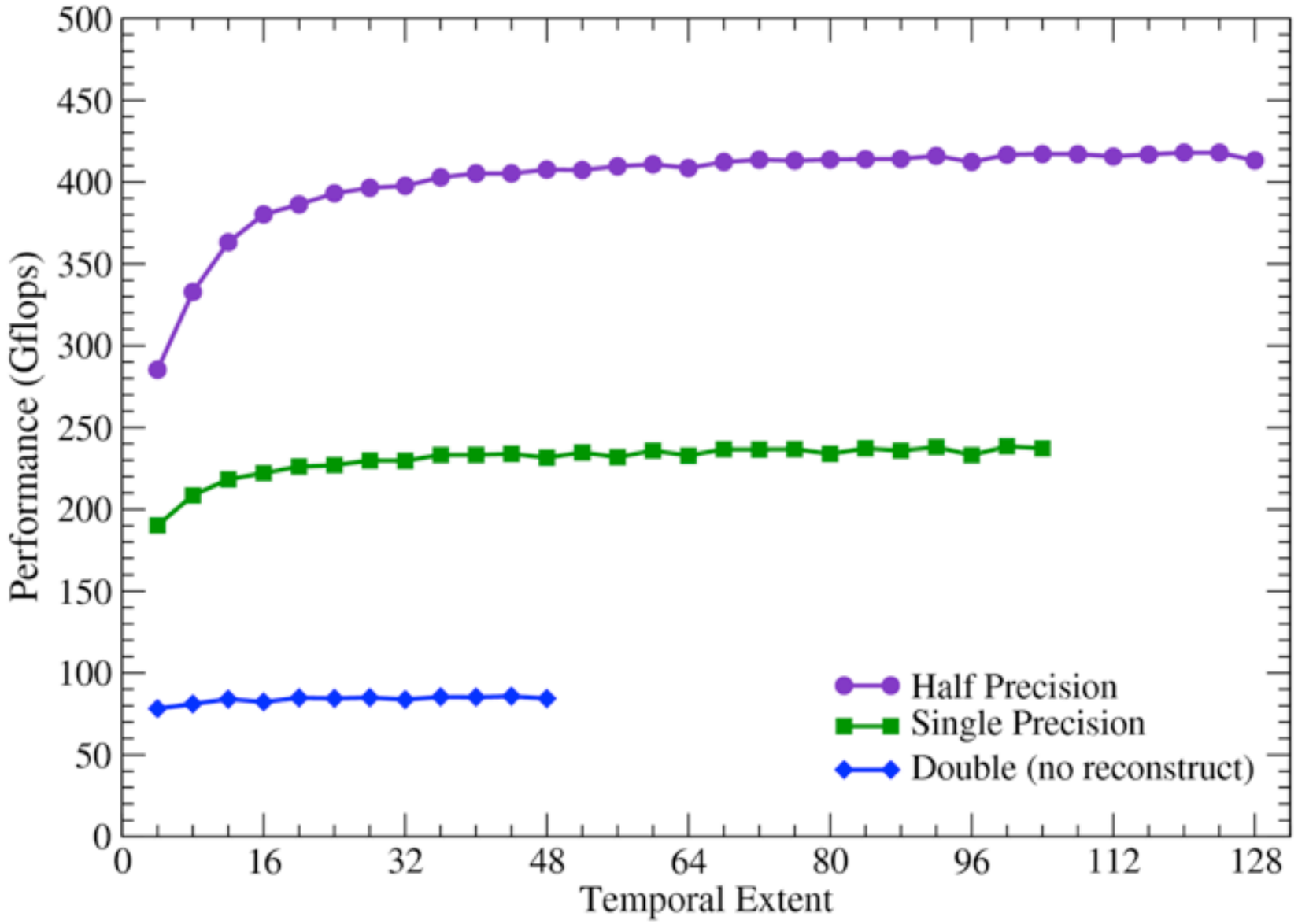
$$\text{MessageSize [Bytes]} = 24L^3$$

Inspired by Gottlieb (2000), http://physics.indiana.edu/~sg/pcnets/

via Holmgren (2004), arXiv:hep-lat/0410049

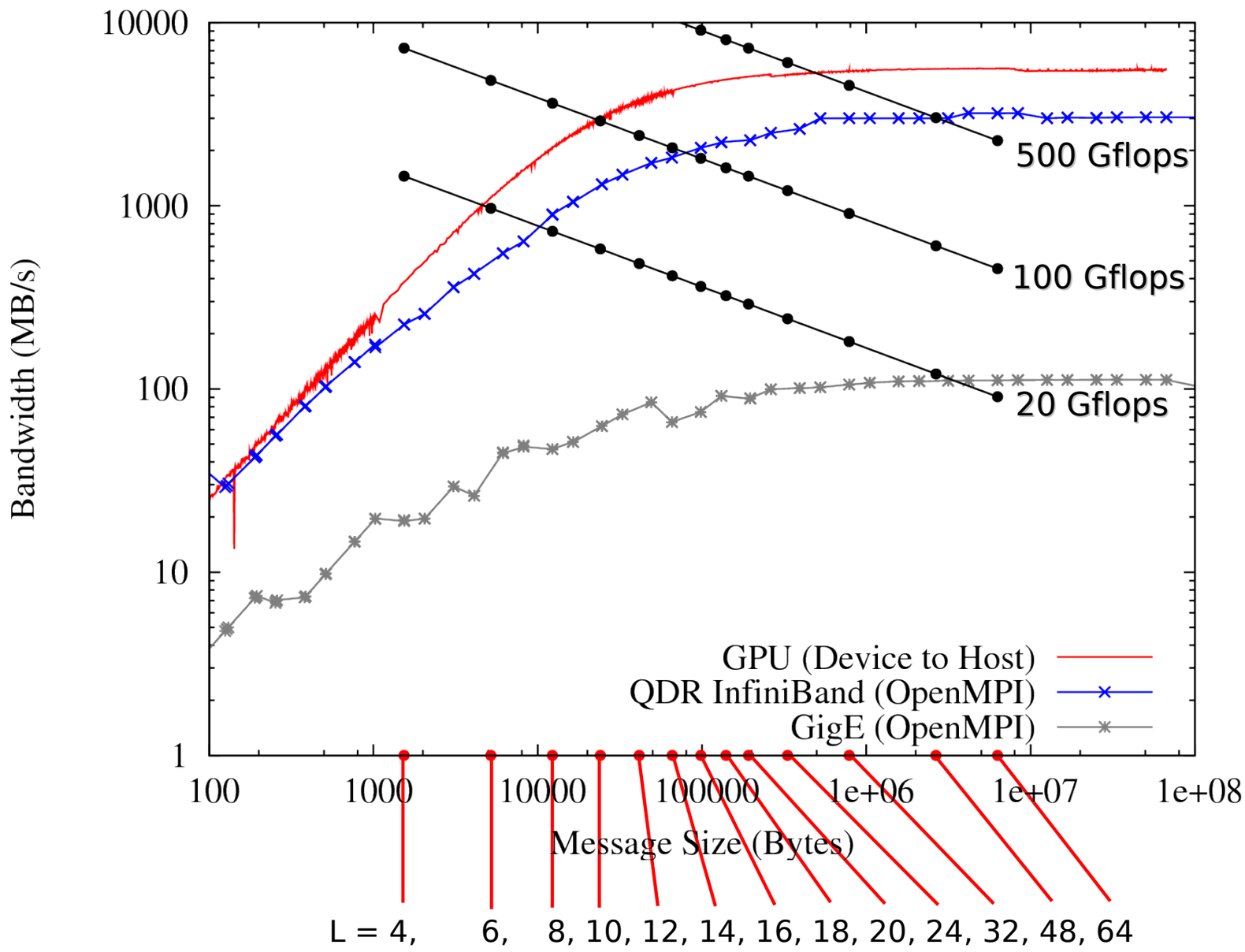
Modeling the multi-GPU performance

- and the rate of computation



Modeling the multi-GPU performance

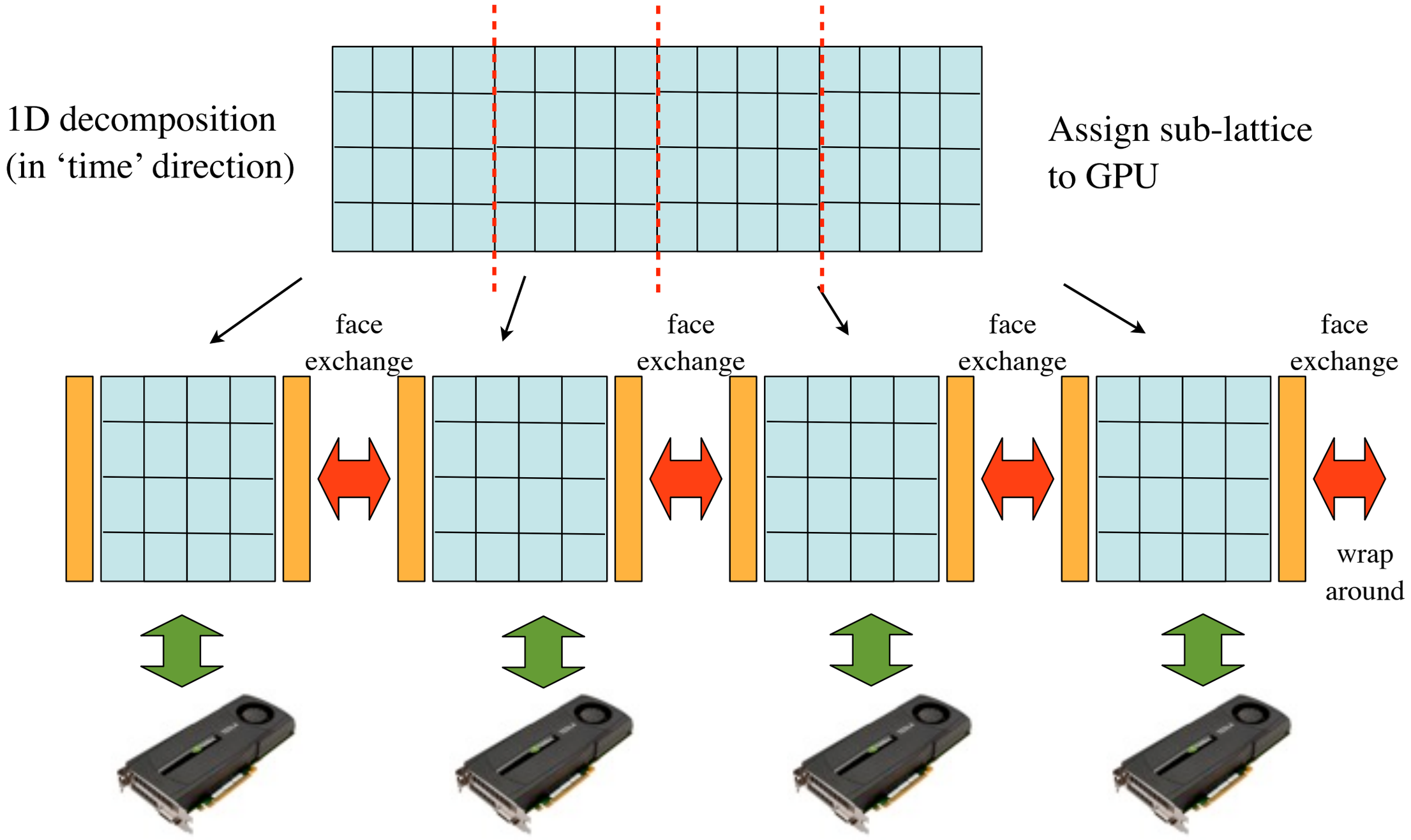
- we can model the achievable performance



Initial Multiple GPU Implementation

- Consider highly asymmetric lattices
  - 24^3128, 32^3256
- Parallelize over temporal dimension (slowest running)
- Volume V partitioned into N sub-volumes of $V/N = V_s T/N$
- MPI employed for CPU parallelization

Lattice decomposition

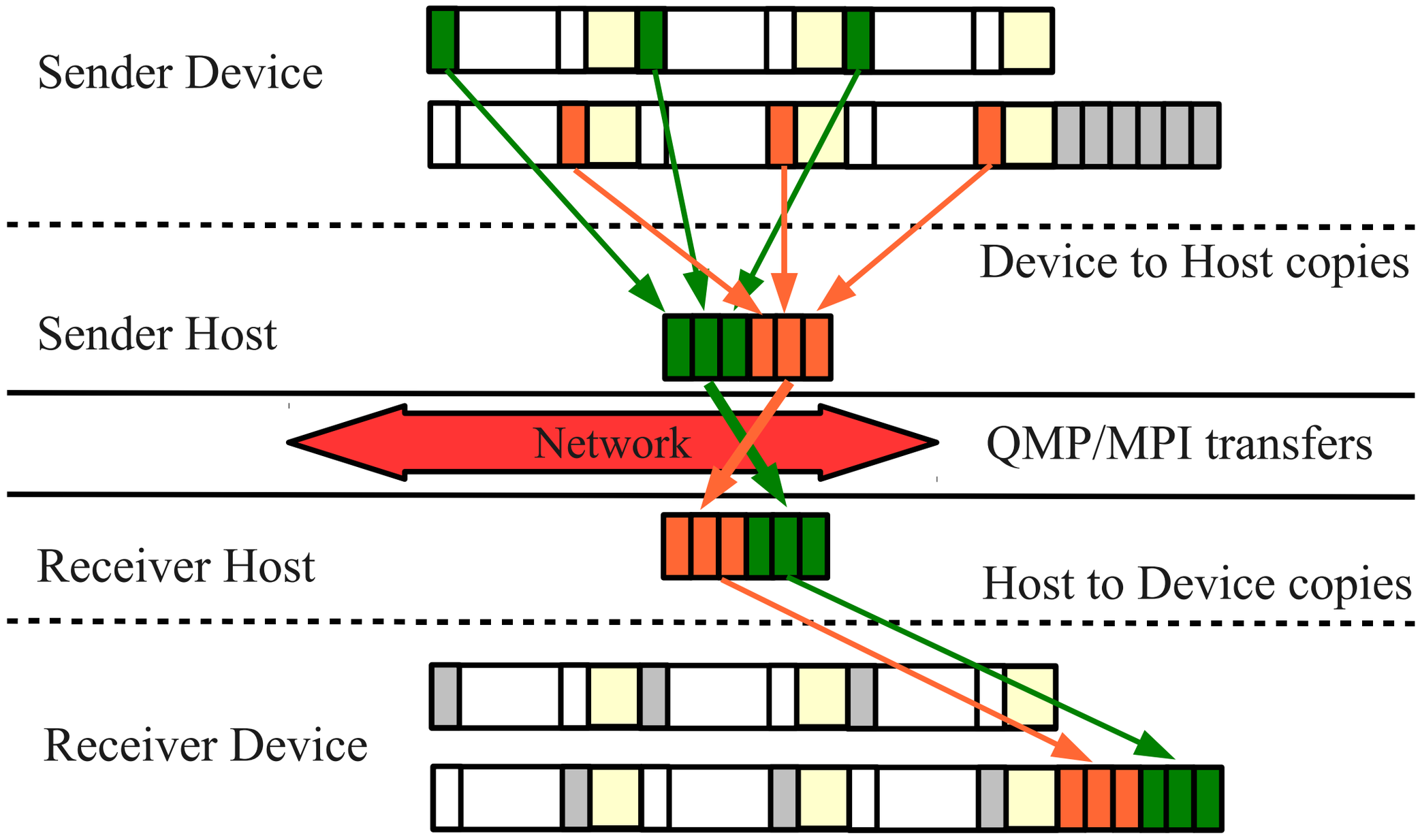


Spinor Field Ghost Zone

- Required communications
  - Send spinor surface $t=0$ surface to GPU $i-1$
  - Send spinor surface $t=T-1$ surface to GPU $i+1$
- Each (spin projected) surface has $12V_s$ elements
  - Arranged in 3 contiguous float4 arrays
  - Copy arrays directly to host from device
- Store ghost zone at end of spinor field
  - Enlarge spinor field to 6 V * float4 by 24 V_s
  - Easy to ignore end zone for reductions

$$P^{+4} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad P^{-4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

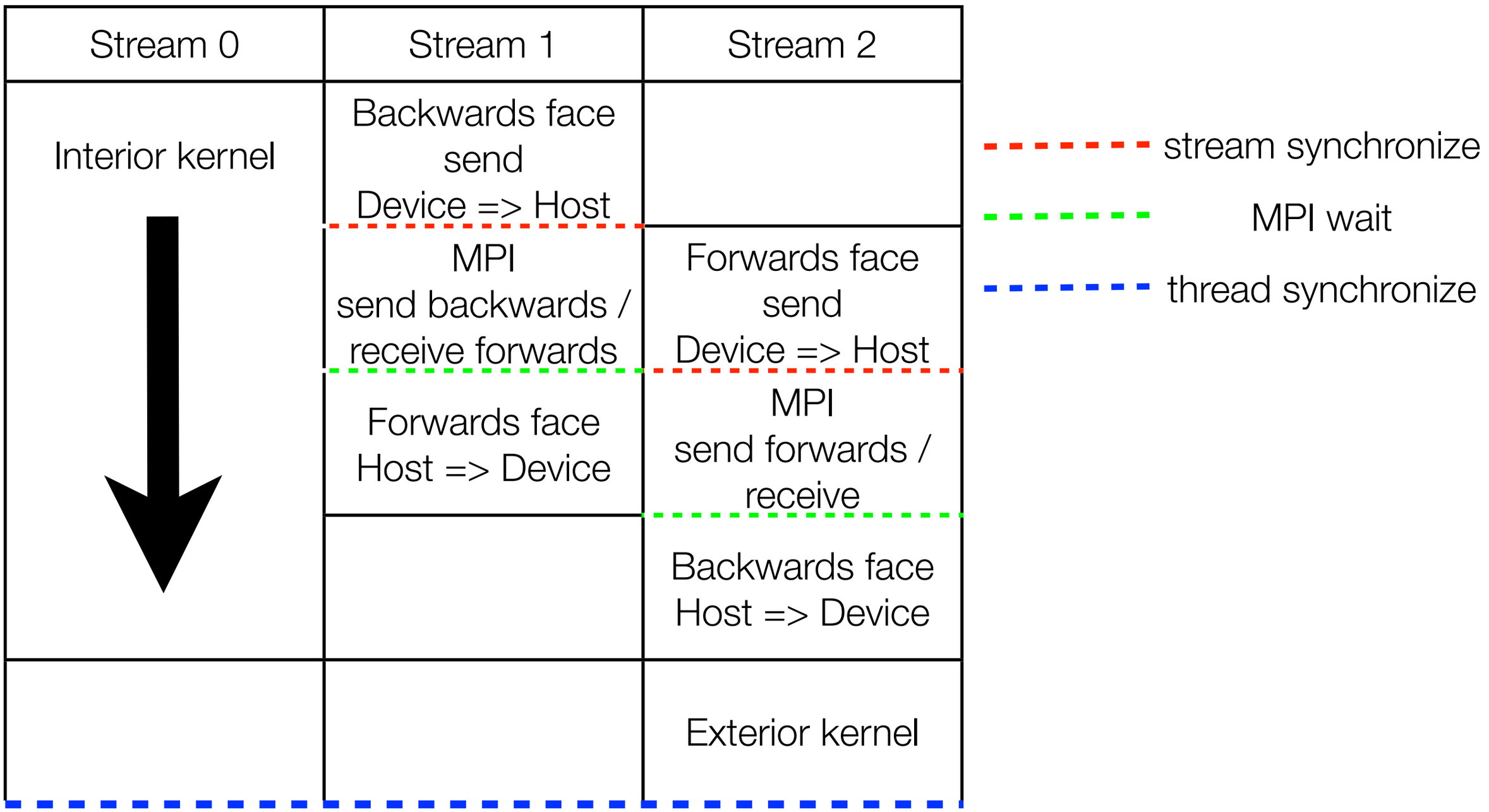
Spinor Ghost Communication



Hiding the communication

- Non-overlapping comms
    - Perform all communications
    - Computation done as a single kernel
- Overlapping comms and compute
    - Use CUDA asynchronous API for host <=> device copies
    - Non-blocking MPI communication between processes
    - Separate kernel for interior and surfaces

Overlapping Pipeline



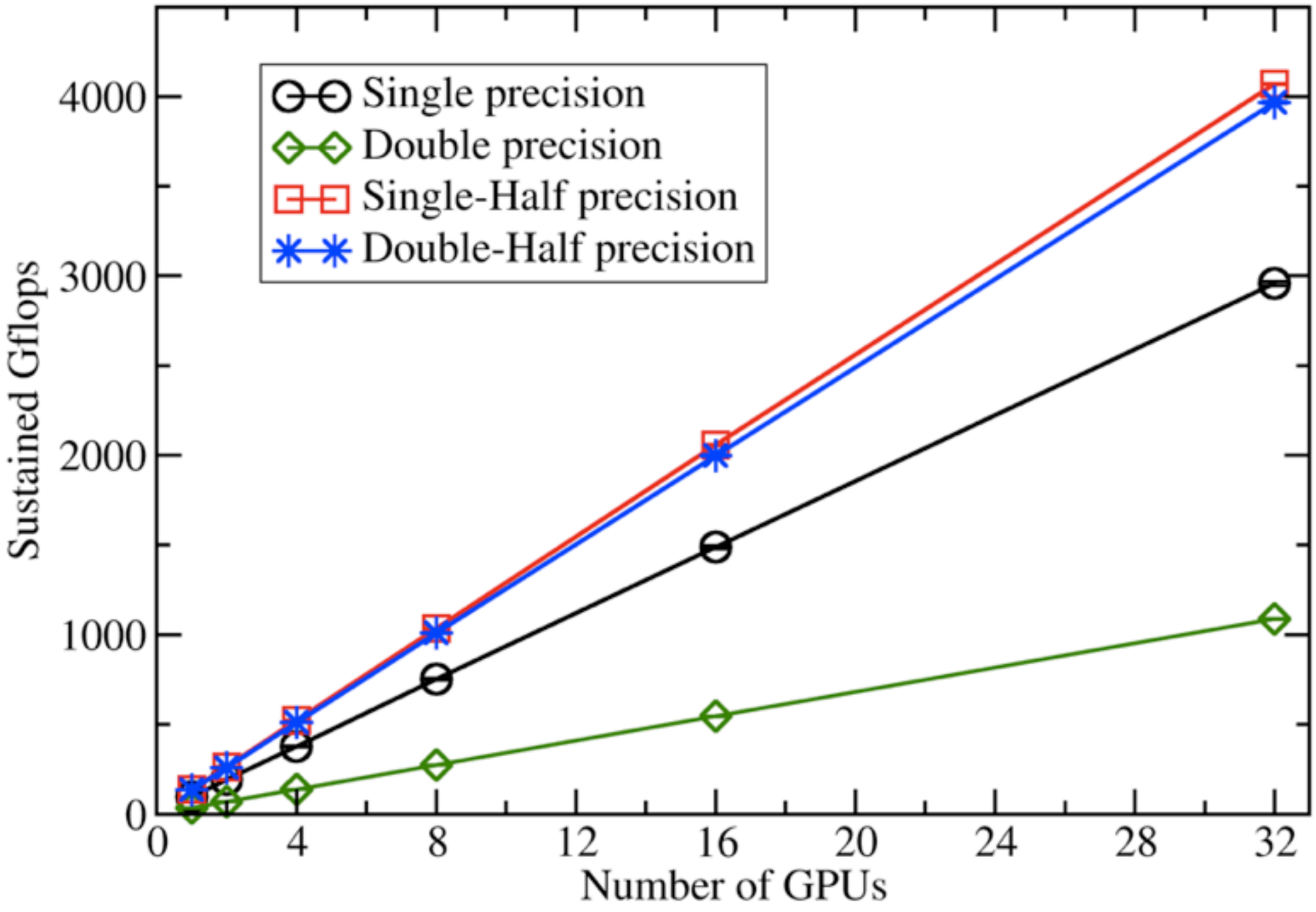
Multi-GPU performance

- Wilson-clover
  - BiCGstab solver
  - Even-odd preconditioning
- Benchmarks run at JLab and LLNL
- Test strong scaling at 24^3x128, 32^3x256

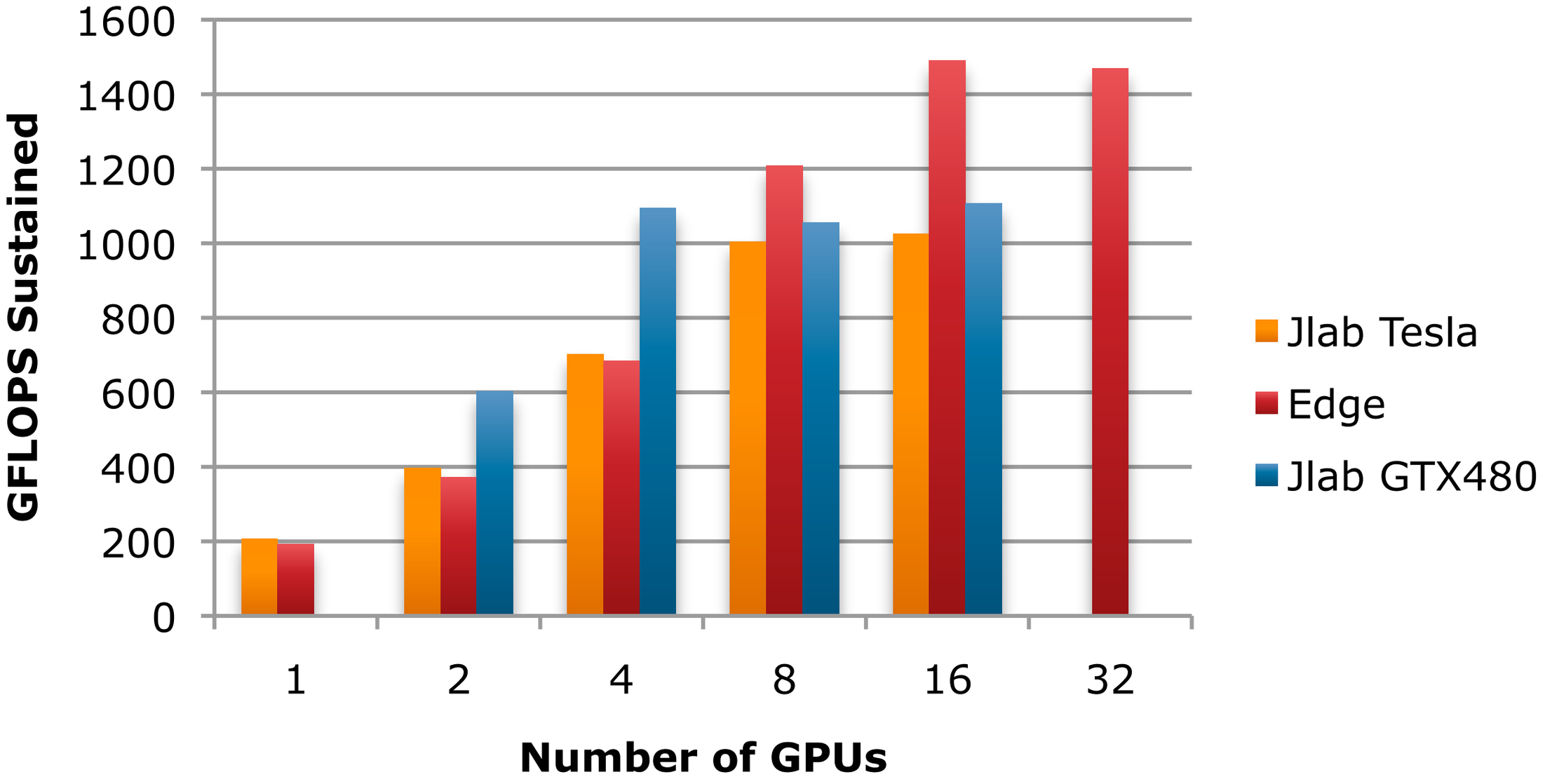
10g @ Jlab

Edge @ LLNL

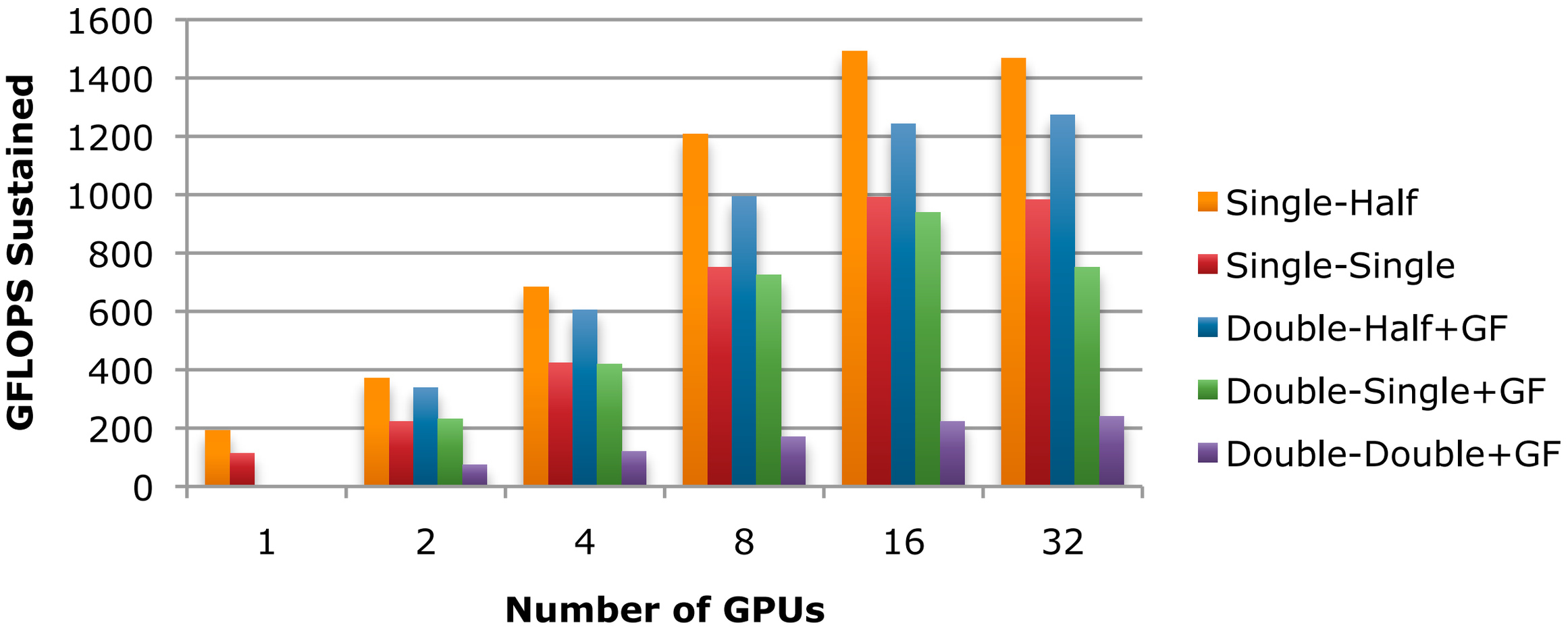
Weak scaling is no problem



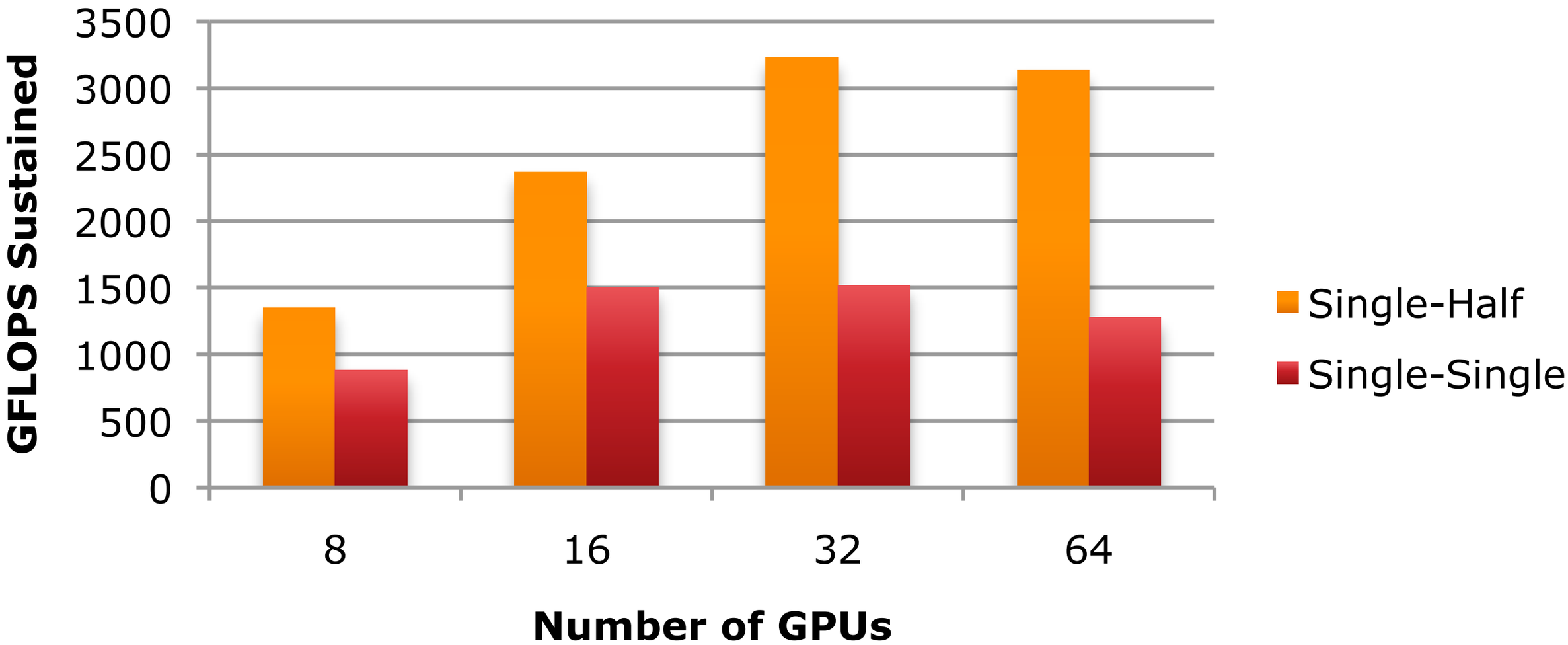
Single-Half: 24x24x24x128



Edge: 24x24x24x128



Edge: 32x32x32x256



Retrospective

- 2004: First 1 TFLOPS sustained for QCD (P. Vranas)
  - 1 rack Blue Gene/L
  - ~ $1M in 2005/2006
- 2010: 1 TFLOPS sustained, under your desk
  - Dual-socket node with 4 GPUs
  - ~ $5K

.. for problems that fit

(1 rack BG/L has 512 GB RAM
vs. 6 GB for 4 C2050s)

Vector Isoscalar (I=0) Spectrum

light-strange (ls) basis

$$\mathcal{O}_l^{\Gamma} = \frac{1}{\sqrt{2}}\left(\bar{u}\Gamma u + \bar{d}\Gamma d\right)$$

$$\mathcal{O}_s^{\Gamma} = \bar{s}\Gamma s$$

I = 0: Must include all disconnected diagrams

- 'Distillation' technique, 16^3x128 lattice
- N_{ev} x N_s x N_t x #quark x #cfg solves
  - N_{ev}=64, N_s=4, N_t=128, #quark=2, #cfg=479
  - 31 Million solves.
  - for 24^3x128, will need N_{ev}=128 - 162
  - Too expensive to do this without GPUs
- omega + 7 excited states, ~1% statistical error.

N_f=2+1, m_{π} ~ 400MeV, L~2fm

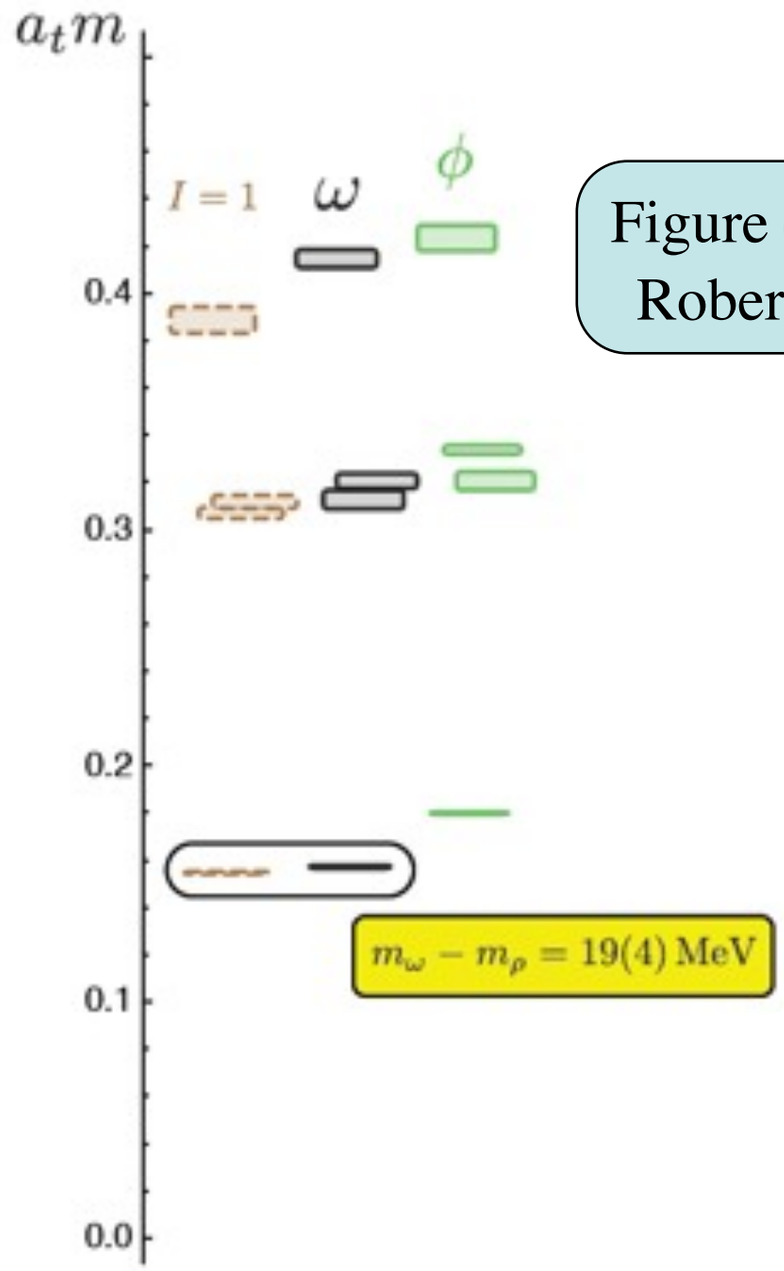


Figure courtesy of Robert Edwards

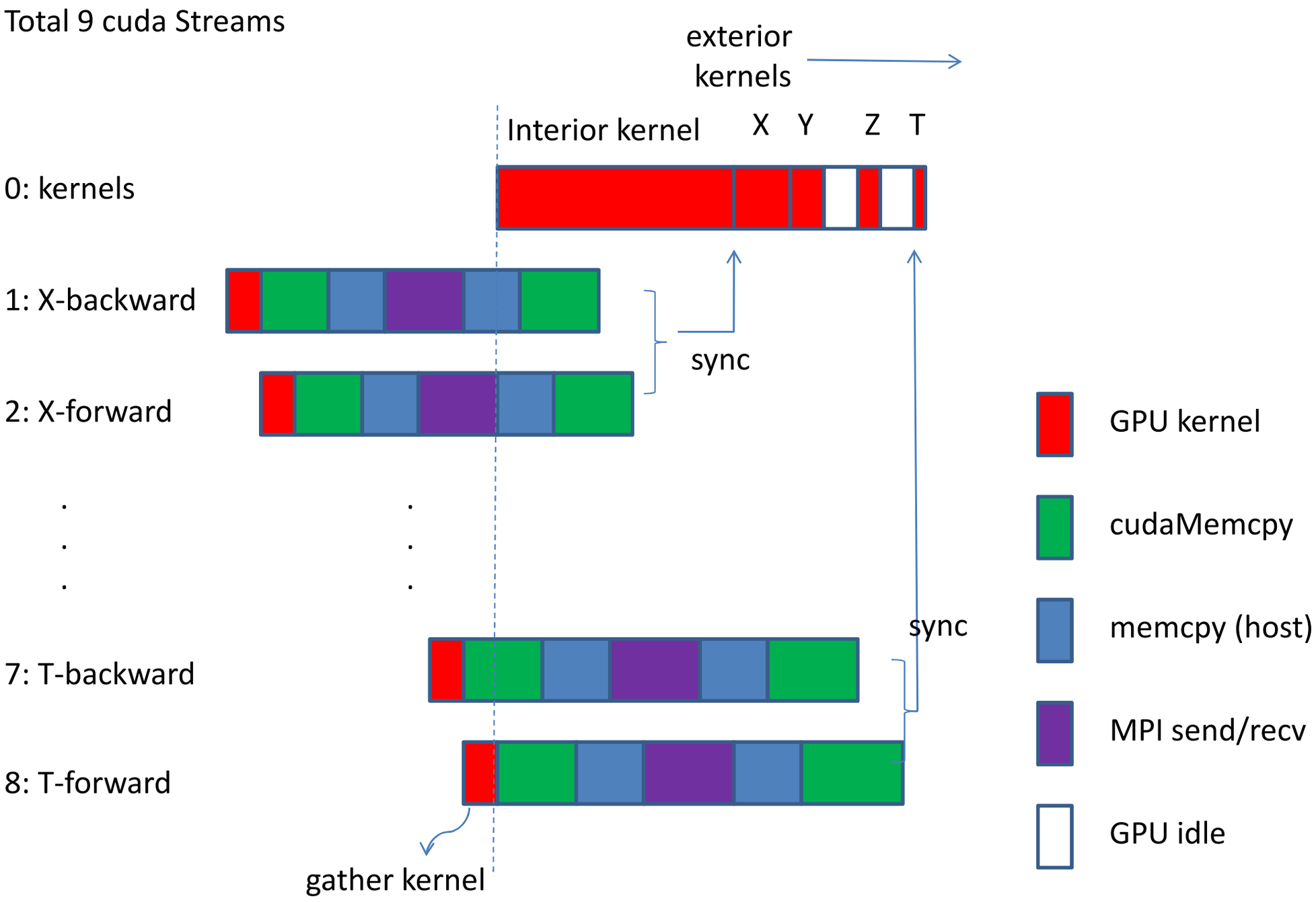
Failure at strong scaling

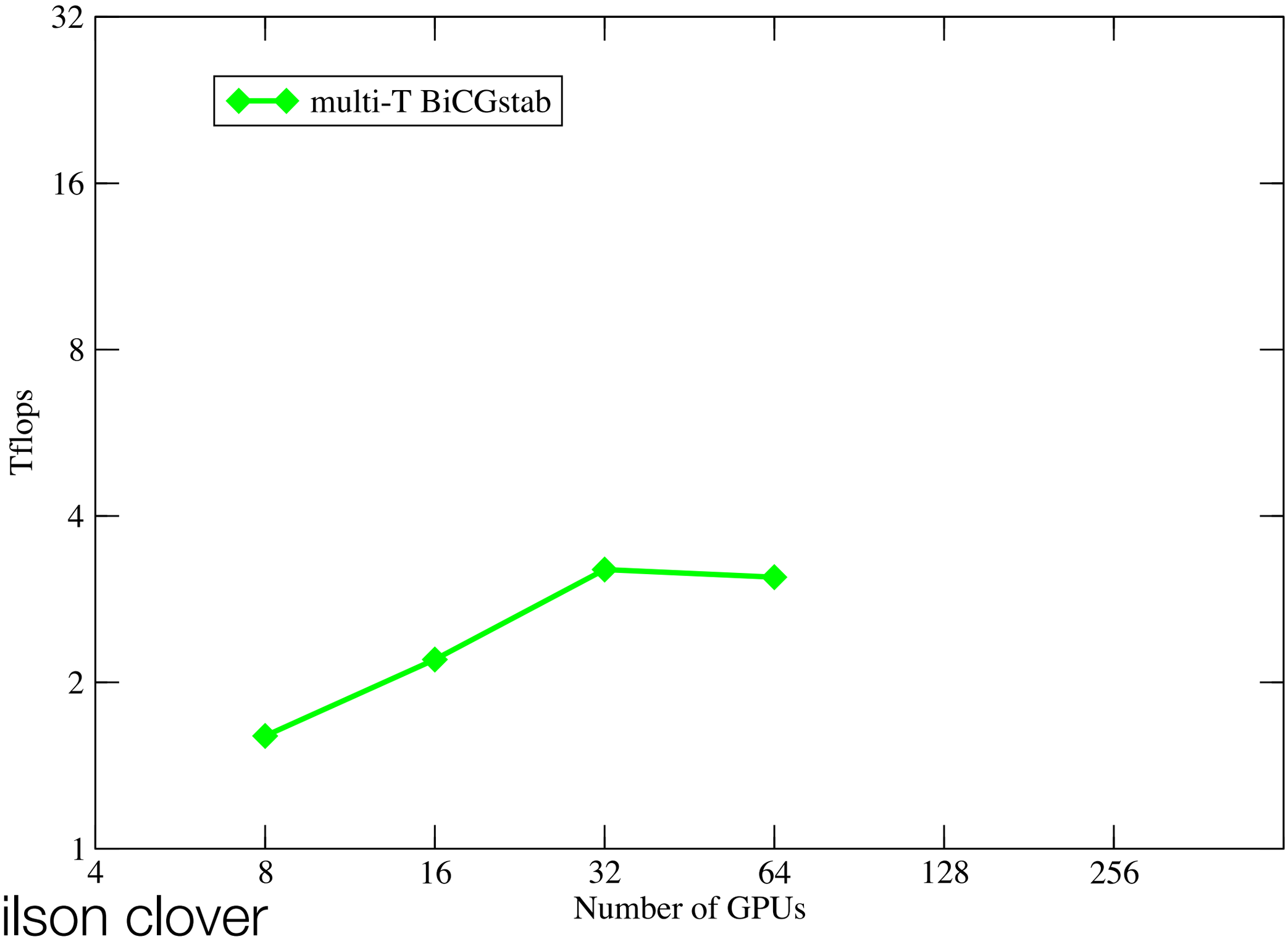
- Timing breakdown of pipeline

# of nodes		2	4	8	16
Lattice size per node		28^3x48	28^3x24	28^3x12	28^3x6
Communi-cation time breakdown (ms)	device to host	0.73	0.78	0.72	0.72
	memcpy 1	1.17	0.8	0.88	0.9
	MPI_Send	1.33	1.3	1.36	1.33
	memcpy 2	1.16	0.78	0.73	0.9
	host to device	0.59	0.59	0.58	0.58
	total	4.98	4.25	4.27	4.43
Interior kernel (ms)		16.23	8.04	3.89	1.84
Exterior kernel (ms)		0.85	0.60	0.52	0.55
Overall Dslash time (ms)		17.12	8.68	4.92	5.08

- Strong scaling limited at 16-32 GPUs
- Large symmetric lattices not possible due to memory limitations

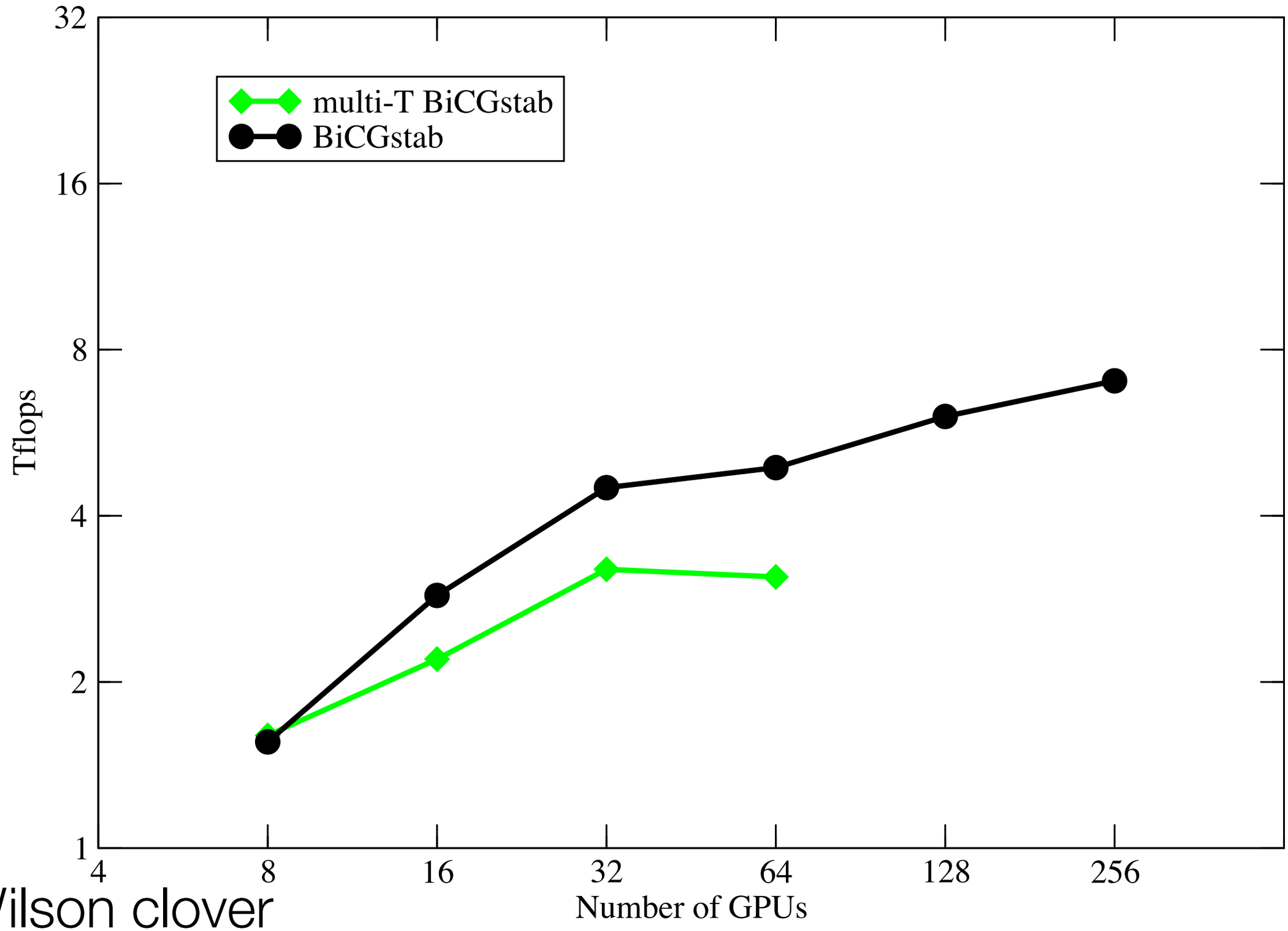
Multi-dimensional Communications Pipeline





Wilson clover

$32^3 x 256$

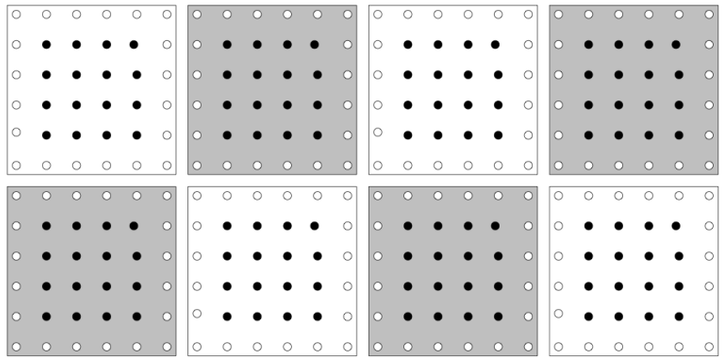


Wilson clover

$32^3 x 256$

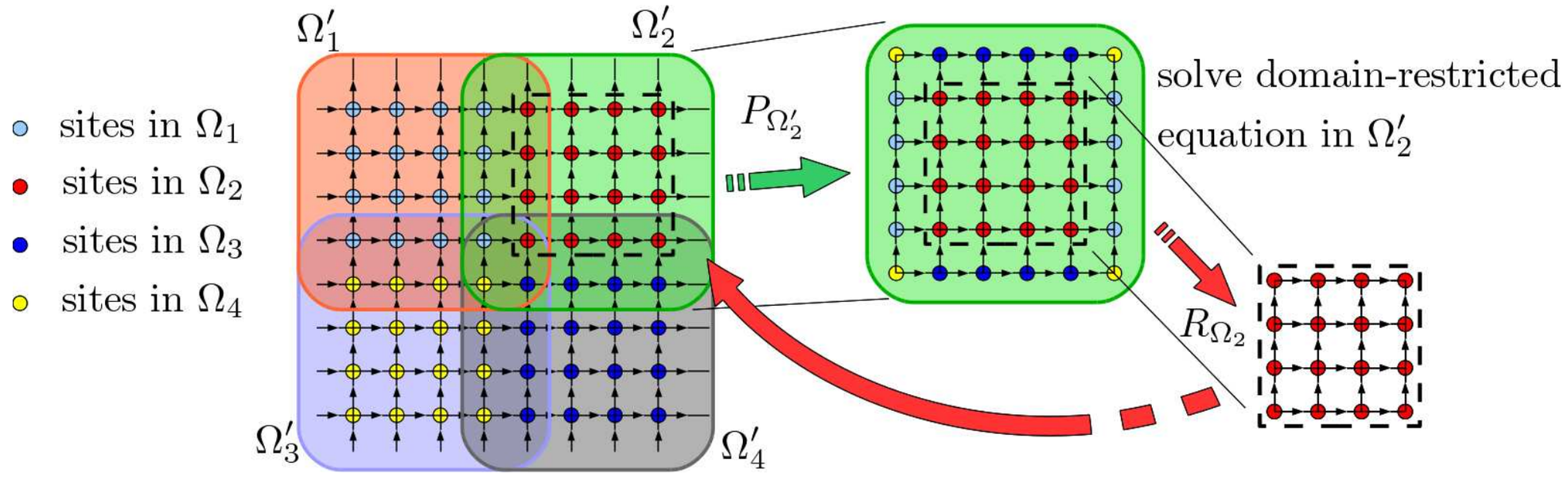
Domain Decomposition

- Only way to get to O(100+) GPUs is with communication reducing algorithms
- Schwarz Methods aka Domain Decomposition
  - Partition the lattice into domains
  - Solve each of the domains independently
  - Use domain solutions to correct full system



- Two common variants (determines input domain residual and correction scheme)
  - Multiplicative Schwarz (e.g., Schwarz Alternating Method, Lüscher)
  - Additive Schwarz
- Typically used a preconditioner

Restricted Additive Schwarz (Cai and Sarkis)



- Initial domain residuals given by the restriction of full system residual
- Impose Dirichlet boundary conditions at domain boundaries
- Solve all domain systems simultaneously
- Full system correction given by the sum of the domain solutions

figure taken from Osaki and Ishikawa

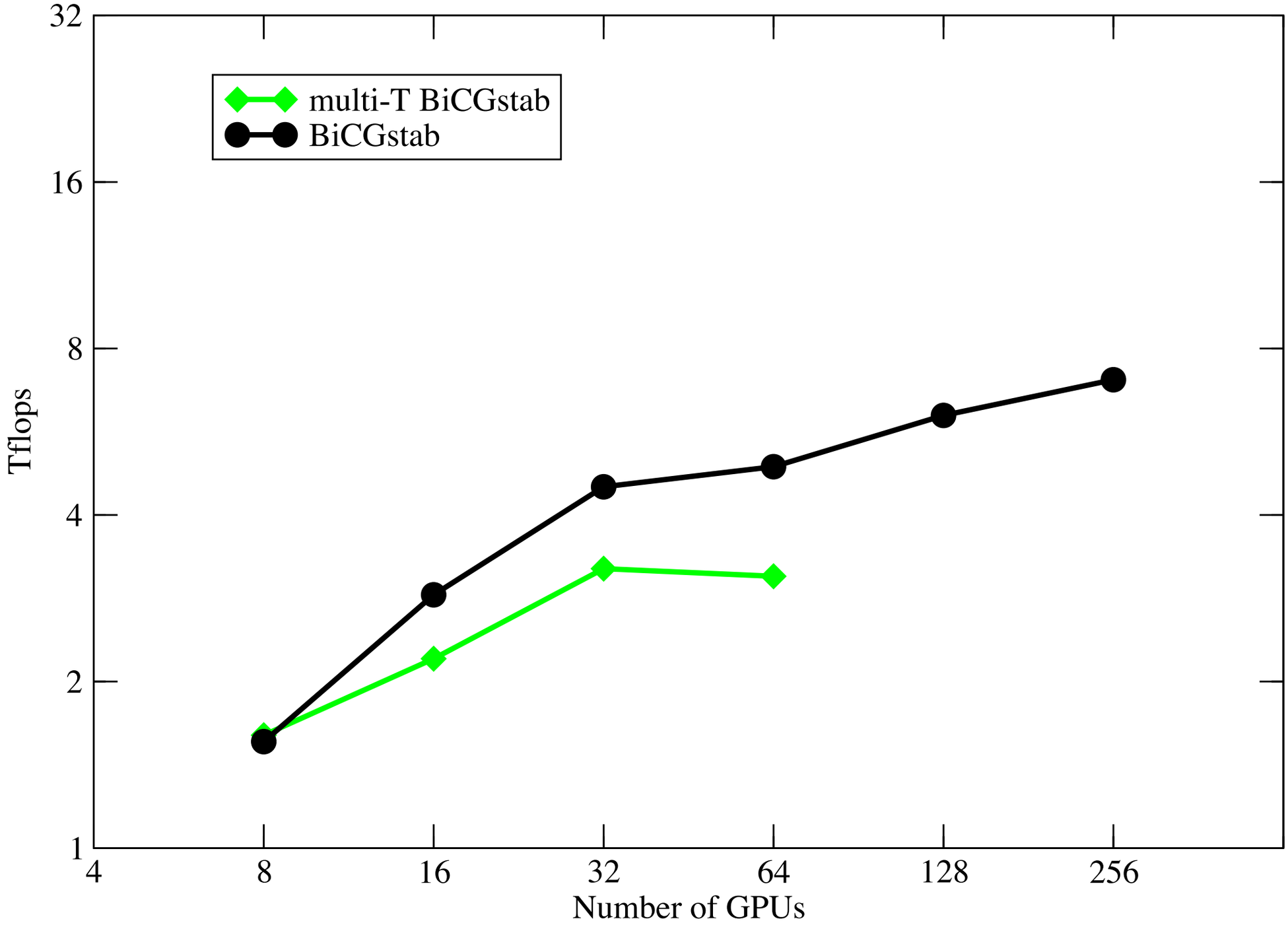
Implementing the Preconditioner

- Non-overlapping domains only
- Domain volume = local GPU volume
    - Solver iteration count varies with N_{GPU}
- Very easy to implement
    - Switch off inter-GPU communication for restricted Dirac operator
- 10 steps of Minimum residual (MR) used to solve for preconditioner
- Preconditioner is a gross approximation
    - Only need 16-bit precision

Mixed Precision GCR-DD solver

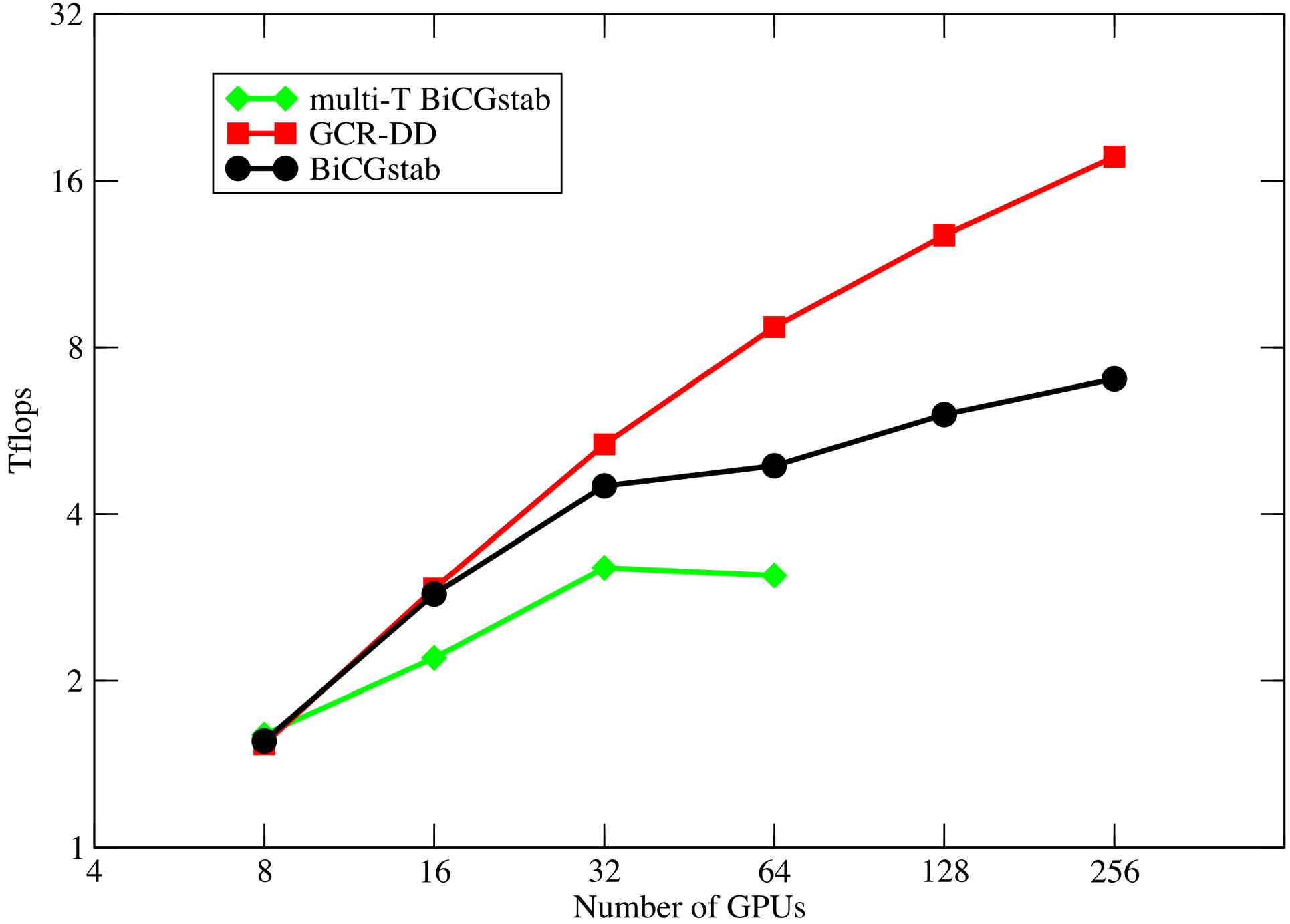
- Need a flexible solver since preconditioner is not a fixed iteration
  - Generalised Conjugate Residual (GCR)
- Memory limits size of Krylov space
  - GCR is a *restarted* solver
- Mixed Precision GCR
  - 16-bit preconditioner
  - Use 16-bit precision for orthogonalization
  - High precision restart (32 or 64 bit)

$k = 0;$
$r_0 = b - Mx_0;$
$\hat{r}_0 = r_0;$
$\hat{x} = 0;$
while $||r_0|| > tol$ **do**
  $\hat{p}_k = \hat{K}\hat{r}_k;$
  $\hat{z}_k = \hat{M}\hat{p}_k;$
  `// Orthogonalization`
  for $i \leftarrow 0$ **to** $k-2$ **do**
    $\beta_{i,k} = (\hat{z}_i, \hat{z}_k);$
    $\hat{z}_k = \hat{z}_k - \beta_{i,k}\hat{z}_i;$
  end
  $\gamma_k = ||\hat{z}_k||;$
  $z_k = z_k/\gamma_k;$
  $\alpha_k = (\hat{z}_k, \hat{r}_k);$
  $\hat{r}_{k+1} = r_k - \alpha_k\hat{z}_k;$
  `// High precision restart`
  if $k = k_{max}$ **or** $||\hat{r}_k||/||r_0|| < \delta$ **or** $||\hat{r}_k|| < tol$ **then**
    solve
    $\gamma_l\chi_l \sum_{i=l+1}^{k} \beta_{l,i}\chi_l = \alpha_l, \quad l = k, k-1, \ldots, 0, ;$
    for $i \leftarrow 0$ **to** $k-1$ **do**
      $\hat{x} = \sum_{i=0}^{k-1} \chi_i p_i$
    end
    $x = x + \hat{x};$
    $r_0 = b' - Ax ;$
    $\hat{r}_0 = r_0;$
    $\hat{x} = 0;$
    $k = 0;$
  end
end



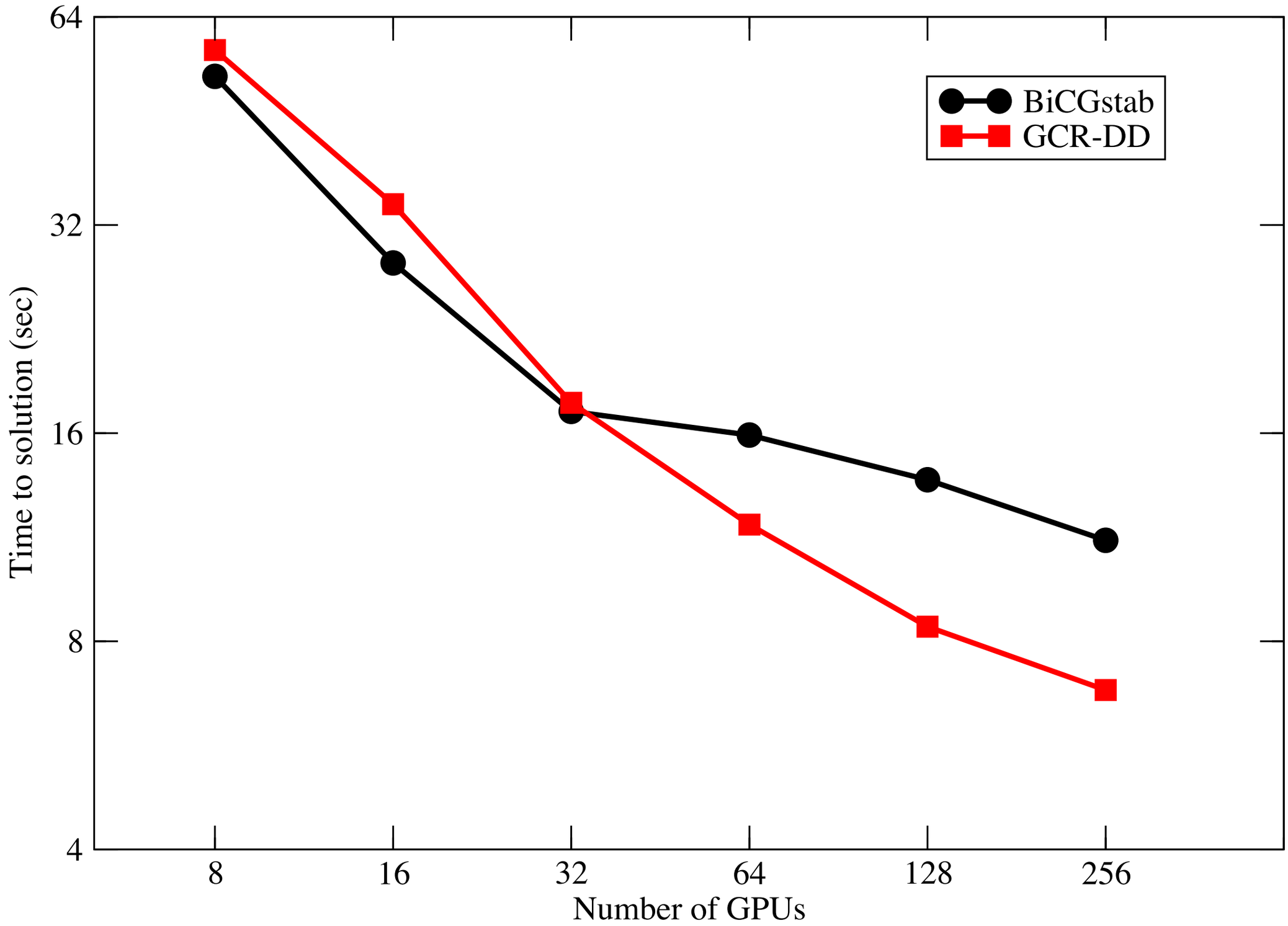
Wilson clover

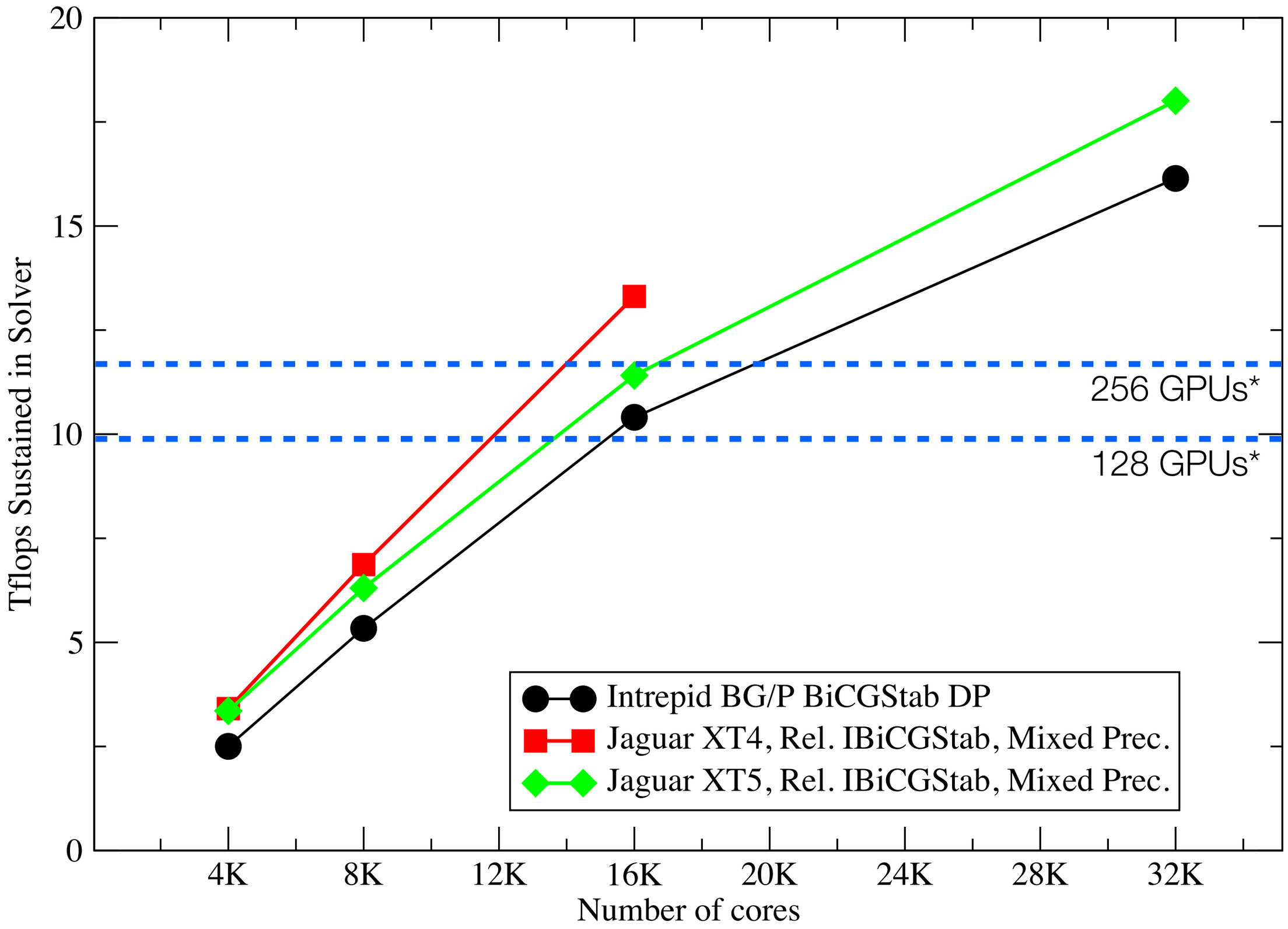
$32^3 x 256$



Wilson clover

$32^3 x 256$

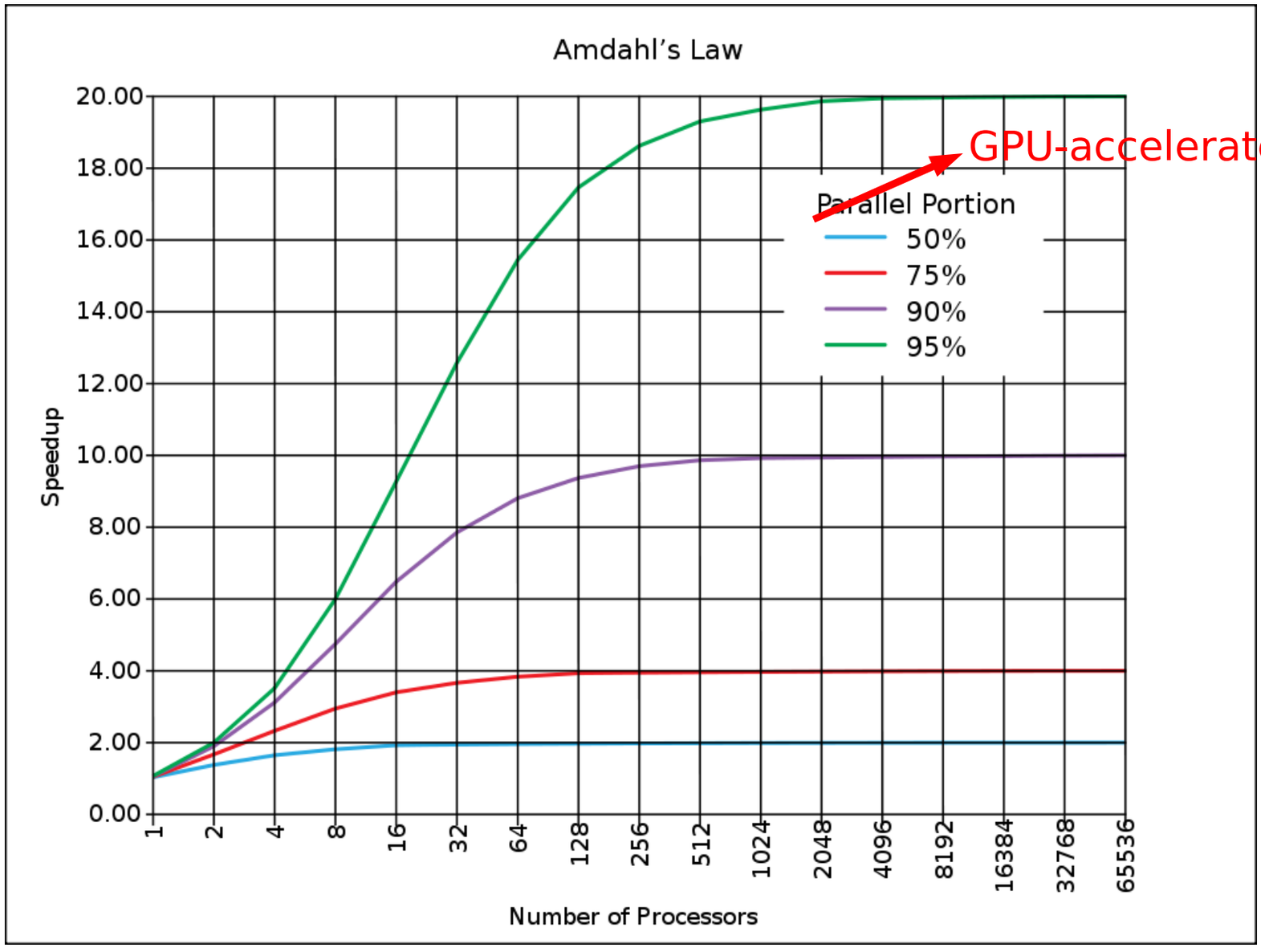

64
32
16
8
4
Time to solution (sec)
BiCGstab
GCR-DD
8
16
32
64
128
256
Number of GPUs



*GPU Tflops scaled according to solver iterations

Amdahl's Law

- It's not enough to speed up just the solver



Source: http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Gauge Generation on GPUs

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		

Gauge Generation on GPUs

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		

Gauge generation on GPUs

- Initial support for HMC kernels

	description	status	Results* (Gflops)
CG	Update spinors using the neighboring links and spinors through Conjugate Gradient process	Fully implemented (18, 12 and 8-reconstruct, SP, DP and half precision, mixed precisions)	33(DP) 108 (SP) 128 (HP)
Fat	Update fatlink using the neighboring links	Implemented the 12-reconstruct, single precision case	178(SP)
GF	Update the momentum using the neighoring links	Implemented 12-reconstruct, single precision	208 (SP)
FF	Update the momenum using the neighboring links and spinors	Implemented 12-reconstruct, single precision	111 (SP)

** The results is obtained in a single gtx280*

- Yet to be fully exposed through the interface

- Full HMC is the next step

Summary and Conclusions

- GPUs are revolutionising LQCD computations
- QUDA is a highly optimized library for QCD on CUDA
    - Provides highly optimized solvers for most Dirac operator discretisations
- Implemented and deployed up to 256 GPUs for LQCD calculation
    - First demonstration of QCD GPU computation at the 10+ Tflops scale
    - GPU clusters can replace capability super computers
- Communication reduction algorithms critical
    - Flops are cheaper than bytes
- Next step is complete gauge generation on GPUs