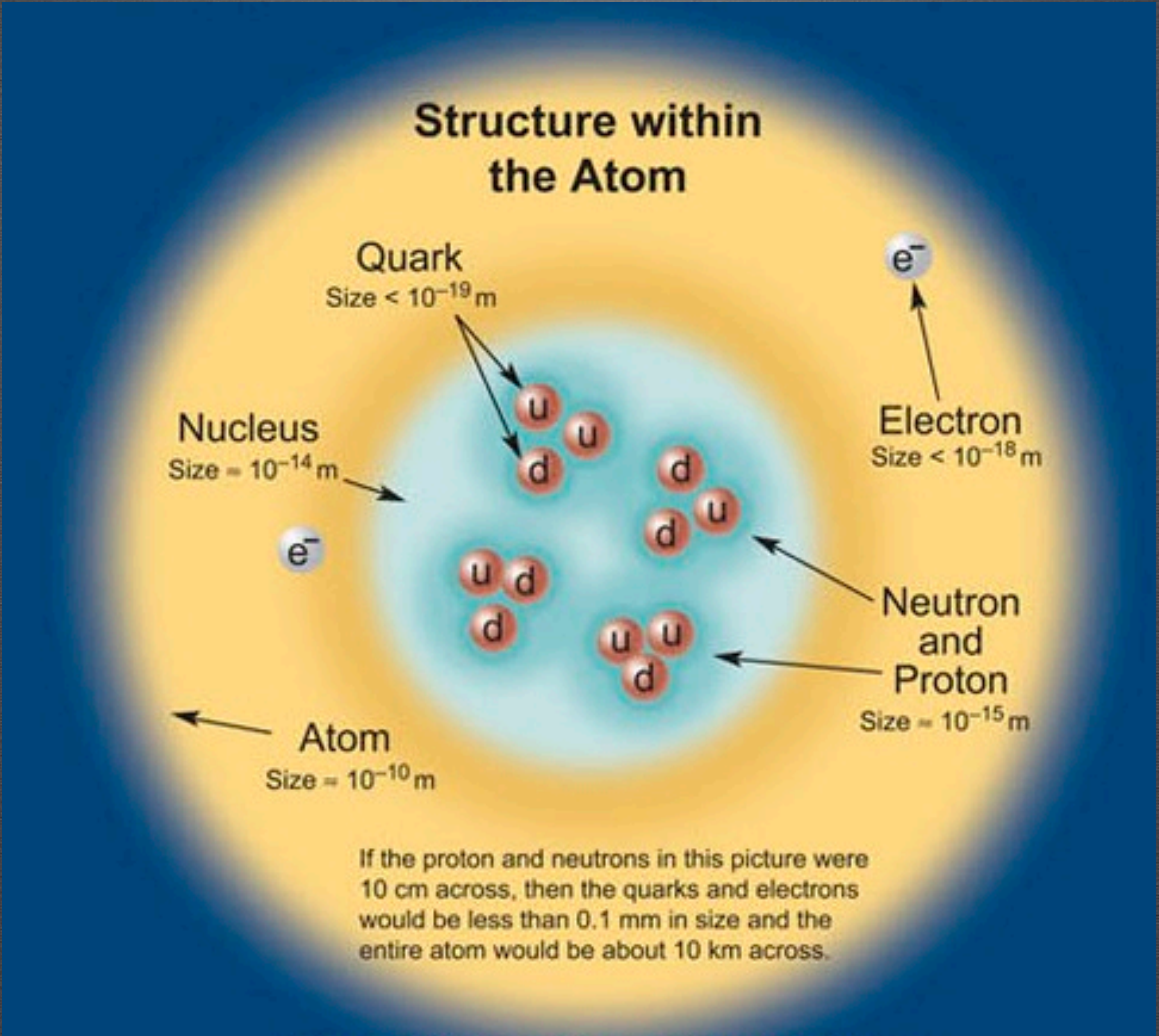
GPU Mixed-precision linear equation solver for lattice QCD

Michael Clark

Harvard University

with
R. Babich, K. Barros, R. Brower, J. Chen and C. Rebbi



If the proton and neutrons in this picture were 10 cm across, then the quarks and electrons would be less than 0.1 mm in size and the entire atom would be about 10 km across.

Properties of the Interactions

Quantum ChromoDynamics

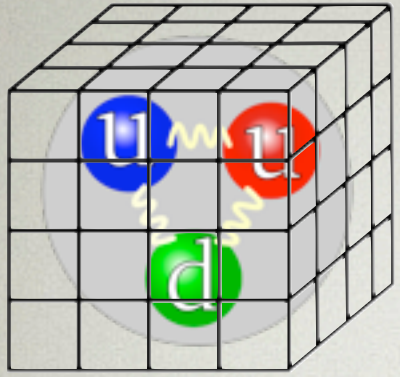
- QCD is the theory of the strong force that binds nucleons
- Impose local SU(3) symmetry on vacuum
  - Color charge analogous to electric charge of EM
- Lagrangian of the theory very simple to write down

$$L_{QCD} = \psi_i \left(i\gamma^{\mu}(D_{\mu})_{ij} - m\delta_{ij}\right) \psi_j - G_{\mu\nu}{}^{a} G^{\mu\nu}{}_{a}$$

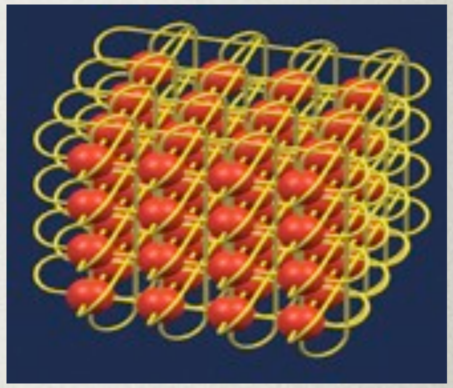
- Path integral formulation

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

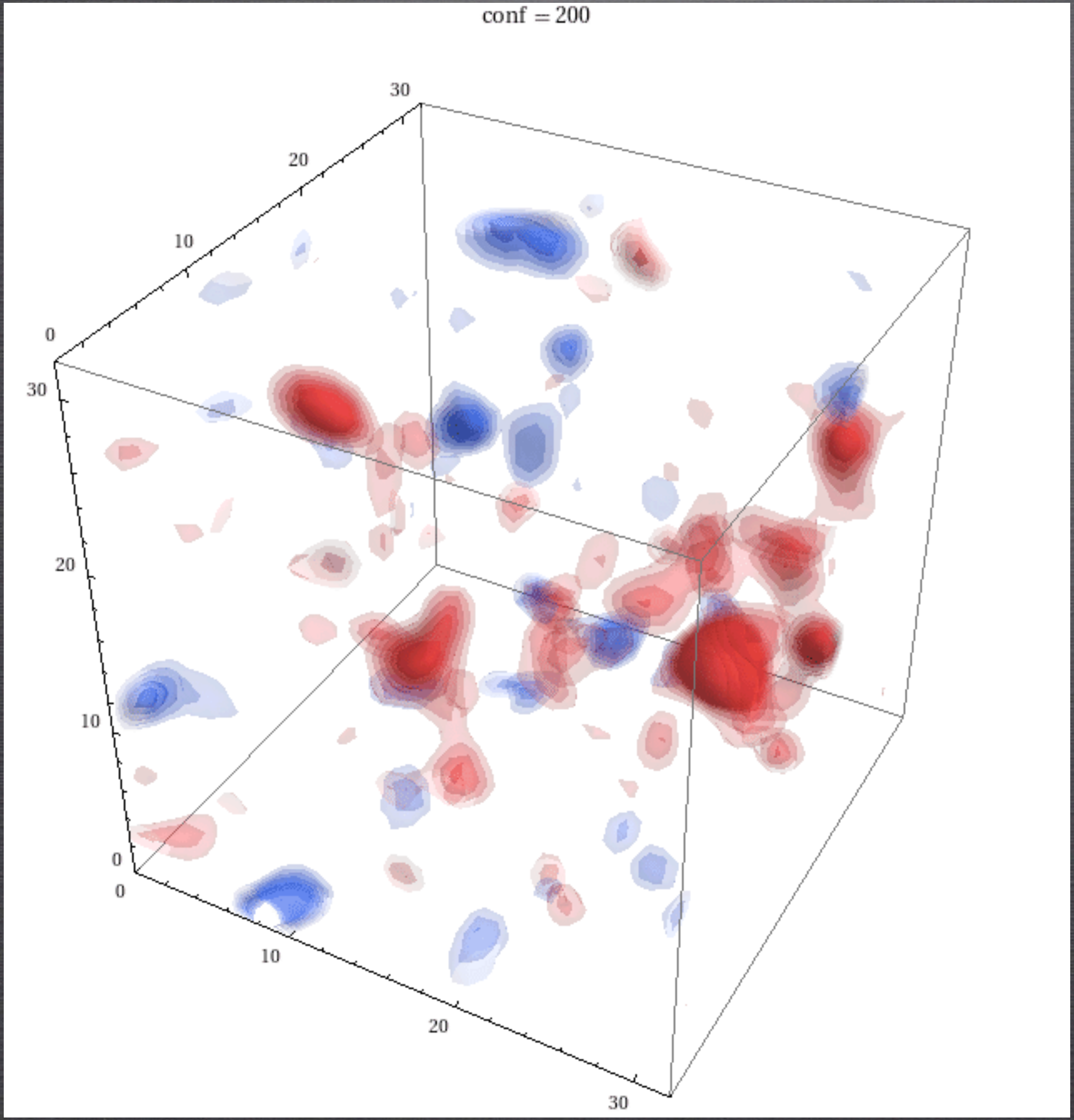
- Infinite dimensional integral
- Theory is strictly non-perturbative at low energies



Lattice QCD



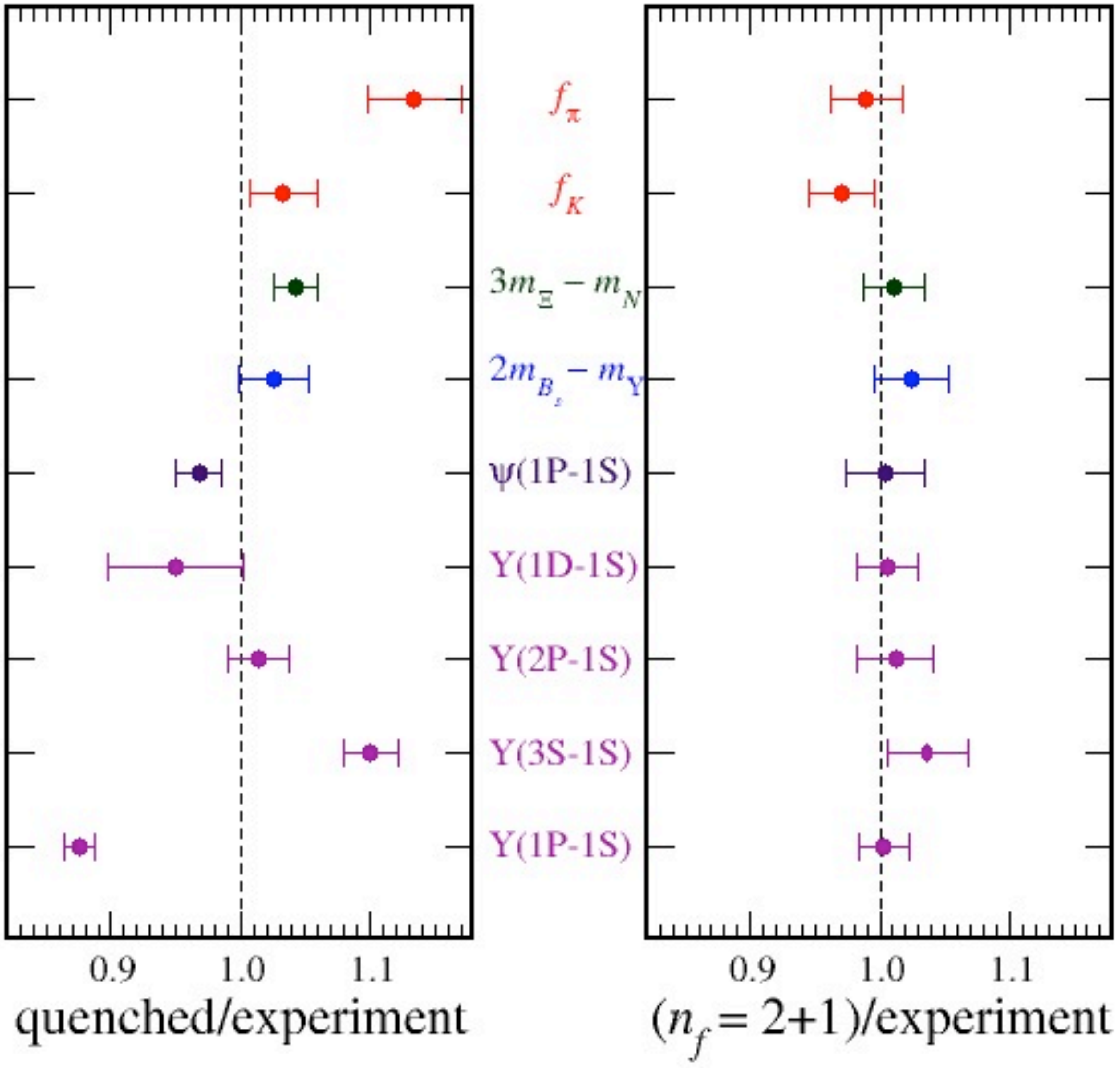
- Only known non-perturbative method is lattice QCD
  - Discretize and finitize spacetime
  - 4d periodic spacetime lattice (e.g., 128^4 x 3 x 4 dof)
- 10^8-10^9 dimension integral => Monte Carlo integration
- Interpret $e^{-\int d^4x L(U)}$ as a Boltzmann weight
  - Use importance sampling $\langle \Omega \rangle \approx \frac{1}{N}\sum_{i=1}^{N} \Omega(U_i)$
- Lattice QCD is a 2 step process
  - Generate (gluon field) configurations with weight $e^{-\int d^4x L(U)}$
  - Calculate mean observables
- Ab initio calculation to verify QCD is theory of strong force



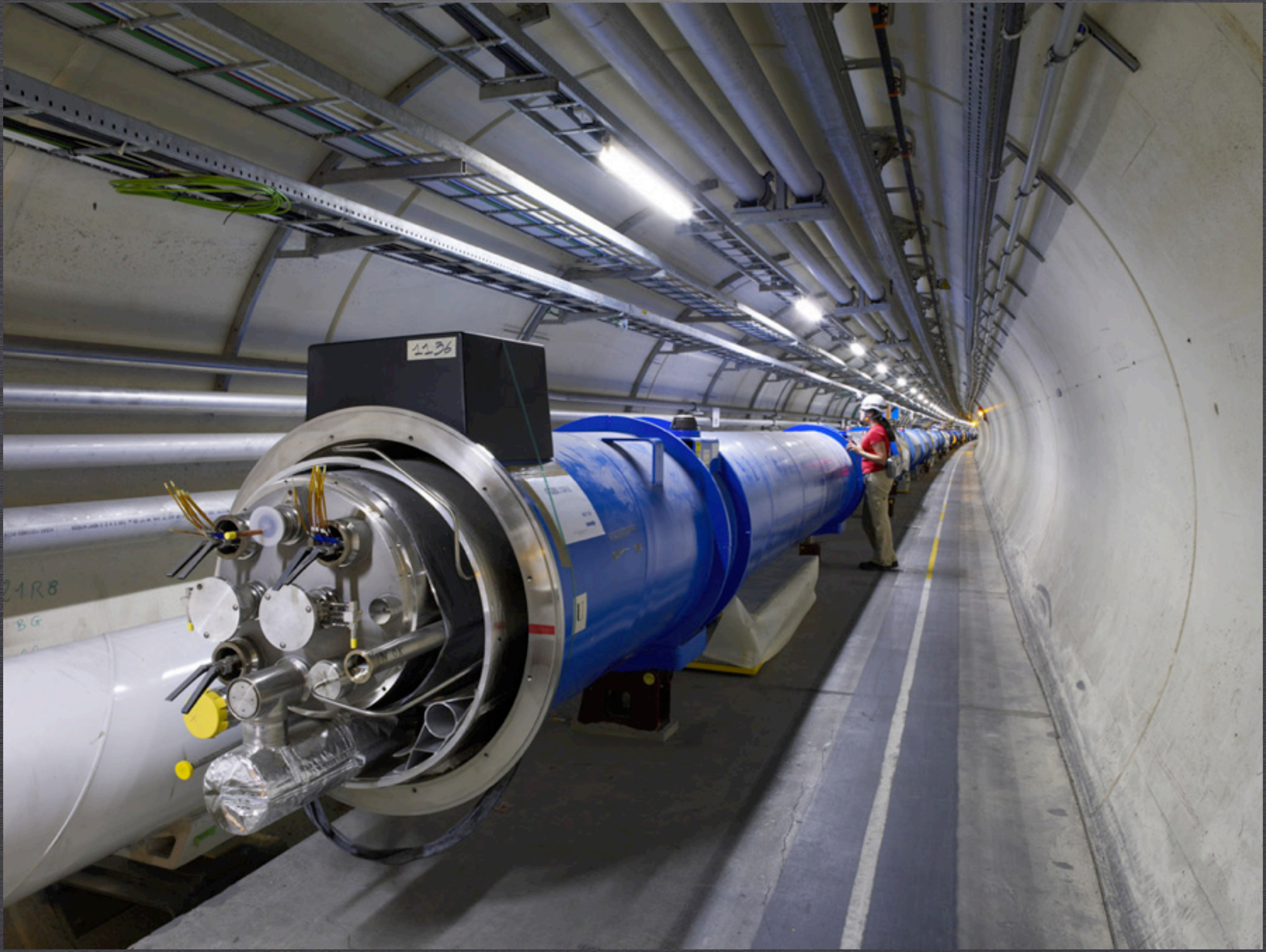
topology

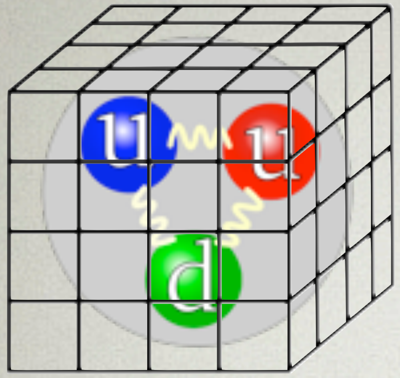
string

Visualisation coutesty of M. Di Pierro



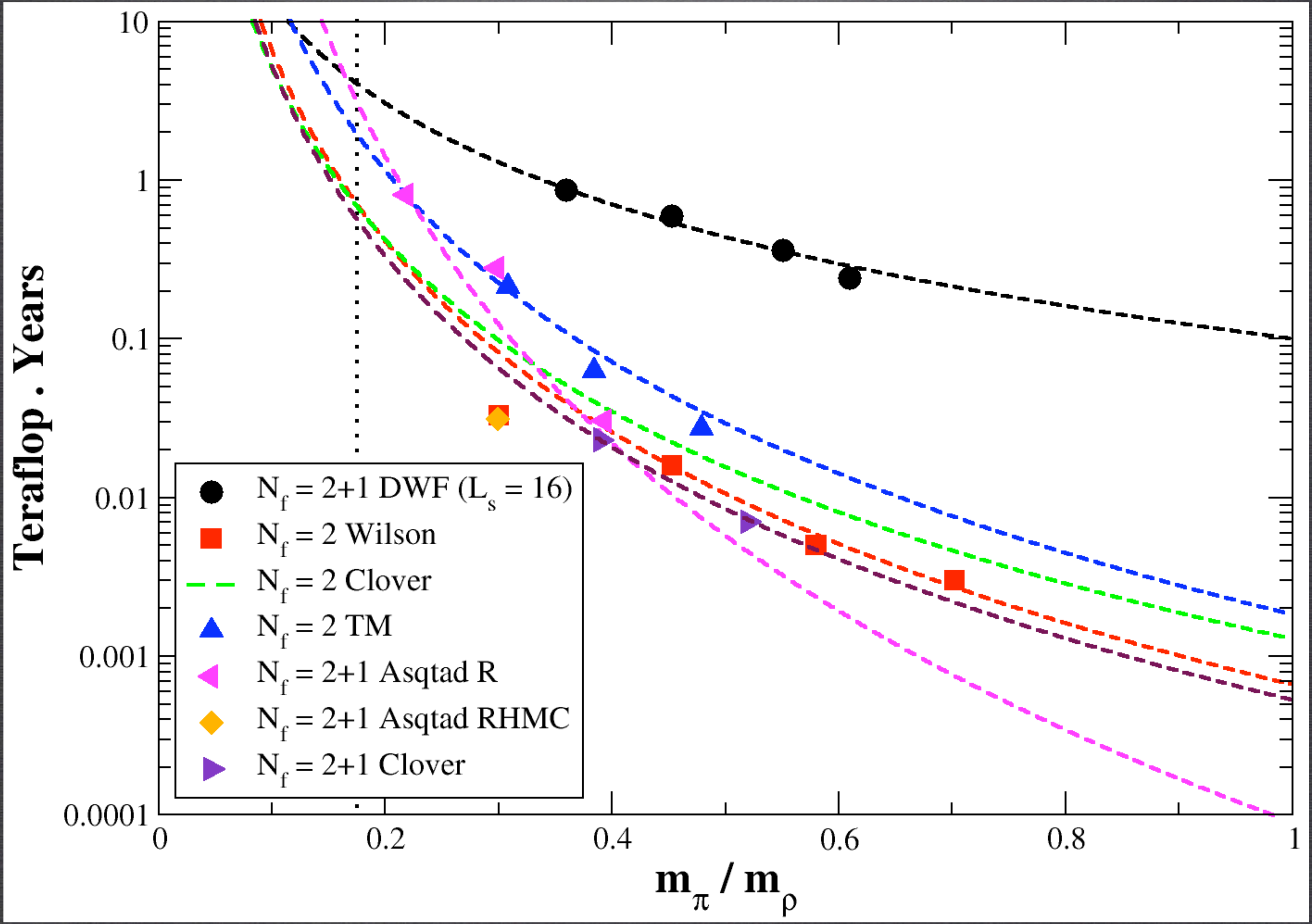
f_π
f_K
$3m_\Xi - m_N$
$2m_{B_s} - m_\Upsilon$
ψ(1P-1S)
Υ(1D-1S)
Υ(2P-1S)
Υ(3S-1S)
Υ(1P-1S)
0.9
1.0
1.1
quenched/experiment
$(n_f = 2+1)$/experiment


1136
21R8
BG



Lattice QCD

- Many computational / algorithmic challenges, e.g.:
  - Discretization
  - Monte Carlo integration
  - Molecular dynamics
  - Matrix function evaluation
  - Sign problem
  - Solving systems of linear equations
- Grand challenge problem
  - Peta/Exaflops required
  - GPUs as a means of getting there?



10
1
0.1
0.01
0.001
0.0001
0
0.2
0.4
0.6
0.8
1
Teraflop . Years
m_π / m_ρ
N_f = 2+1 DWF (L_s = 16)
N_f = 2 Wilson
N_f = 2 Clover
N_f = 2 TM
N_f = 2+1 Asqtad R
N_f = 2+1 Asqtad RHMC
N_f = 2+1 Clover

GPU Hardware

GTX 280
Flops: single ~1 Tflop, double ~80 Gflops
Memory 1GB, Bandwidth 141 GBs^{-1}
230 Watts, $350

Tesla 1060
Flops: single ~1 Tflop, double ~80 Gflops
Memory 4GB, Bandwidth 102 GBs^{-1}
230 Watts, $1200

Tesla 1070
Flops: single ~4 Tflops, double ~320 Gflops
Memory 16GB, Bandwidth 408 GBs^{-1}
900 Watts, $8000

Solving Large Systems of Equations

$$A\mathbf{x} = \mathbf{b}$$

- Assumptions:
  - A is sparse (O(N) non-zeros)
  - N large (10^7-10^{10})
- In general the explicit matrix inverse is never needed
  - Only interested in solution x to some precision ε
- Gaussian elimination $O(N^3)$
- Indirect iterative solvers scale as O(N) - $O(N^2)$

Iterative Linear Solvers

- Many possible iterative solvers
- Optimal method will depend on
  - Nature of matrix: SPD, HPD, RPD, indefinite etc.
  - Nature of hardware: serial, MP, geometry, comms overhead etc.
- E.g.,
  - Jacobi
  - Gauss-Seidel
  - Multigrid
  - Krylov subspace methods

Krylov Solvers

- e.g., Conjugate Gradients

Dominant cost is mat-vec

```
while ($|r_k| > \varepsilon$) {
    $\beta_k = (\mathbf{r}_k,\mathbf{r}_k)/(\mathbf{r}_{k-1},\mathbf{r}_{k-1})$
    $\mathbf{p}_{k+1} = \mathbf{r}_k - \beta_k\mathbf{p}_k$

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

- Krylov solvers can all be decomposed into simple linalg kernels
- Matrix-vector product is inherently parallel
- Ideal for GPU implementation
  - Just need a fast mat-vec?

Dirac operator of QCD

- From the QCD Lagrangian

$$L_{QCD} = \psi_i \, (i\gamma^{\mu}(D_{\mu})_{ij} - m\delta_{ij})\, \psi_j - G_{\mu\nu}{}^{a} G^{\mu\nu}{}_{a}$$

Dirac operator of QCD

- The Dirac operator represent quark interactions

$$i\gamma^{\mu}(D_{\mu})_{ij} - m\delta_{ij}$$

- Essentially a PDE with background SU(3) field
- Many discretization strategies
  - Wilson discretization
  - others: Overlap, staggered etc.

Wilson Matrix of QCD

$$i\gamma^{\mu}(D_{\mu})_{ij} - m\delta_{ij}$$

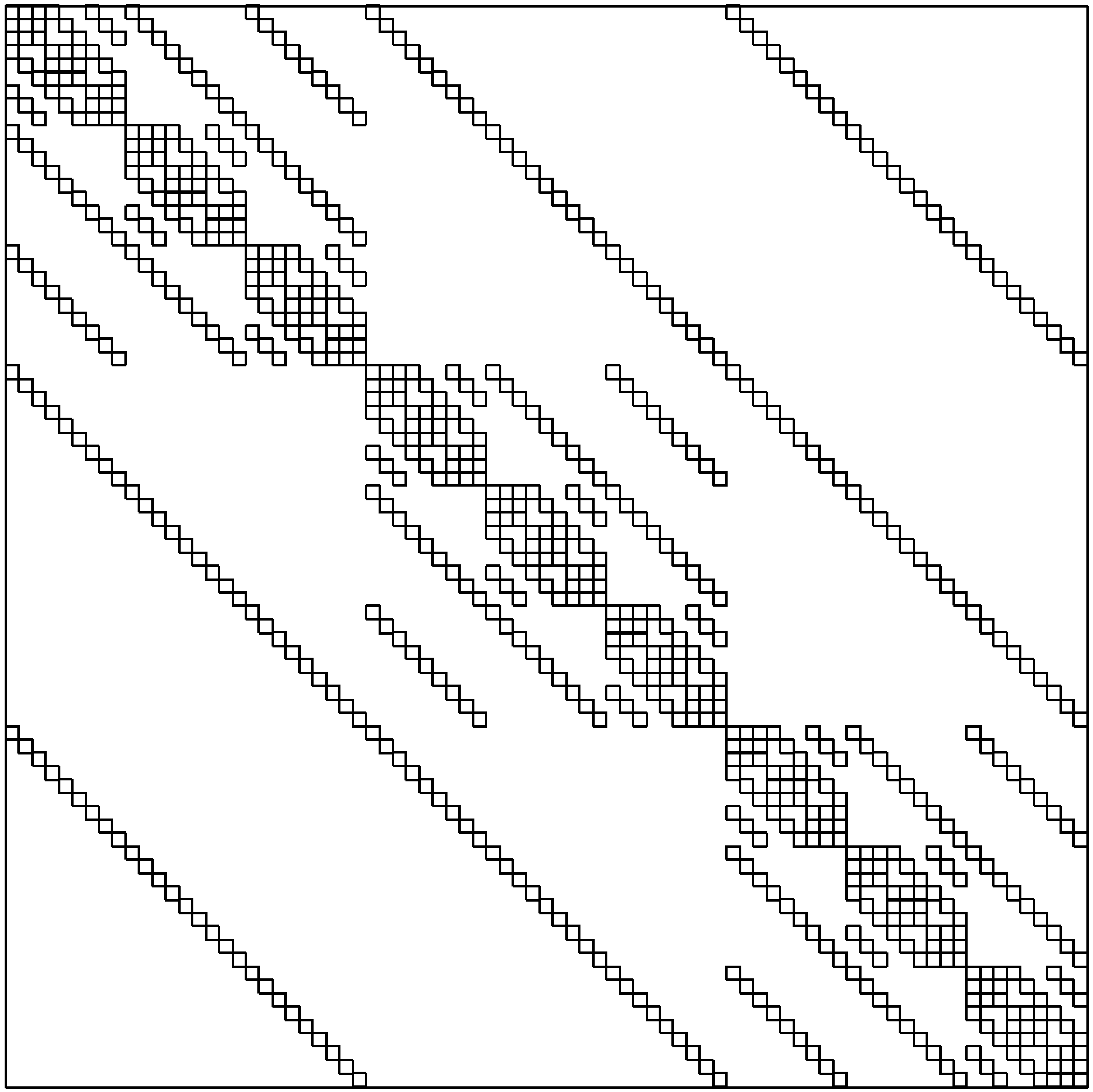
Wilson Matrix of QCD

$$\Sigma_\mu \left((1-\gamma^\mu)U_{x,y}{}^\mu\delta_{x+\mu,y} + (1+\gamma^\mu)U_{y,x}{}^{\mu\dagger}\delta_{x-\mu,y}\right) + (4+m)\delta_{x,y}$$

Wilson Matrix of QCD

$$\Sigma_\mu \left((1-\gamma^\mu)U_{x,y}{}^{\mu}\delta_{x+\mu,y} + (1+\gamma^\mu)U_{y,x}{}^{\mu\dagger}\delta_{x-\mu,y}\right) + (4+m)\delta_{x,y}$$

- U is discretized gauge field (SU(3))
- γ are Dirac matrices (4x4)
- 8 off-diagonals in spacetime, mass on diagonal
  - Off-diagonals are 12x12 matrices (SU(3) x γ)
- Each point in spacetime referred to as a *spinor*
- Matrix not Hermitian but γ_5-Hermitian
- Block “Laplace” in 4 dimensions



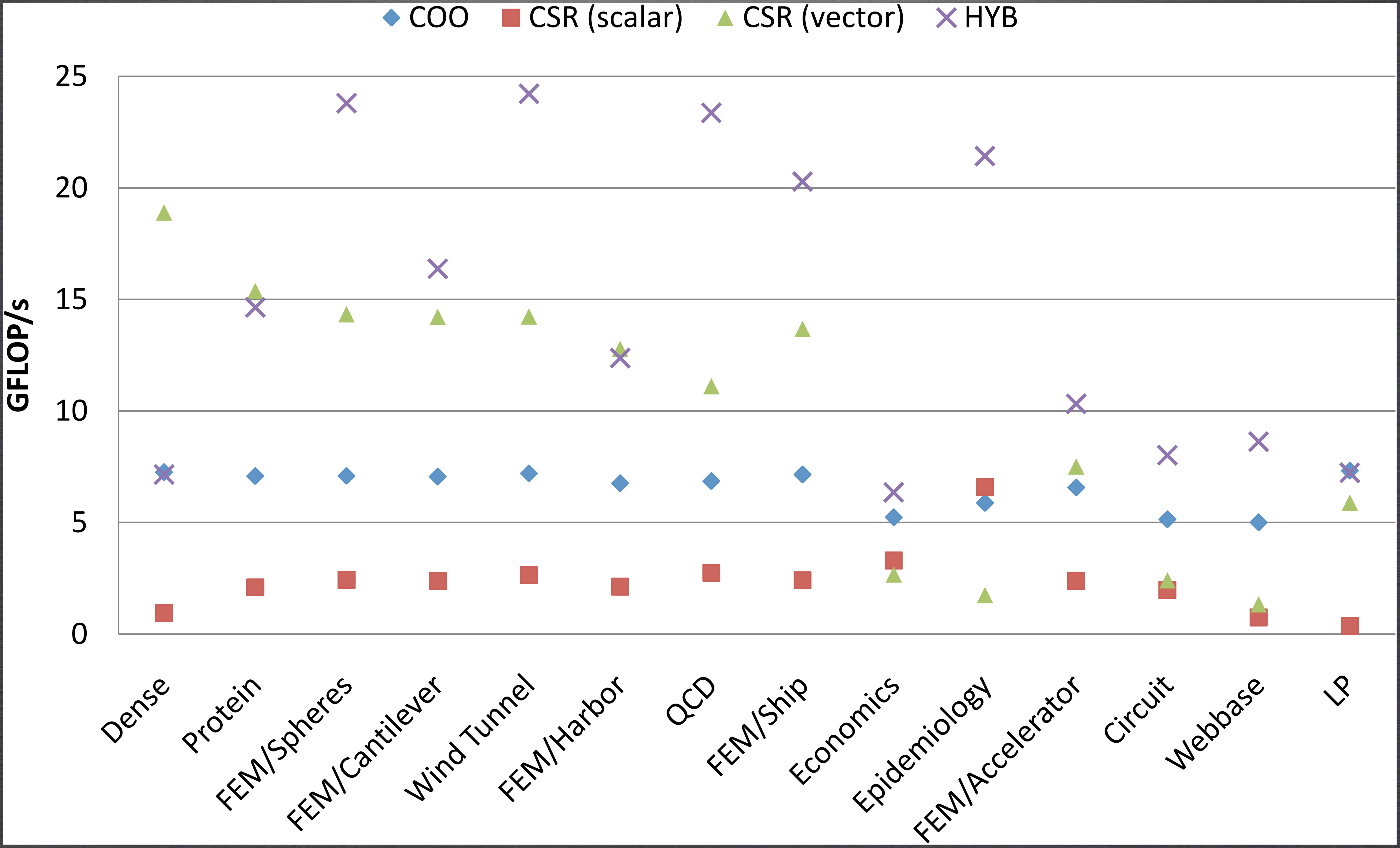
Wilson Matrix of QCD

$$\Sigma_\mu \left((1-\gamma^\mu)U_{x,y}{}^\mu\delta_{x+\mu,y} + (1+\gamma^\mu)U_{y,x}{}^{\mu\dagger}\delta_{x-\mu,y}\right) + (4+m)\delta_{x,y}$$

- Quark physics requires solution A(U)x=b
- Krylov solvers standard method
- Condition number given by ~(quark mass)$^{-1}$
  - Up / down quark masses are light
  - Computationally expensive

SpMV I

- Standard matrix-vector libraries available (Bell and Garland)
  - Pack your matrix, call library function, unpack
- Ignorant of structure and symmetries of problem
  - Bad for storage (double storage of matrix elements)
  - Bad for operation count (200% flops)
  - Bad for compute intensity (1:2 flop / byte ratio)
- Consider single precision and single GPU only



Bell and Garland (NVIDIA) 2009

SpMV II

- Wilson matrix is just a block stencil
  - Much better to consider matrix as a nearest neighbor gather operation
- Assign thread to each spacetime point -> massive parallelism
  - Avoids double storage of matrix elements (Hermiticity)
  - Repetitive structure means no explicit indexing required
- Can order data optimally for any given hardware
  - Reorder field elements for coalescing
- Large reduction in flops and required bandwidth
  - 1:1 flop / bandwidth ratio
  - Better, but still very bandwidth limited

SpMV III

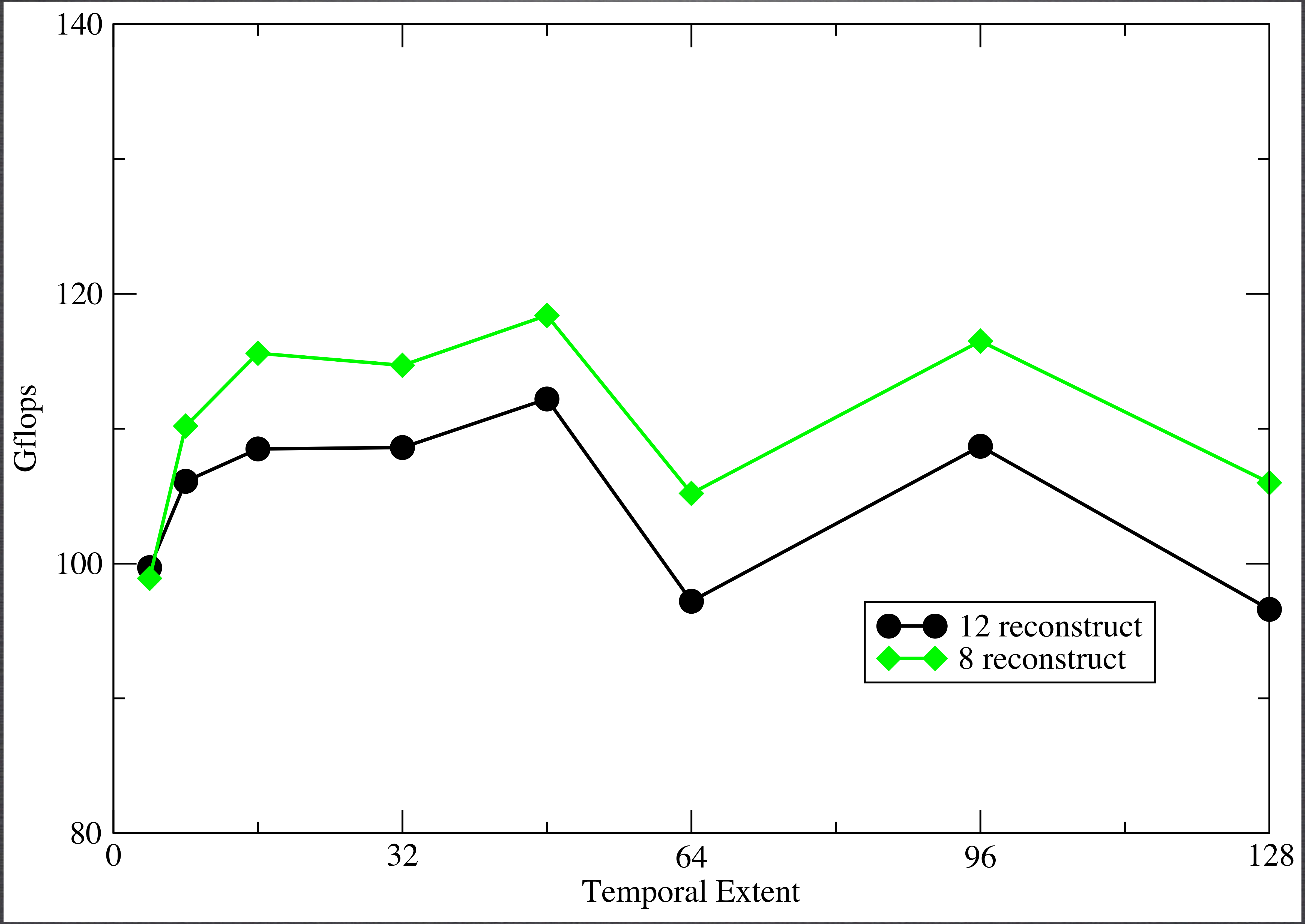
- Wilson matrix is a matrix of matrices
- SU(3) matrices are all unitary complex matrices with det = 1
  - 18 numbers with 4 orthogonality and 6 normality constraints
- 12 number parameterization: bytes 80%, flops 128%

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \longrightarrow \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix} \qquad \mathbf{c} = (\mathbf{a} \times \mathbf{b})^*$$

- Minimal 8 number parameterization: bytes 71%, flops 163%

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \longrightarrow \begin{pmatrix} \arg(a_1) & \arg(c_1) & \mathrm{Re}(a_2) & \mathrm{Im}(a_2) \\ \mathrm{Re}(a_3) & \mathrm{Im}(a_3) & \mathrm{Re}(b_1) & \mathrm{Im}(b_1) \end{pmatrix}$$

  - Obtain a_1 and c_1 from normality
  - Reconstruct b_2, b_3, c_2, c_3 from SU(2) rotation



Wilson Matrix-Vector Performance

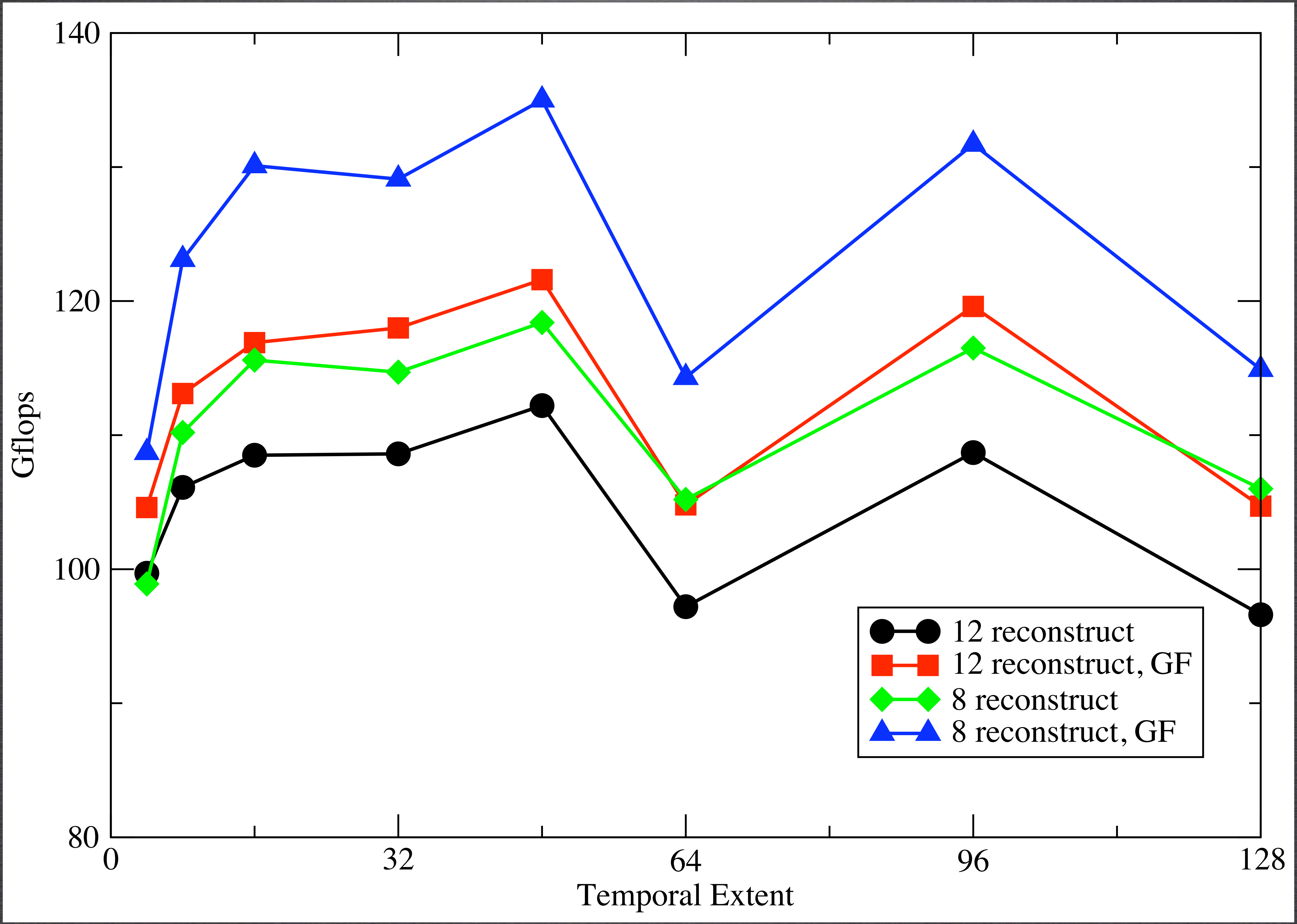
Single Precision (V=24^3xT)

SpMV IV

- Can impose similarity transforms to improve sparsity
- Can globally change Dirac matrix basis

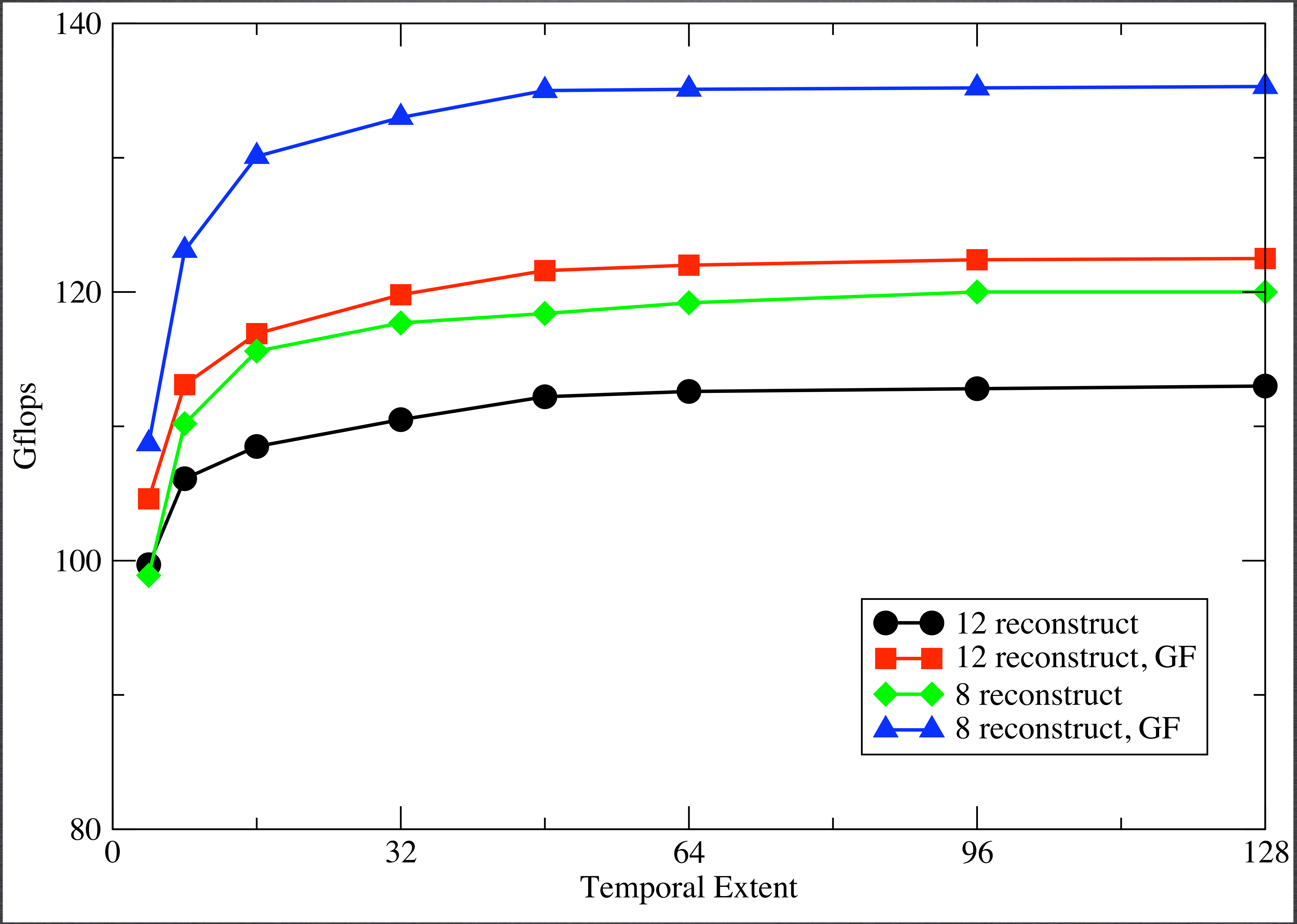
$$P^{\pm 4} = \begin{pmatrix} 1 & 0 & \pm 1 & 0 \\ 0 & 1 & 0 & \pm 1 \\ \pm 1 & 0 & 1 & 0 \\ 0 & \pm 1 & 0 & 1 \end{pmatrix} \longrightarrow P^{+4} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} P^{-4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0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

- Impose local color transformation (gauge transformation)
    - SU(3) field = unit matrix in temporal direction
    - Must calculate this transformation (done once only)
- In total 33% reduction in bandwidth



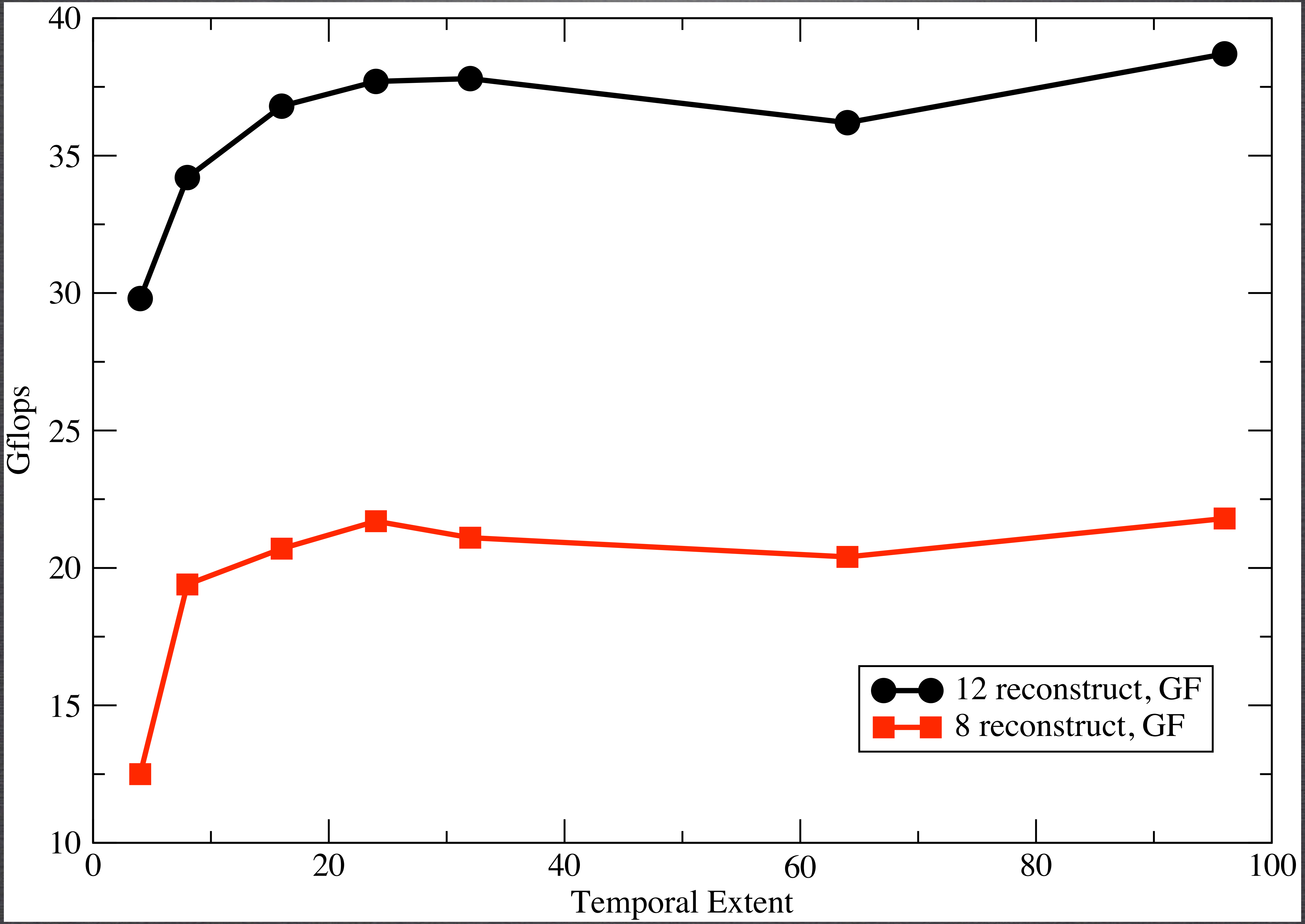
Wilson Matrix-Vector Performance

Single Precision ($V=24^3xT$)



Wilson Matrix-Vector Performance

Single Precision, padded (V=24^3xT)

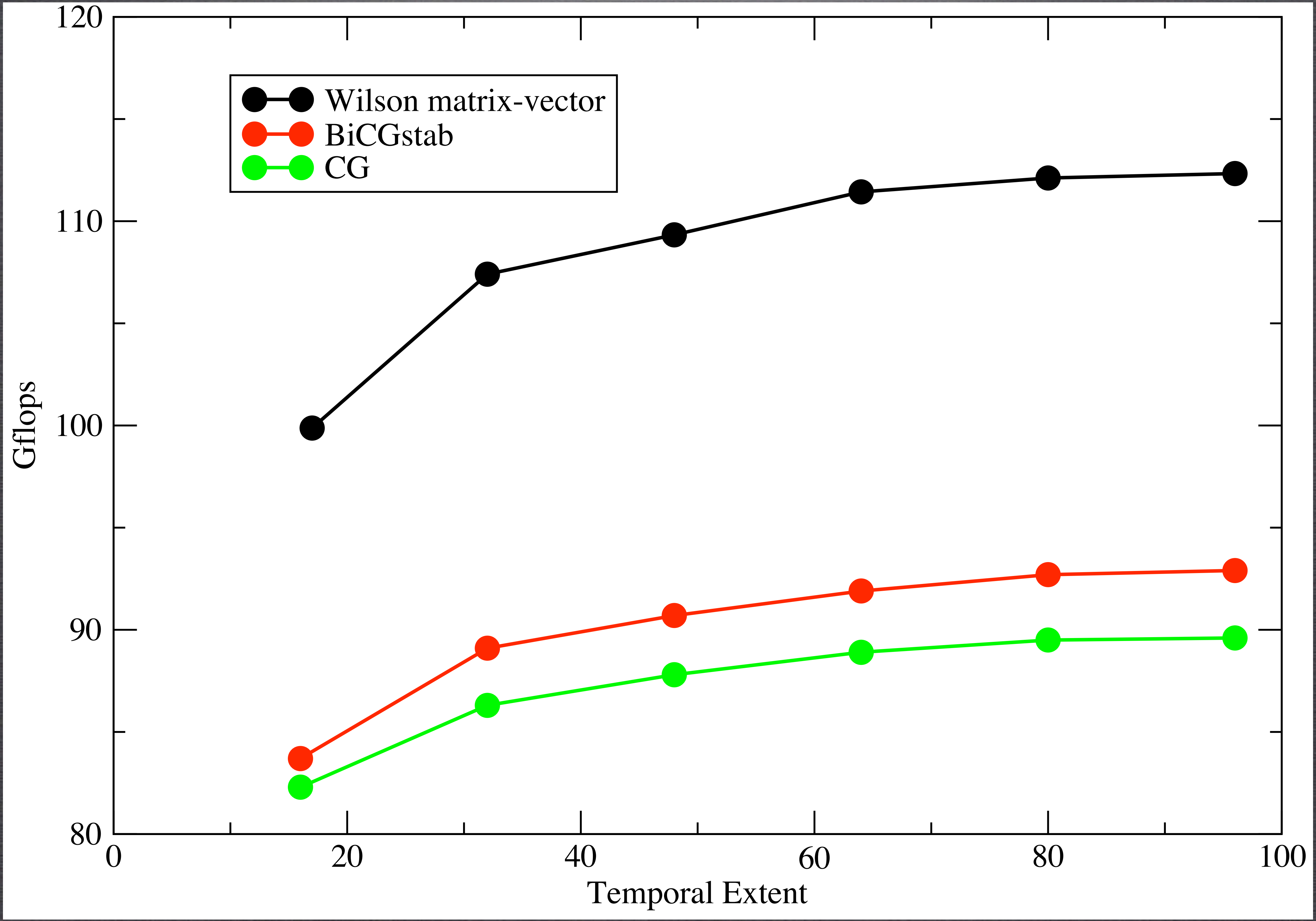


Wilson Matrix-Vector Performance

Double Precision ($V = 24^3 x T$)

Krylov Solver Implementation

- Complete solver must be on GPU
  - Transfer **b** to GPU
  - Solve A**x**=**b**
  - Transfer **x** to CPU
- Besides matrix-vector, require BLAS level 1 type operations
  - AXPY operations: **b** += a**x**
  - NORM operations: c = (**b**,**b**)
- CUBLAS library available
- Better to fuse kernels to minimize bandwidth
  - e.g., AXPY_NORM

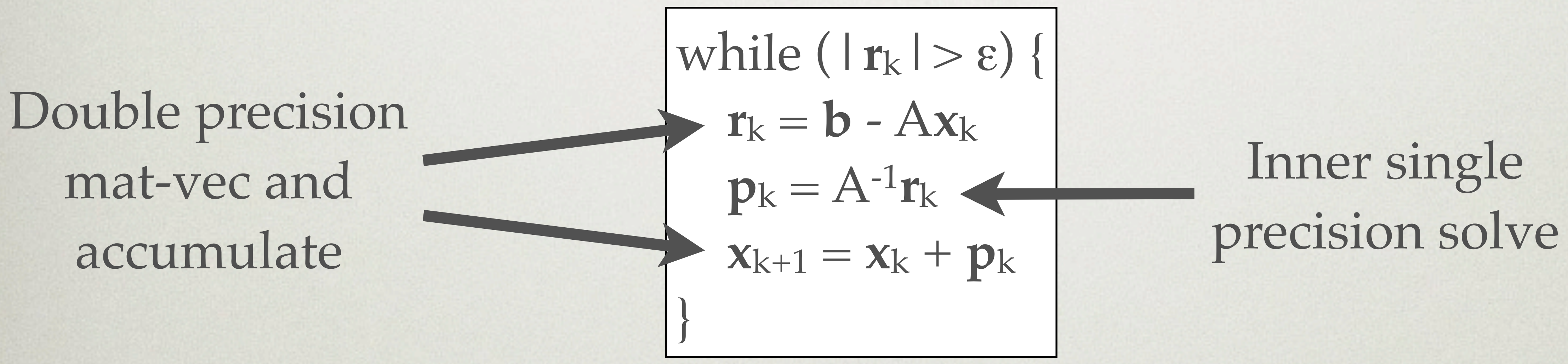


Wilson Inverter Performance

Single Precision (12 reconstruct, $V=24^3 x T$)

Mixed-Precision Solvers

- Require solver tolerance beyond limit of single precision
  - Double precision is at least x2 slower
- Use defect-correction (iterative refinement)


Double precision mat-vec and accumulate
Inner single precision solve

```
while ($|\mathbf{r}_k| > \varepsilon$) {
    $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$
    $\mathbf{p}_k = A^{-1}\mathbf{r}_k$
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$
}
```

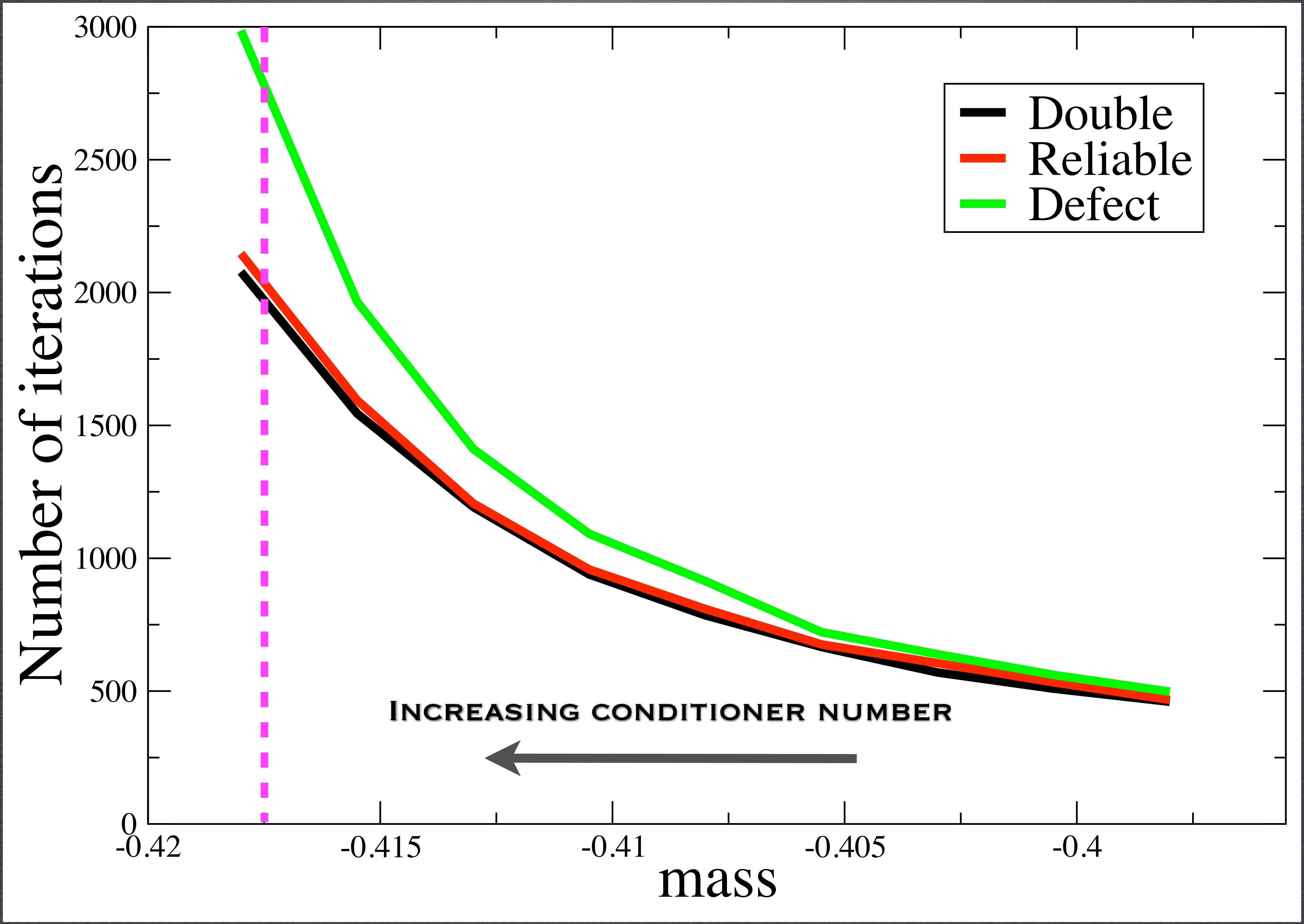
- Double precision done can be done on CPU or GPU
  - Can always check GPU gets correct answer
- Disadvantage is each new single precision solve is a restart

Mixed-Precision Solvers

- Want to use mixed precision, but avoid restart penalty
- Reliable Updates (Sleijpen and Van der Worst 1996)
  - Iterated residual diverges from true residual
  - Occasionally replace iterated residual with true residual
  - Also use second accumulator for solution vector

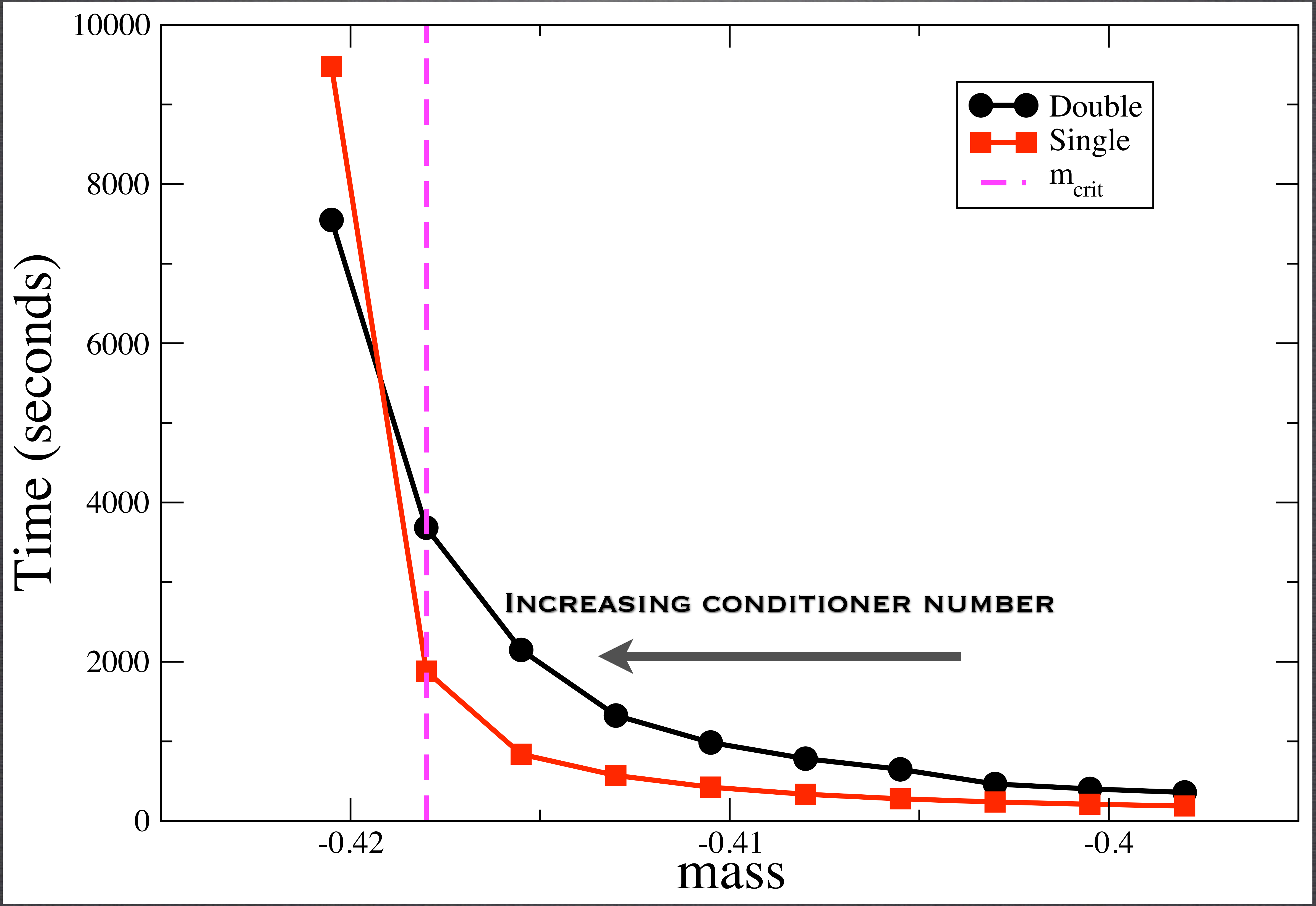
```
if ( $|\mathbf{r}_k| < \delta |\mathbf{b}|$ ) {
    $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$
    $\mathbf{b} = \mathbf{r}_k$
    $\mathbf{y} = \mathbf{y} + \mathbf{x}_k$
    $\mathbf{x}_k = 0$
}
```

- Idea: Use high precision for reliable update



Wilson Inverter Iterations

($\varepsilon=10^{-12}$, $V=24^3\times64$, BiCGstab)

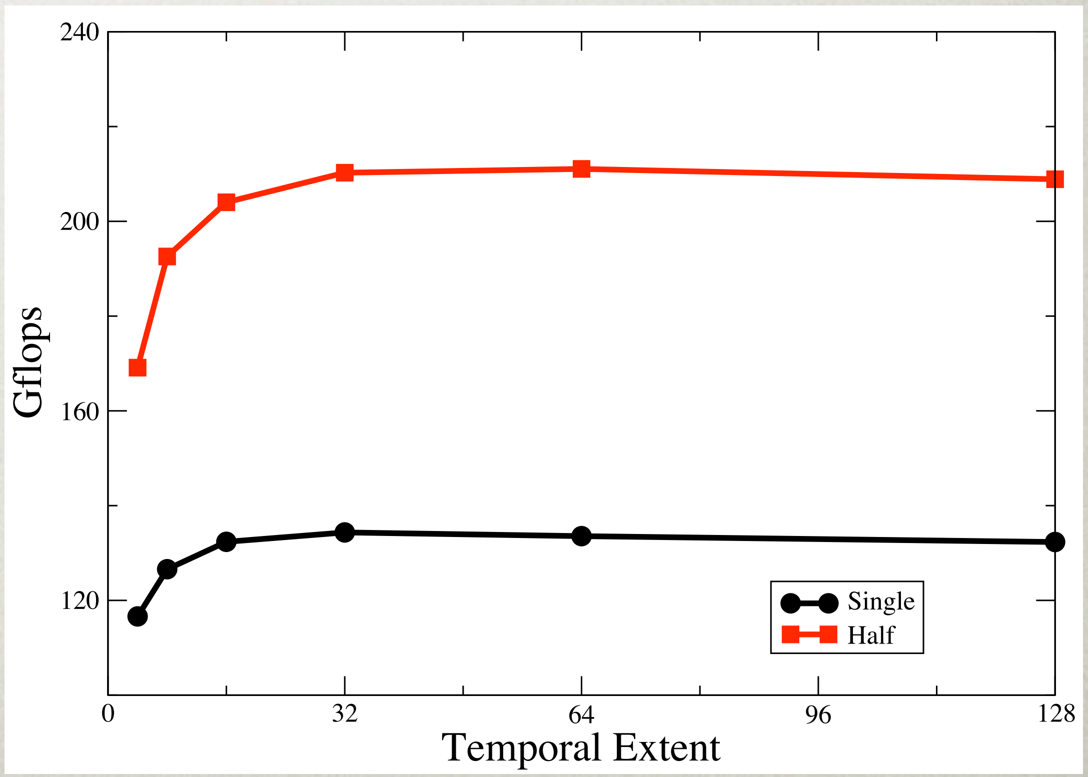


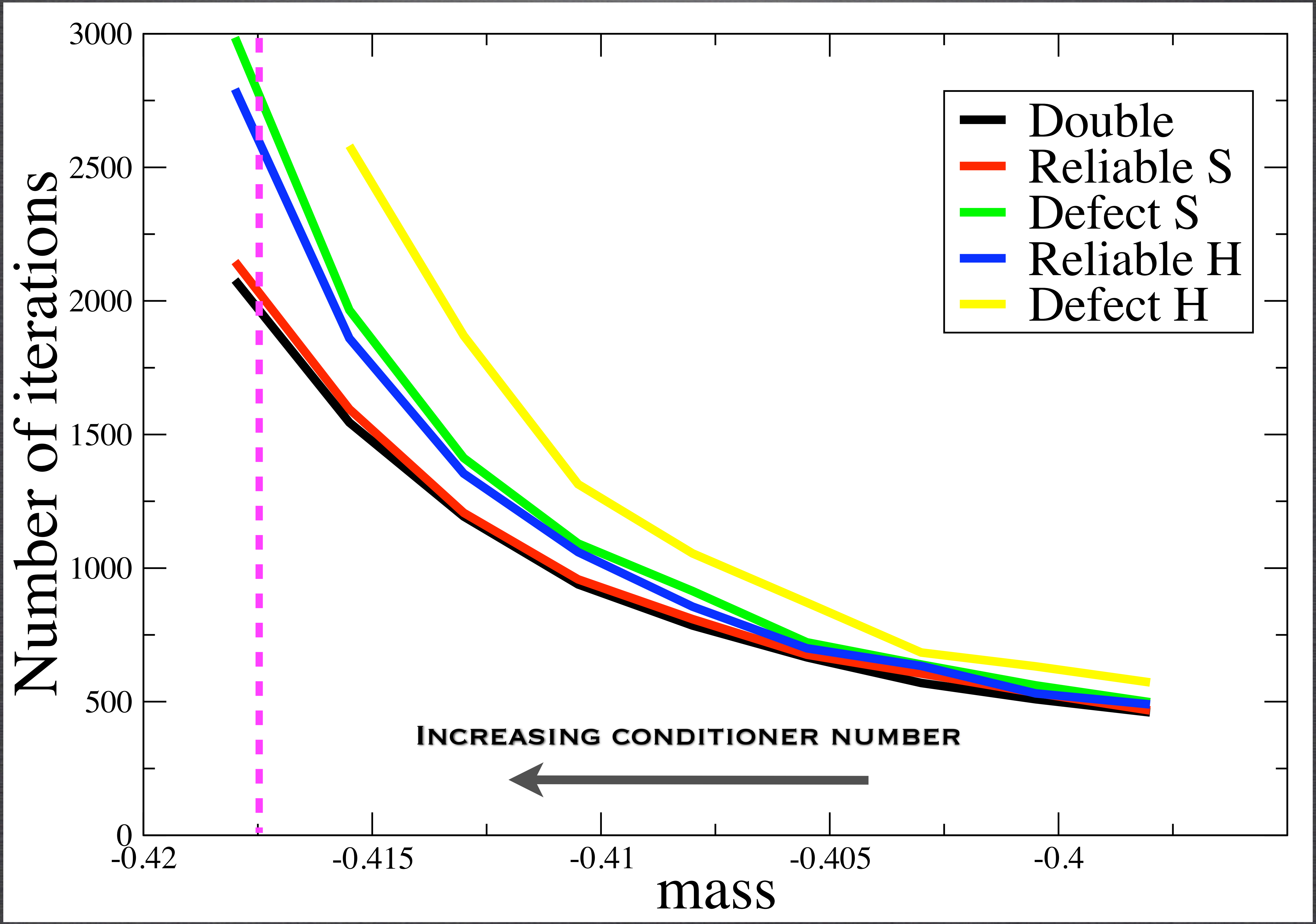
Wilson Inverter Time to Solution

($\varepsilon=10^{-8}$, $V=32^3\text{x}96$, BiCGstab)

Half Precision

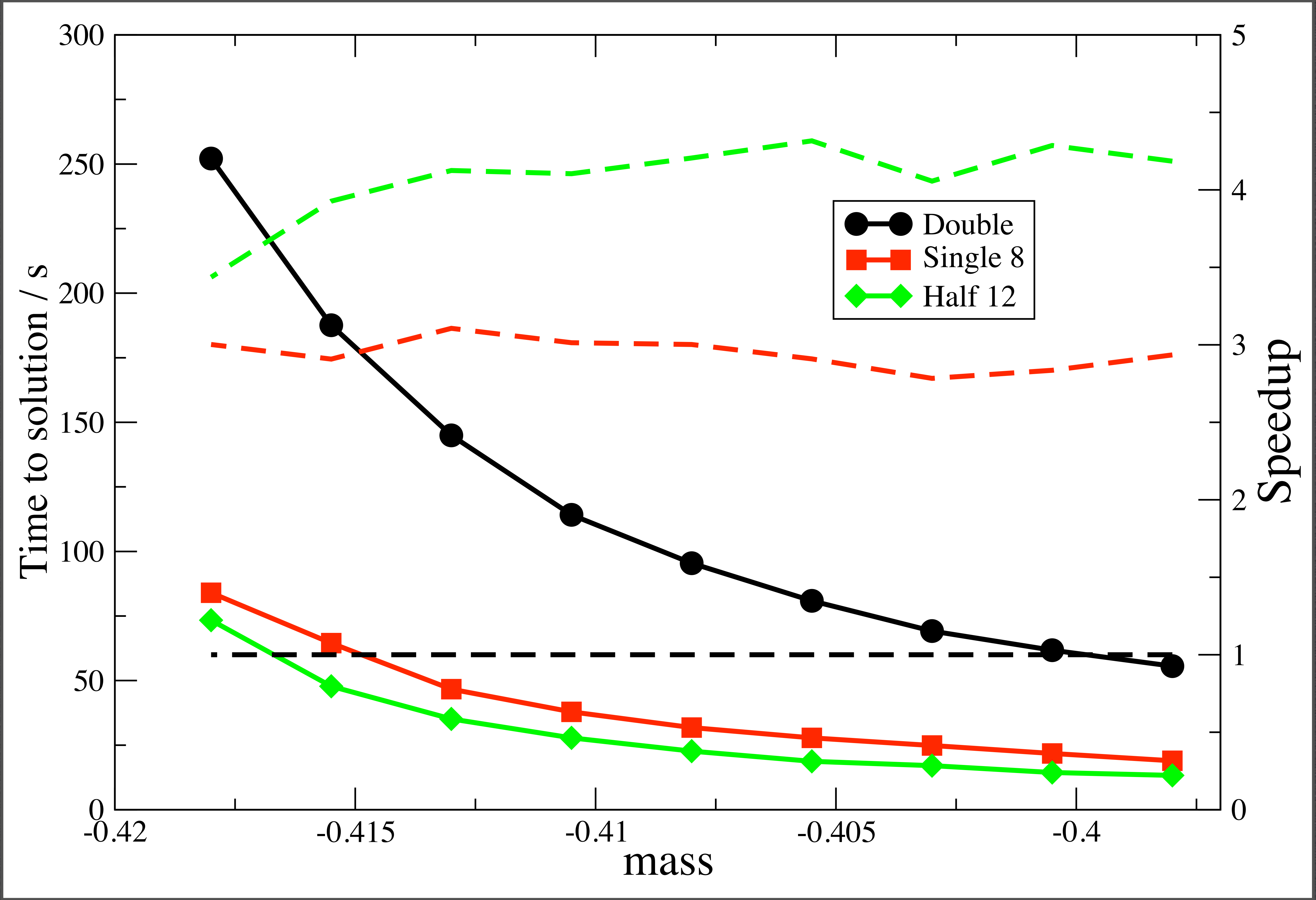
- Single-precision can be used to find double-precision result
- GPU kernel is still bandwidth bound
    - Use half precision for inner solve?
- Performance increases > 210 Gflops





Wilson Inverter Iterations

($\varepsilon=10^{-12}$, $V=24^3x64$, BiCGstab)

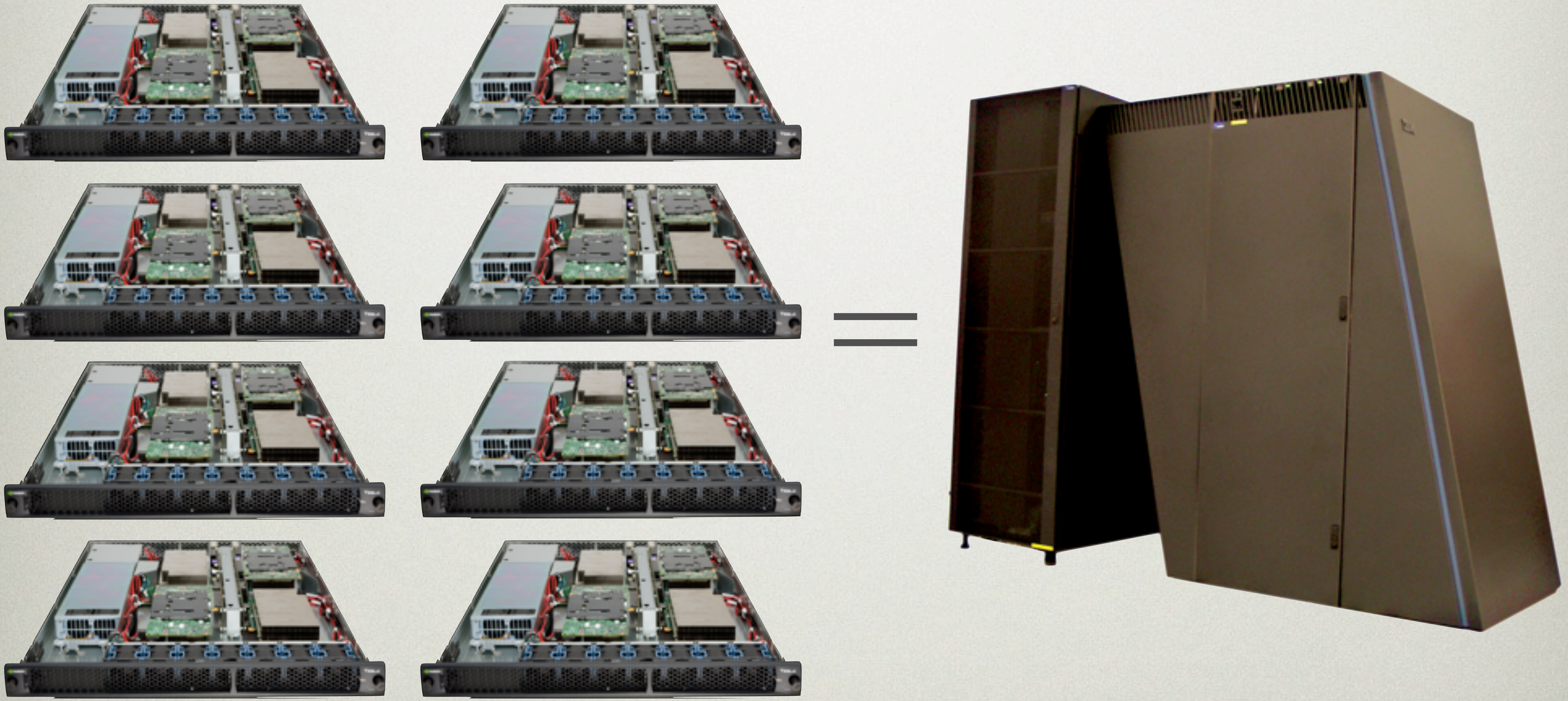


Wilson Inverter Time to Solution

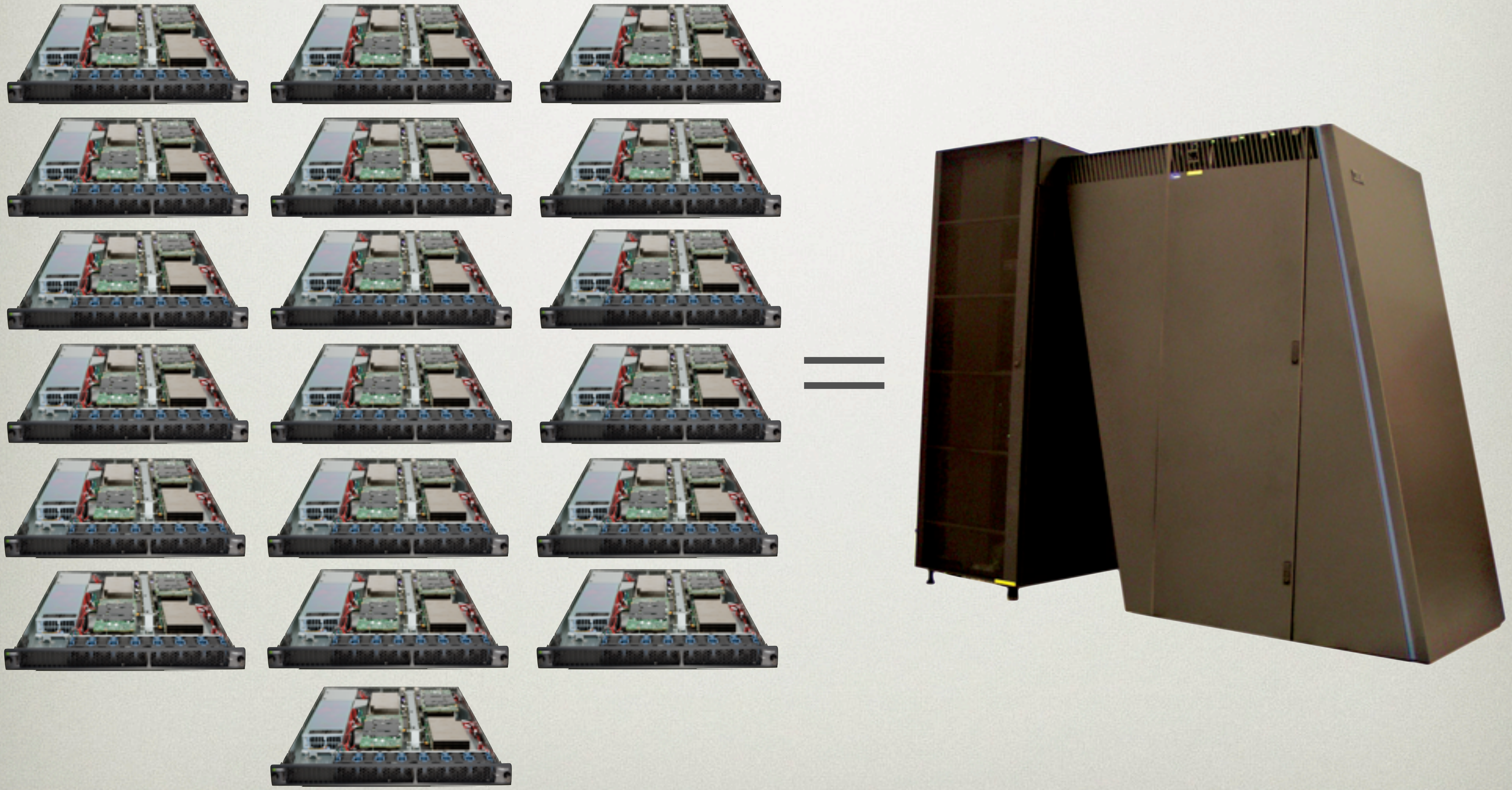
($\varepsilon=10^{-8}$, $V=32^3\times96$, BiCGstab)

How Fast is Fast?

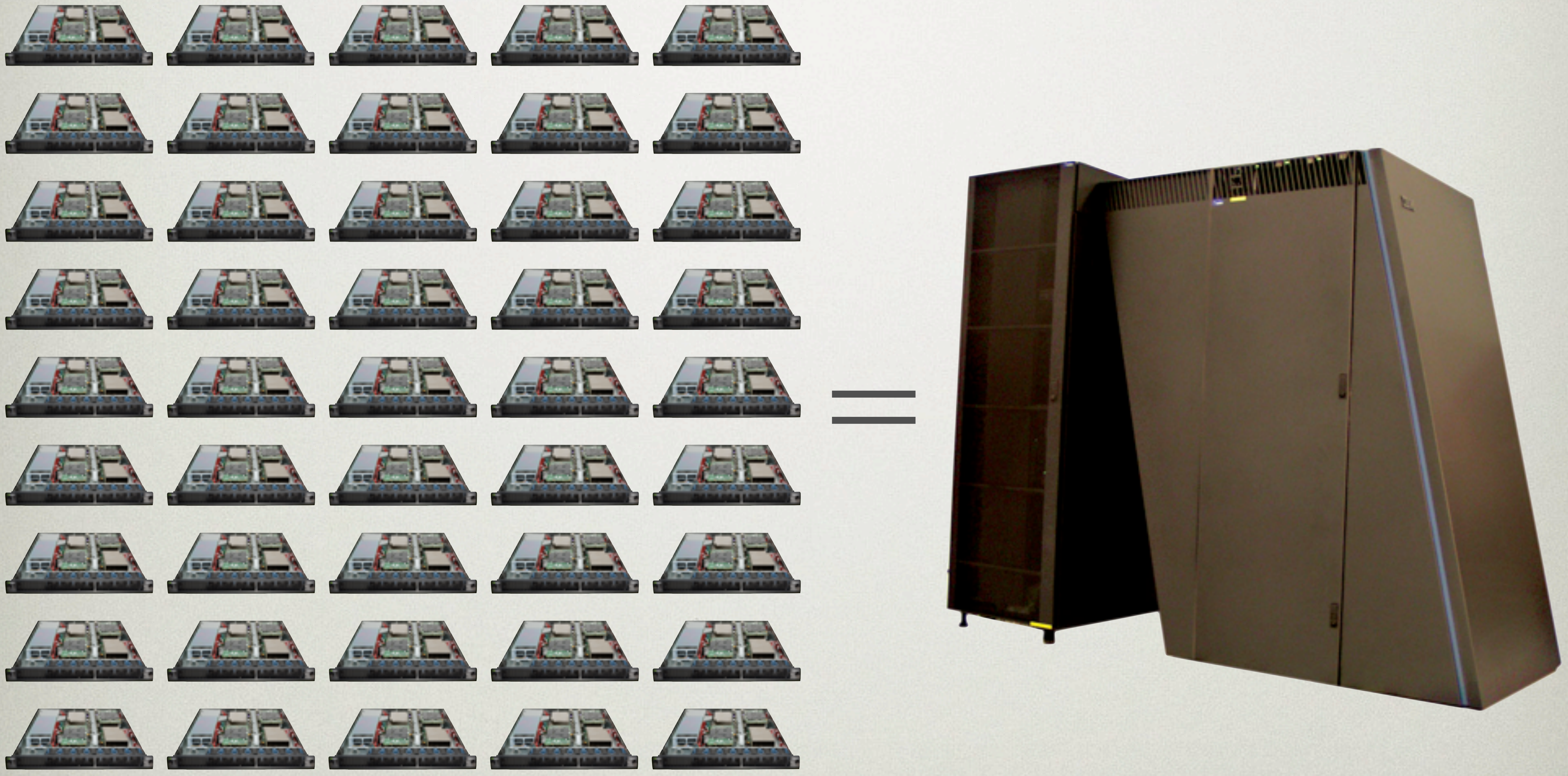
Performance Per MFLOP



Performance Per Watt



Performance Per $



Case Study: Jlab

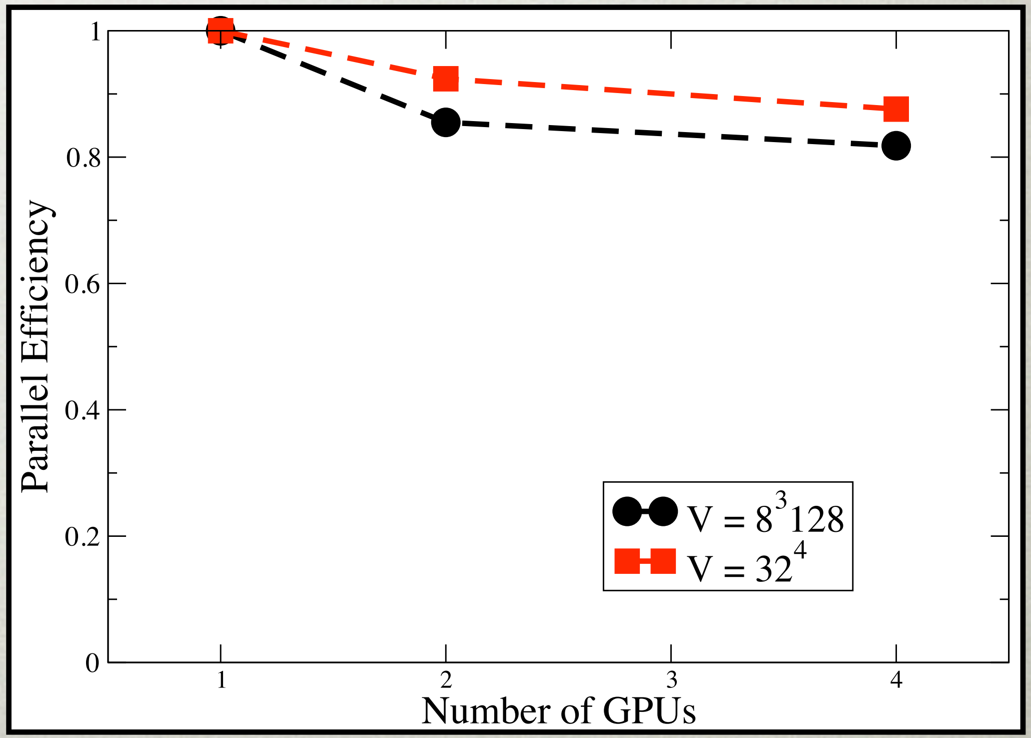
- Require 150 Gflops sustained performance per job
- Nehalem-Infiniband cluster
  - 8 nodes
  - \$0.20 per Mflop
- GPU solution
  - 4x GTX 285 per Nehalem box
  - Trivial parallelism
  - \$0.02 per Mflop
  - 30 Tflops sustained

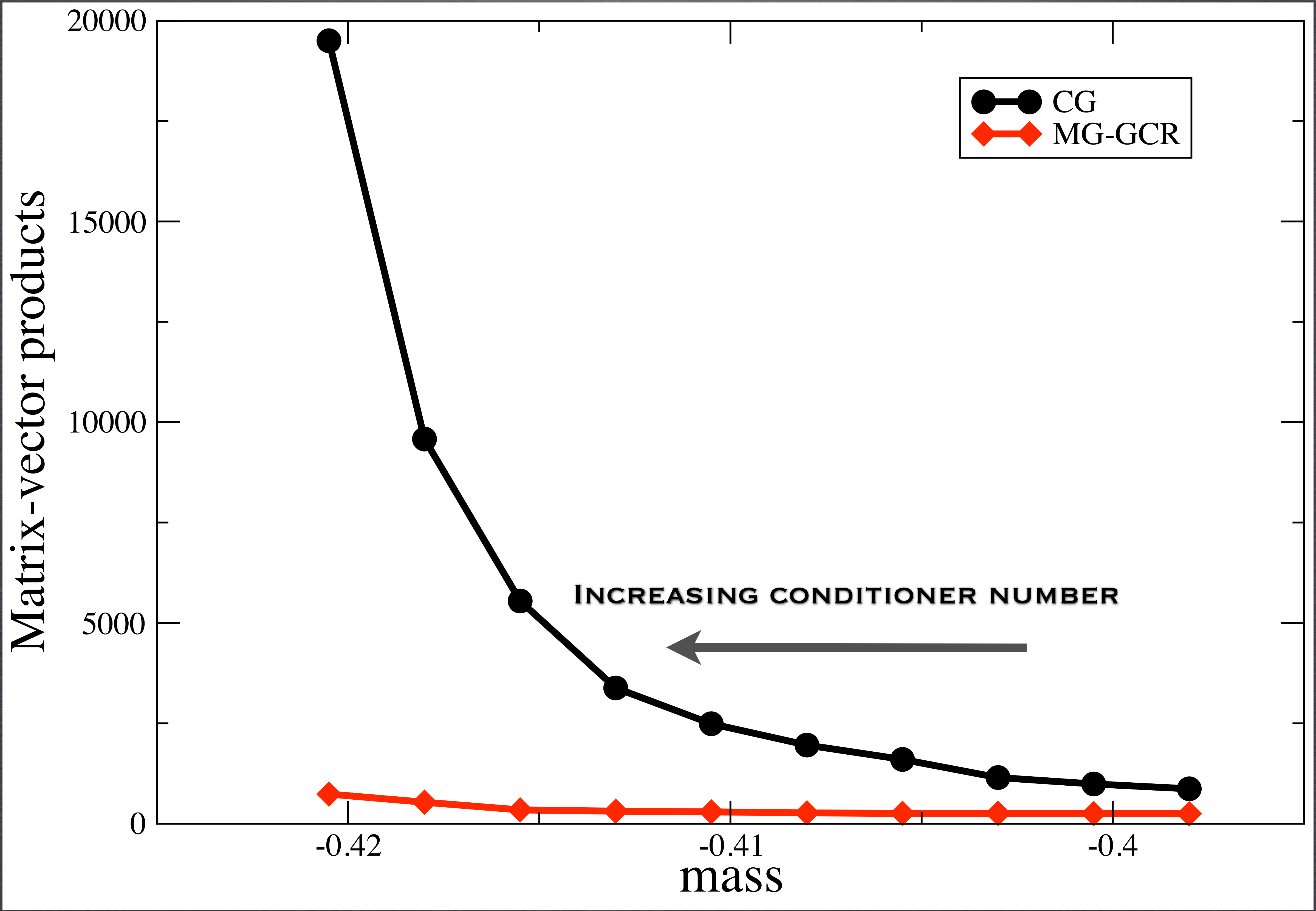
Multi-GPU

- Need to scale to many GPUs
  - Size of problem
  - Raw flops
- Preliminary implementation
  - No overlap of comms and compute
  - 1 MPI process per GPU
  - 90% efficiency on 4 GPUs (S1070)
- Many GPUs challenging but possible
  - 1 GPU per PCIe slot
  - New algorithms

Multi-GPU

- Need to scale to many GPUs
  - Size of problem
  - Raw flops
- Preliminary implementation
  - No overlap of comms and compute
  - 1 MPI process per GPU
  - 90% efficiency on 4 GPUs (S1070)
- Many GPUs challenging but possible
  - 1 GPU per PCIe slot
  - New algorithms

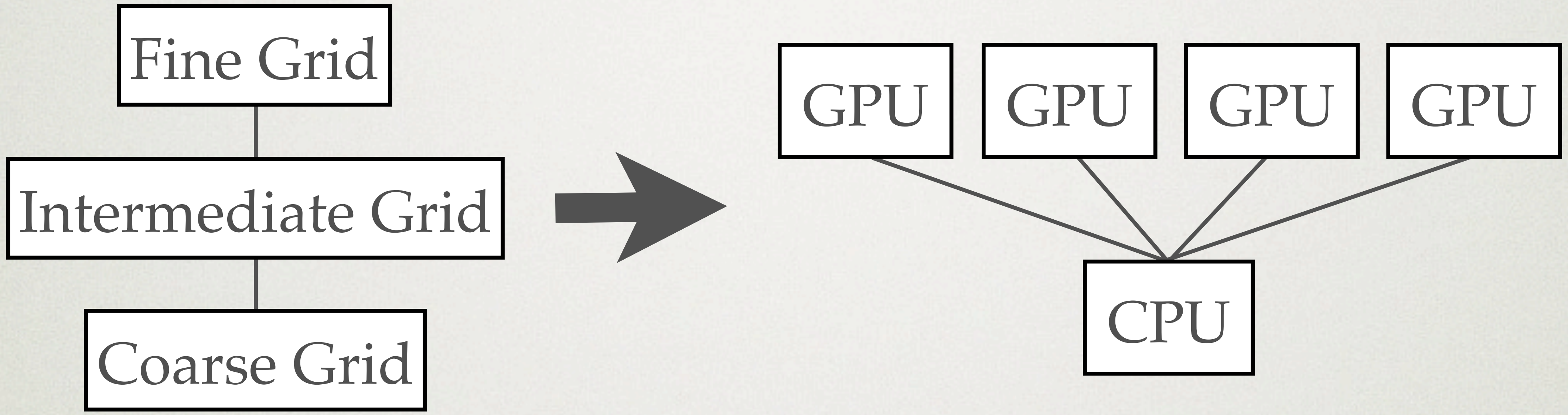




Iterations Until convergence: MG vs CG

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

Multigrid on GPU



- Poor mapping of multigrid onto MPP architectures
- Heterogenous Algorithm => Heterogenous Architecture
  - Fine and intermediate grid operations performed on GPU
  - Coarse grid operators performed on CPU
  - GPU + CPU combination ideal for multigrid

Conclusions

- Fantastic algorithmic performance obtained on today GPUs
  - Flops per Watt, Flops per $
- Game changer for QCD
  - Fermi expected to be even more so
- Some work required to get best performance
  - Standard libraries were not an option
  - Knowledge of the problem required
- Algorithm design critical component
  - Flops are free, bandwidth is expensive
  - Mixed-precision methods

More Info...

- mikec@seas.harvard.edu
- Source code available
  http://lattice.bu.edu/quda
- Paper online as of today
  http://arxiv.org/abs/0911.3191