Case Study: Quantum Chromodynamics

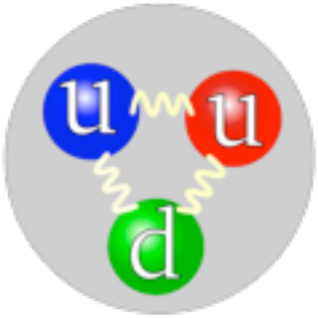
Michael Clark

Harvard University

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

Outline

- Primer to QCD
- QCD on a GPU
- Mixed Precision Solvers
- Multigrid solver on a GPU
- Conclusions



Quantum ChromoDynamics

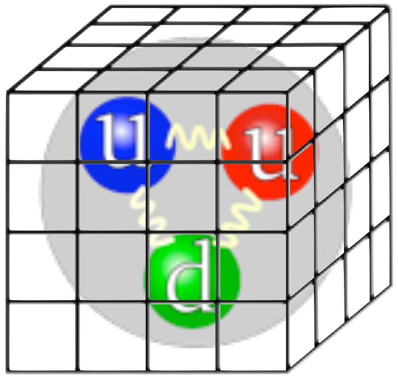
- QCD is the theory of the strong force that binds nucleons
- Impose local SU(3) symmetry on vacuum
  - Color charge analogous to electric charge of EM
- Lagrangian of the theory very simple to write down

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

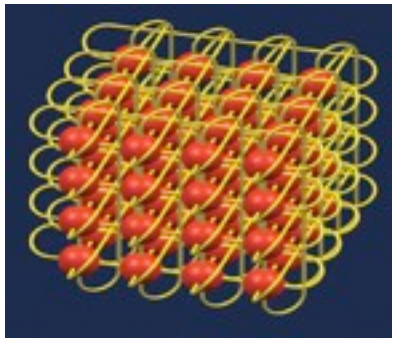
- Path integral formulation

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

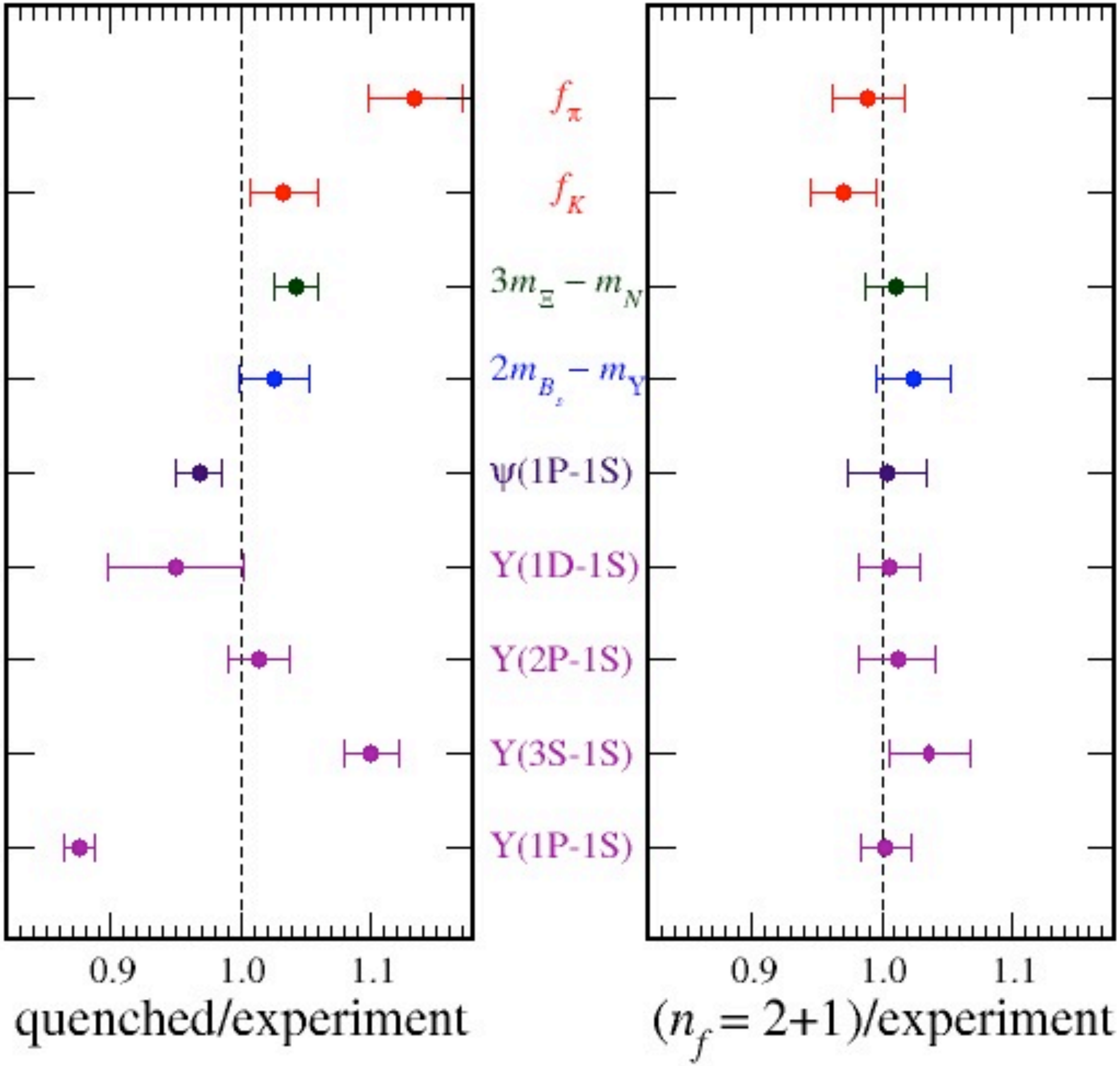
- Infinite dimensional integral
- Theory is strictly non-perturbative at low energies



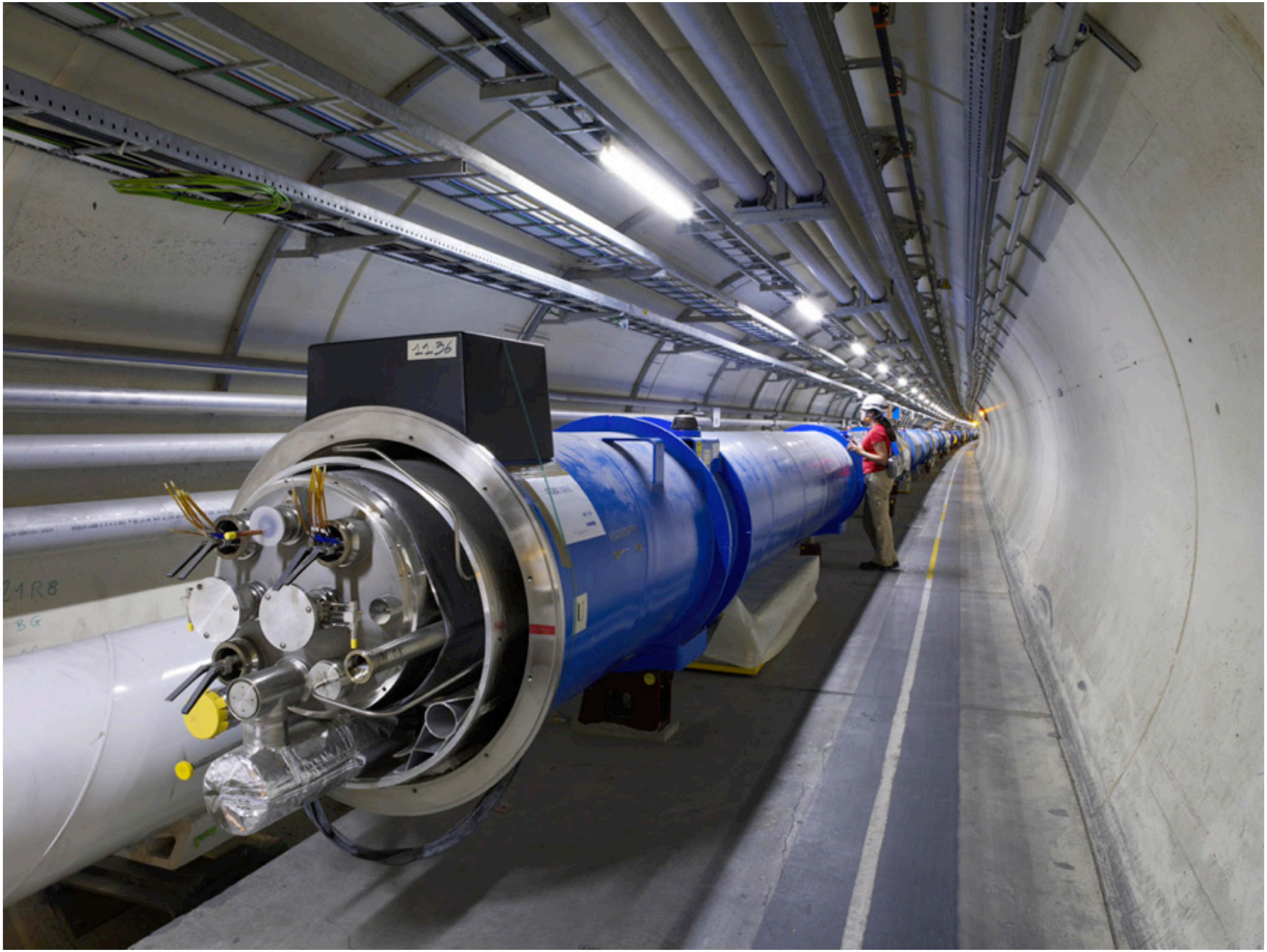
Lattice QCD



- 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



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

Lattice QCD

- Requires Peta/Exaflops: Grand Challenge Problem
- Computation dominated by solving system of linear equations

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

b is the source (vector), **x** the solution and A a sparse NxN matrix

- 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 $O(N) - O(N^2)$
  - Cost dominated by sparse matrix-vector product
- Consider Krylov methods and Multigrid on GPUs

What is A?

- From the QCD Lagrangian

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

Dirac operator of QCD

- The Dirac operator represent quark interactions

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

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

Wilson Matrix of QCD

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

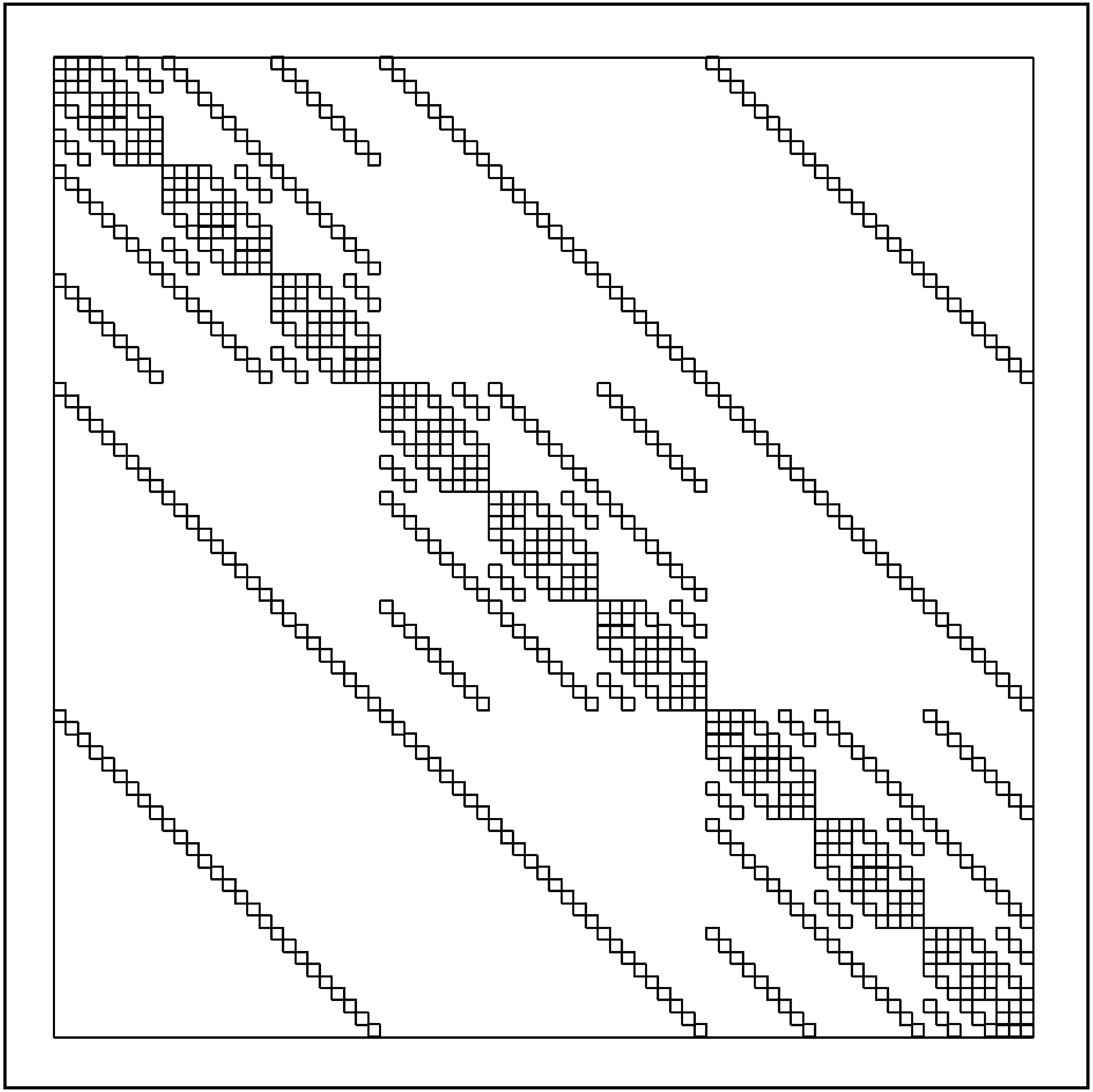
Wilson Matrix of QCD

$$\frac{1}{2}\sum_{\mu}\left(P^{-\mu}\otimes U^{\mu}_{x,y}\delta_{x+\mu,y}+P^{+\mu}\otimes U^{\mu\dagger}_{y,x}\delta_{x-\mu,y}\right)+(4+m)\delta_{x,y}$$

Wilson Matrix of QCD

$$\frac{1}{2}\sum_{\mu}\left(P^{-\mu}\otimes U^{\mu}_{x,y}\delta_{x+\mu,y}+P^{+\mu}\otimes U^{\mu\dagger}_{y,x}\delta_{x-\mu,y}\right)+(4+m)\delta_{x,y}$$

- U is discretized gauge field (SU(3))
- *P* are Dirac spin projector matrices (4x4)
- 8 off-diagonals in spacetime, mass on diagonal
  - Off-diagonals are 12x12 complex matrices
- Each point in spacetime referred to as a *spinor*
  - 12 complex component vector
- Matrix not Hermitian but γ_5-Hermitian



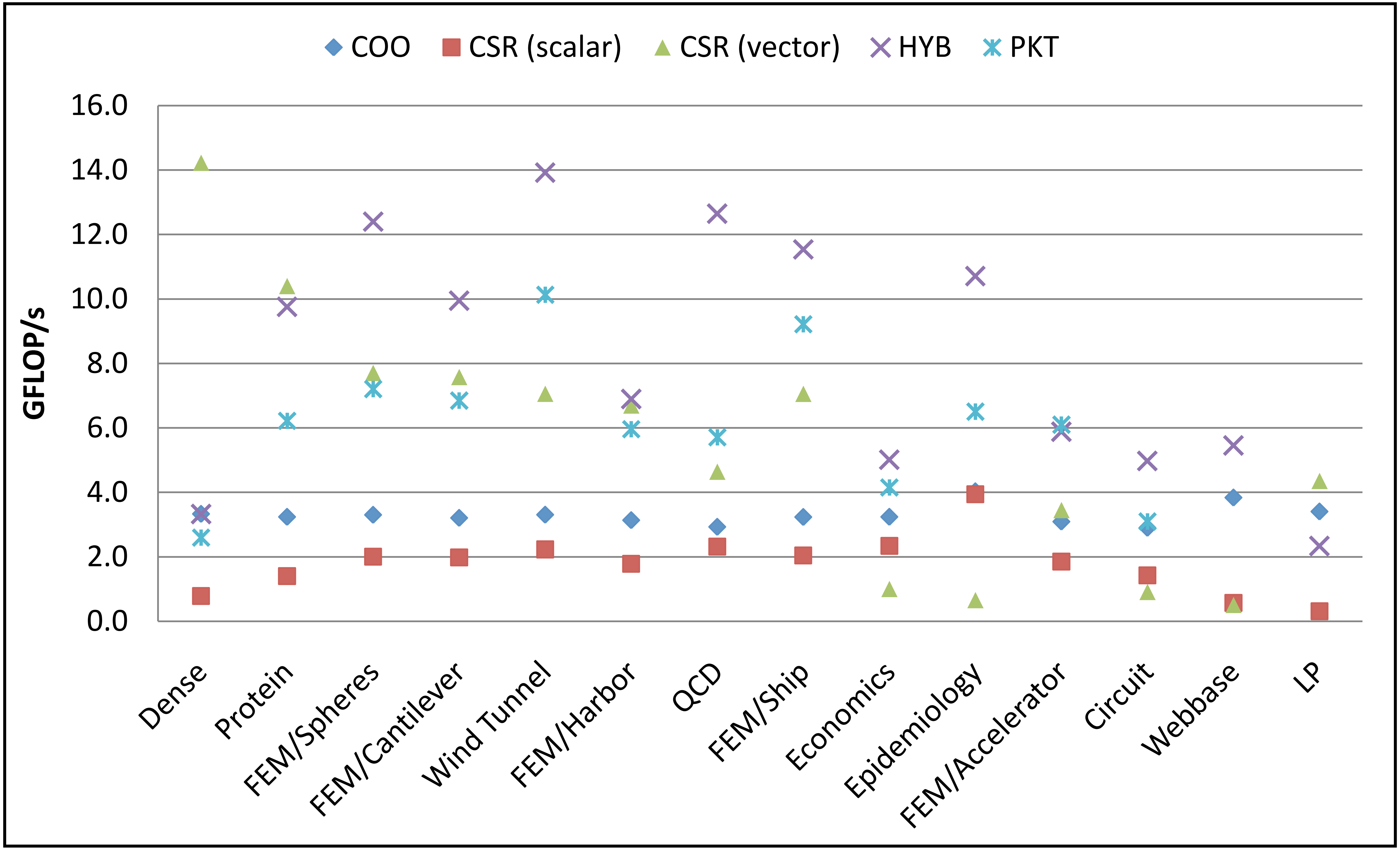
Wilson Matrix of QCD

$$\frac{1}{2}\sum_{\mu}\left(P^{-\mu}\otimes U^{\mu}_{x,y}\delta_{x+\mu,y}+P^{+\mu}\otimes U^{\mu\dagger}_{y,x}\delta_{x-\mu,y}\right)+(4+m)\delta_{x,y}$$

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

Explicit Matrix

- Possible to store matrix in explicit sparse format (CSR, etc.)
- Standard matrix-vector libraries available
  - Pack your matrix, call library function, unpack
  - Problem solved?



Bell and Garland (NVIDIA) 2008

Explicit Matrix

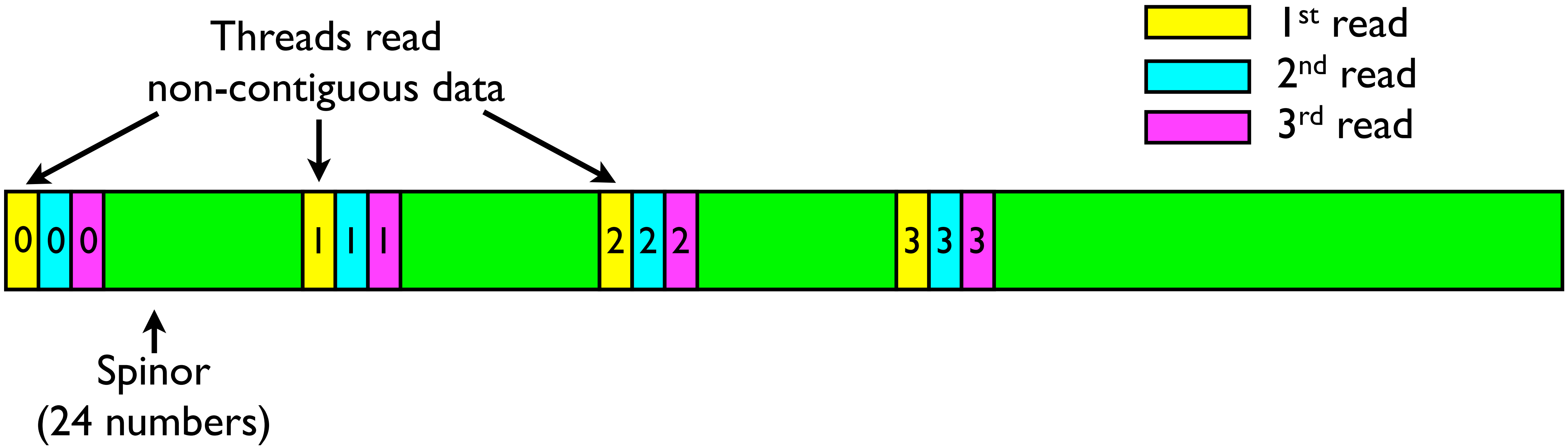
- Possible to store matrix in explicit sparse format (CSR, etc.)
- Standard matrix-vector libraries available
  - Pack your matrix, call library function, unpack
  - Problem solved?
- Ignorant of structure and symmetries of problem
  - Bad for storage (double storage of U)
  - Bad for memory coalescing
  - Bad for memory traffic (9408 bytes per site)
  - Bad for operation count (4680 flops per site)

GPU Operator Representation

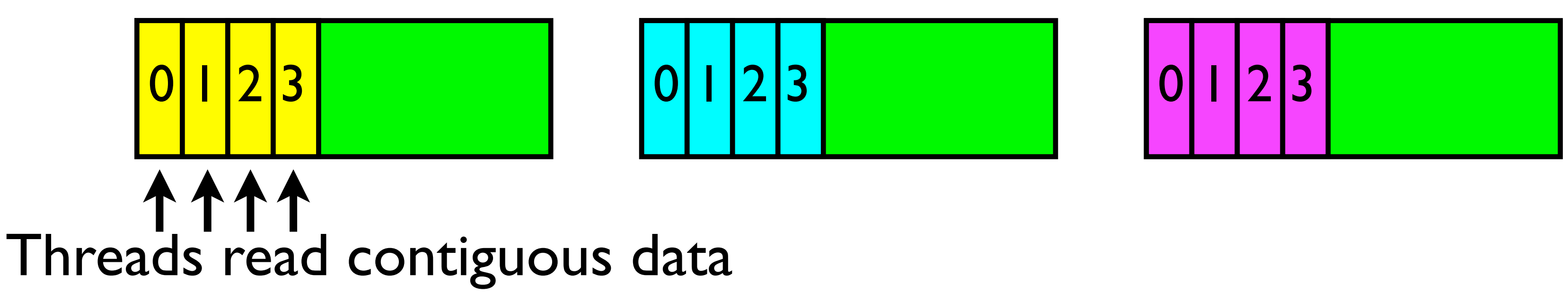
- Much better to consider matrix as a nearest neighbor gather operation (stencil)
  - Avoids double storage of matrix elements (Hermiticity)
  - Repetitive structure means no explicit indexing required
- Threads must be lightweight
  - Assign a single space-time point to each thread -> XYZT threads
  - Must use shared memory and registers for high occupancy (256 threads)
- Can order data optimally for any given hardware
- Large reduction in storage, flops and memory traffic
  - 1440 bytes per site (c.f. 9408)
  - 1368 flops per site (c.f. 4680)

Memory Layout

- Typical CPU spinor field ordering: contiguous array of 24 floats



- Reorder fields for coalescing: 6x array of float4s



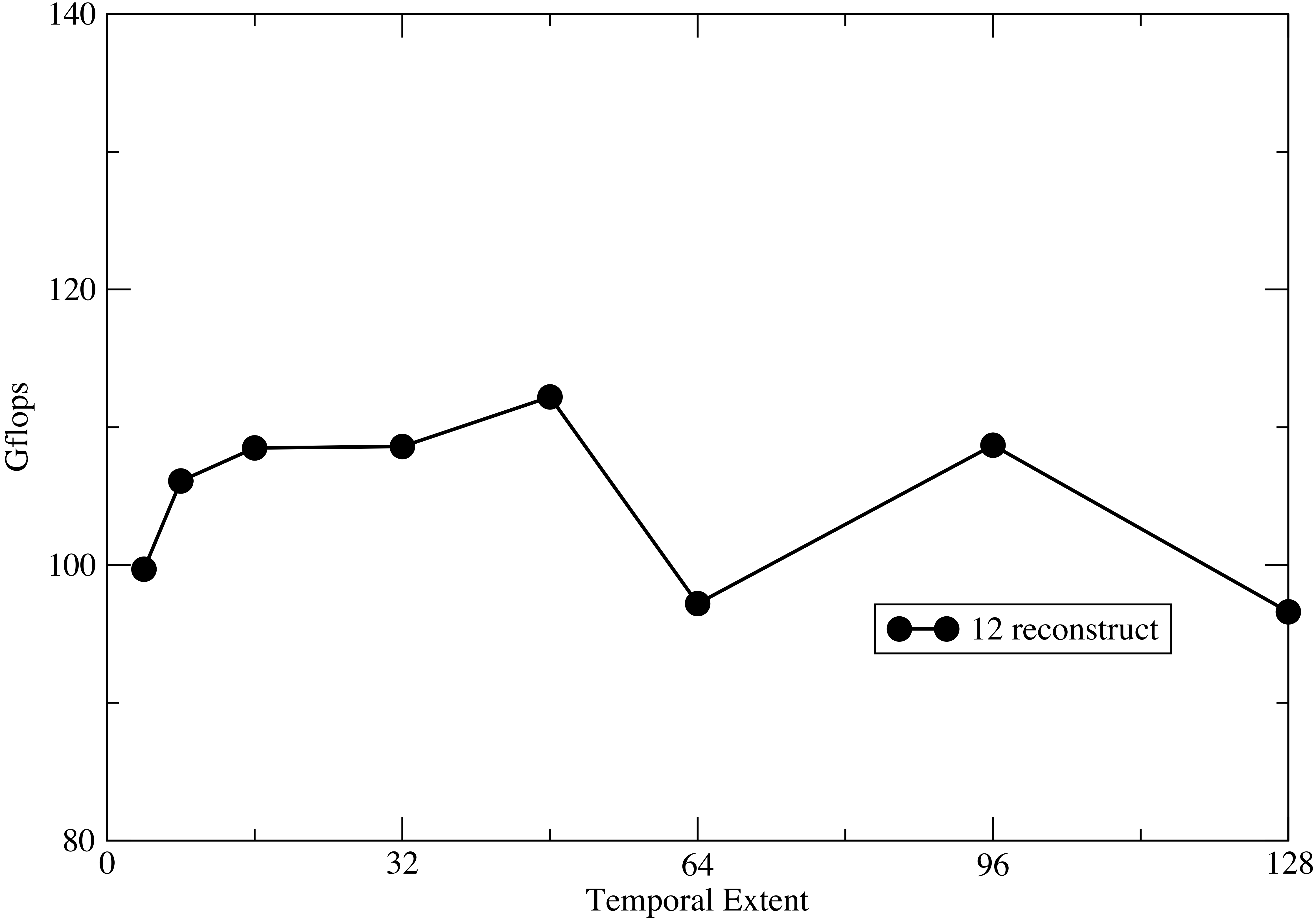
- Similar reordering required for matrix elements

SU(3) Representation

- SU(3) matrices are all unitary complex matrices with det = 1
  - 18 real numbers, but only 8 free parameters (generators)
- 12 number parameterization

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

- Reconstruct full matrix on the fly
- 1152 Bytes per site
- Additional 384 flops per site



Wilson Matrix-Vector Performance

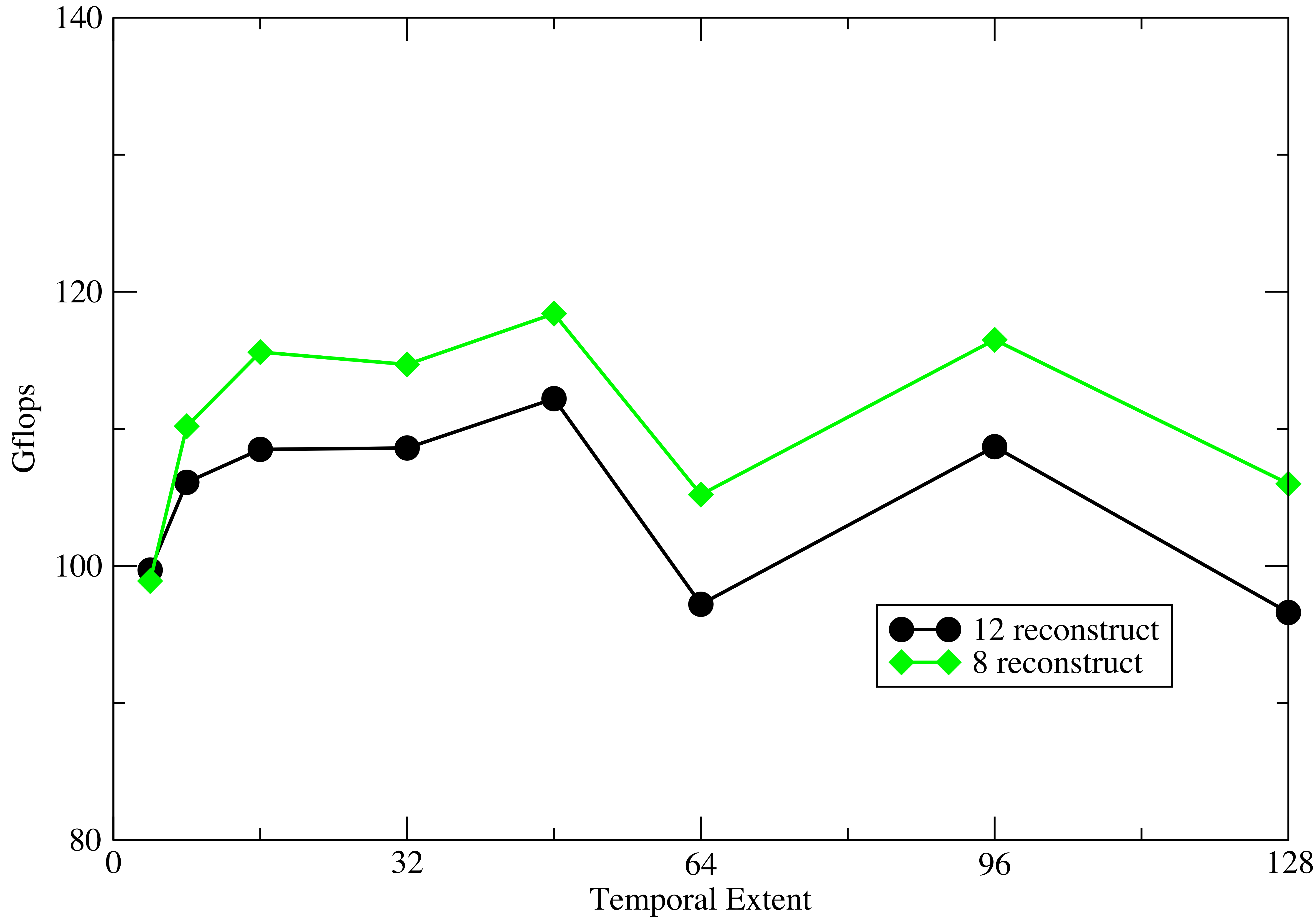
Single Precision ($V=24^3xT$)

SU(3) Representation

- Minimal 8 number parameterization

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

  - Obtain a_1 and c_1 from normality
  - Reconstruct b_2, b_3, c_2, c_3 from SU(2) rotation
- 1024 Bytes per site
- Additional 856 flops per site
  - Including 2 sqrt, 4 trigonometric, 2 divide



Wilson Matrix-Vector Performance

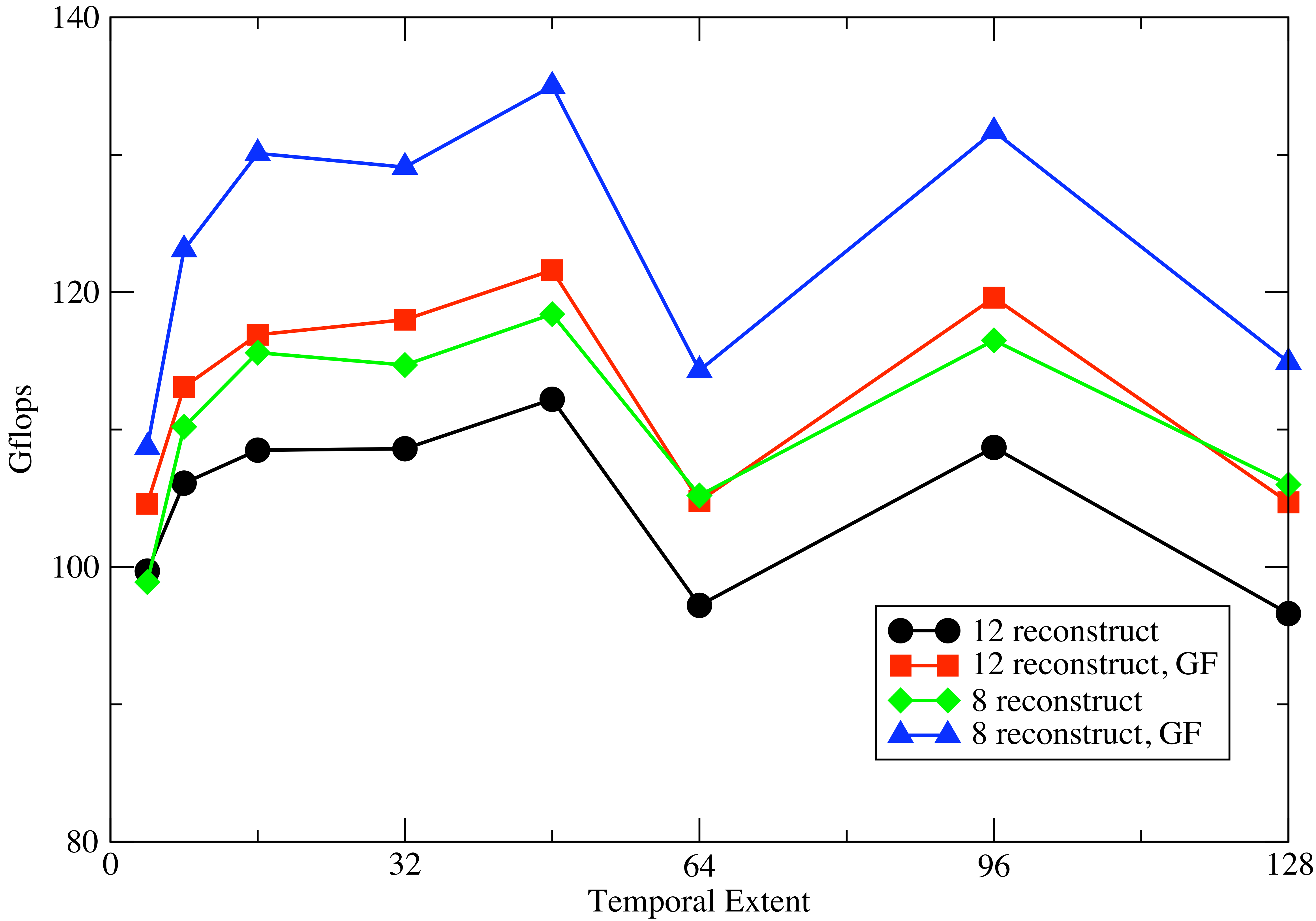
Single Precision ($V=24^3 xT$)

More tricks

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

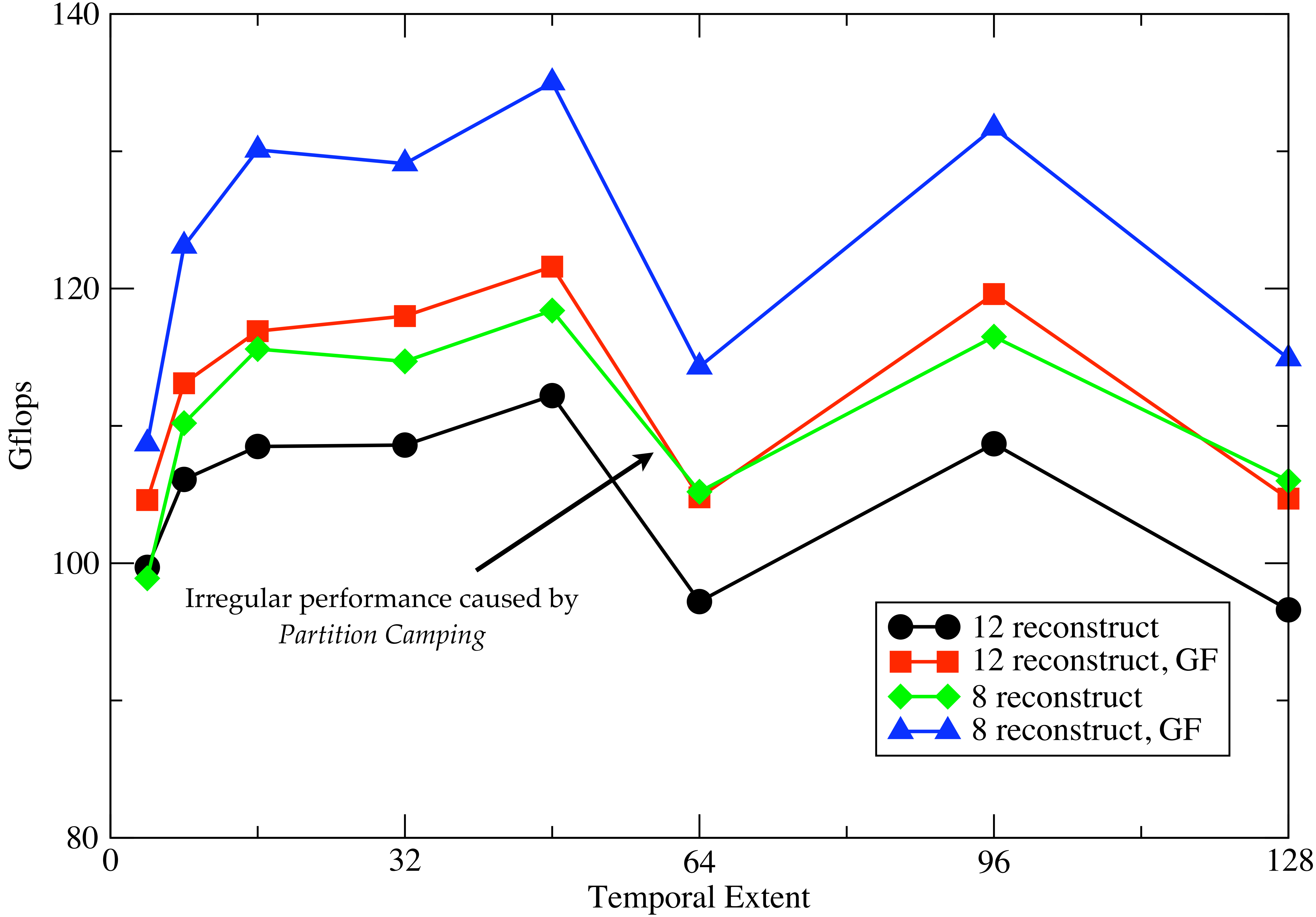
$$P^{\pm 4} = \begin{pmatrix} 1 & 0 & \pm 1 & 0 \\ 0 & 1 & 0 & \pm 1 \\ \pm 1 & 0 & 1 & 0 \\ 0 & \pm 1 & 0 & 1 \end{pmatrix} \longrightarrow P^{+4} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} P^{-4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0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

- Impose local color transformation (gauge transformation)
  - SU(3) field = unit matrix in temporal direction
  - Must calculate this transformation (done once only)
- 960 Bytes per site (c.f. 1440)
- In total: 33% bandwidth reduction



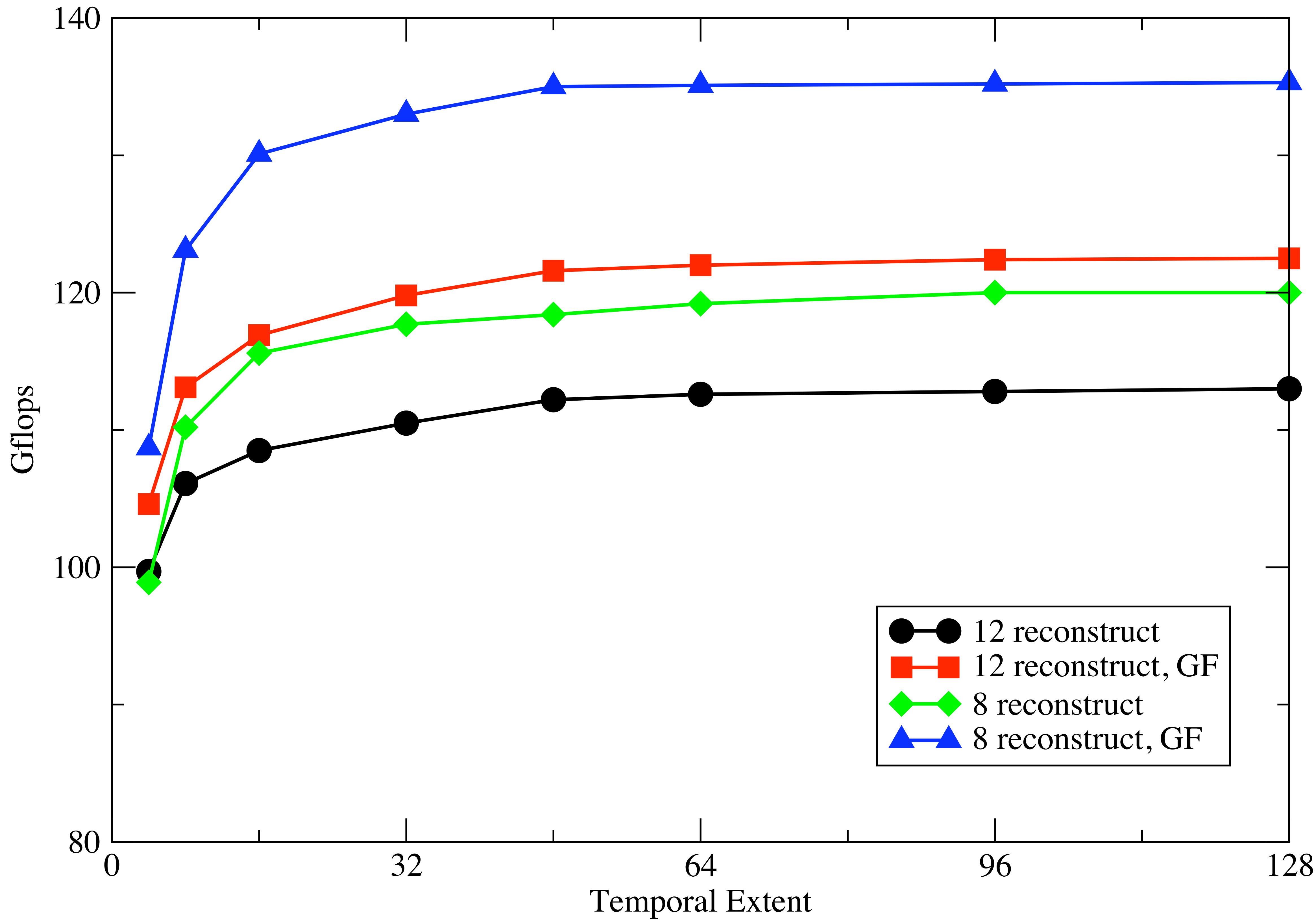
Wilson Matrix-Vector Performance

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



Wilson Matrix-Vector Performance

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

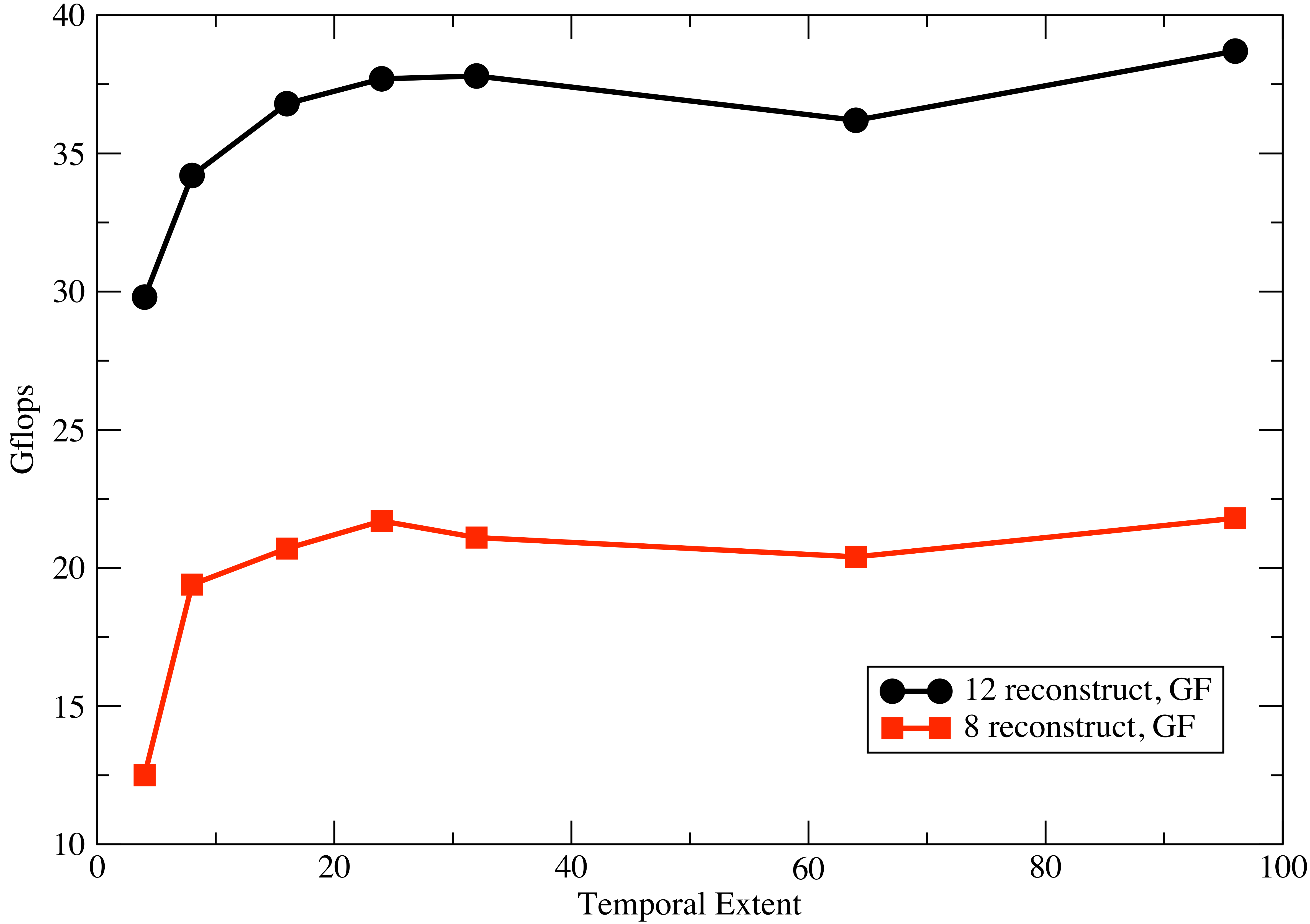


Wilson Matrix-Vector Performance

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

Double Precision

- Double precision peak ~ 78 Gflops
  - Flop / Bandwidth ratio much more forgiving
- Find and replace float -> double
  - Order fields using double2 primitive for coalescing
- Register and shared memory pressure an issue
  - Maximum of 128 concurrent threads
- Not all tricks are useful anymore....
- Performance penalty only a factor ~3 vs. single



Wilson Matrix-Vector Performance

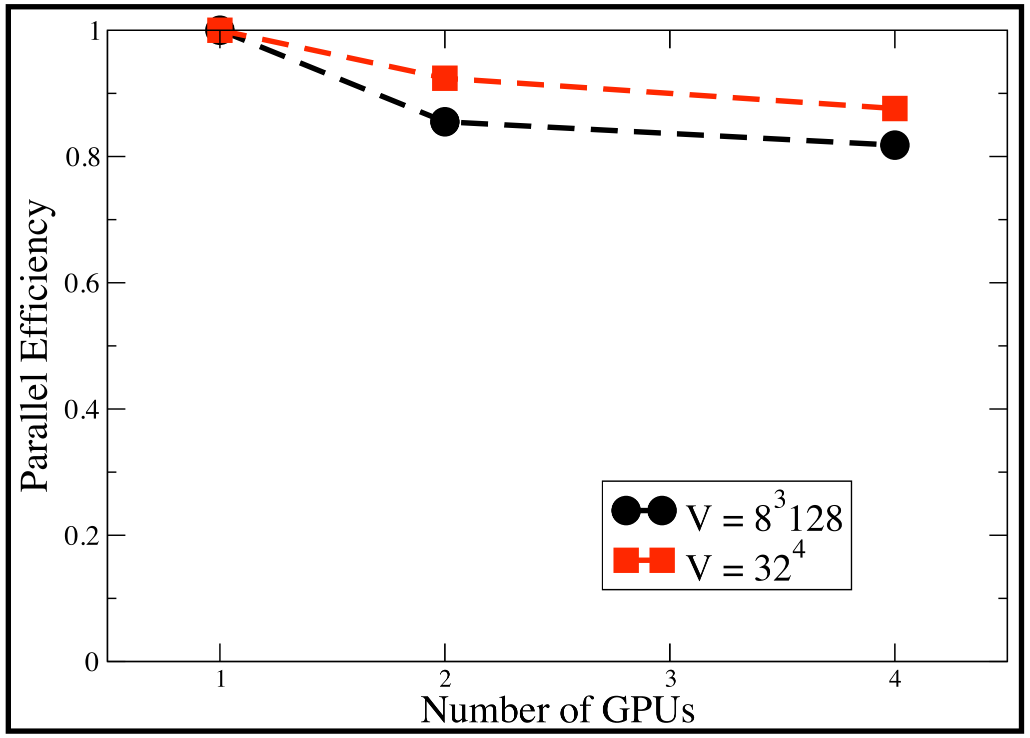
Double Precision ($V = 24^3 x T$)

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

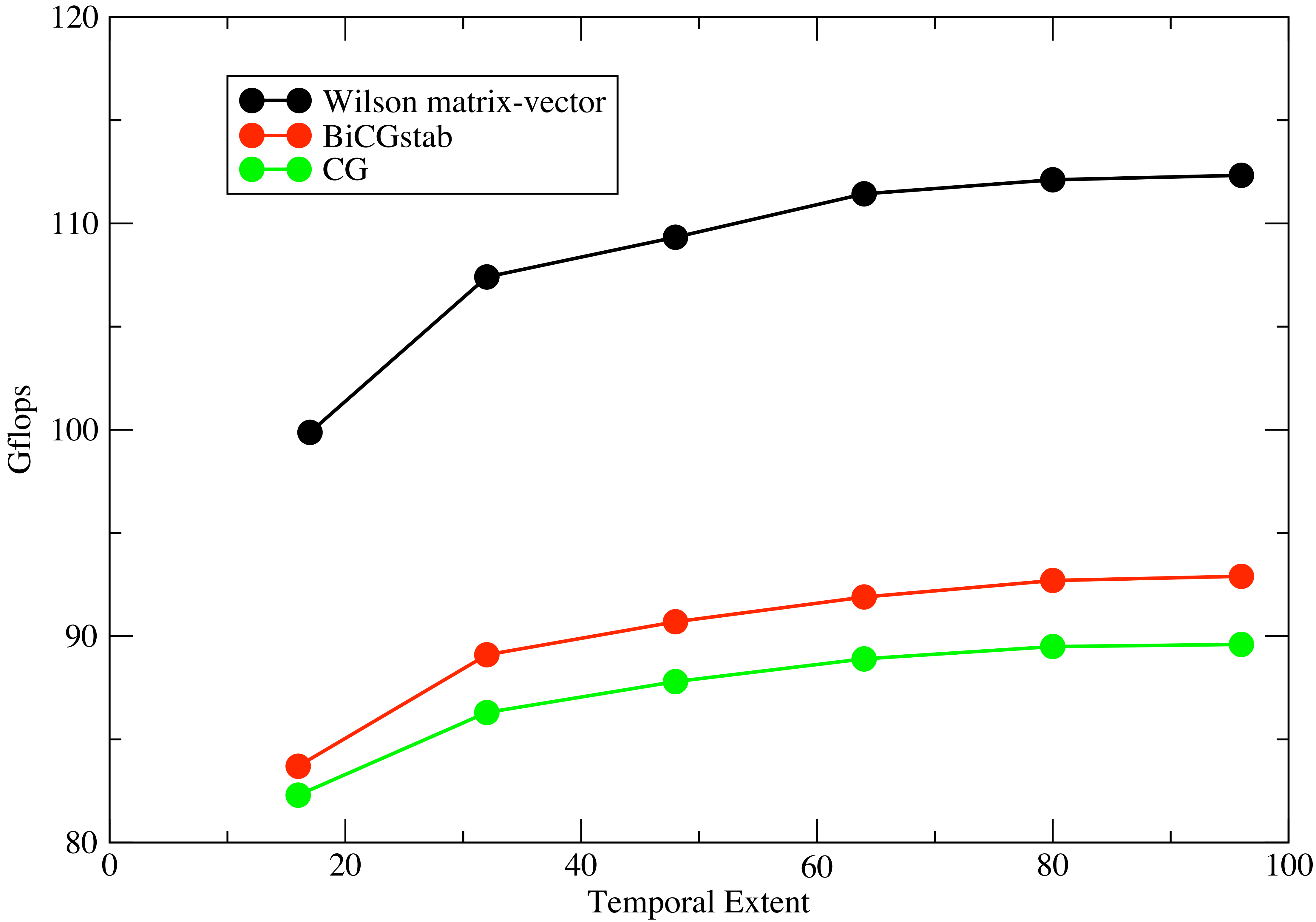


Krylov Solver Implementation

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

```
while (|r_k|> ε) {
    β_k = (r_k,r_k)/(r_k-1,r_k-1)
    p_k+1 = r_k - β_k p_k

    α = (r_k,r_k)/(p_k+1,Ap_k+1)
    r_k+1 = r_k - αAp_k+1
    x_k+1 = x_k + αp_k+1
    k = k+1
}
```



Wilson Inverter Performance

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

Mixed-Precision Solvers

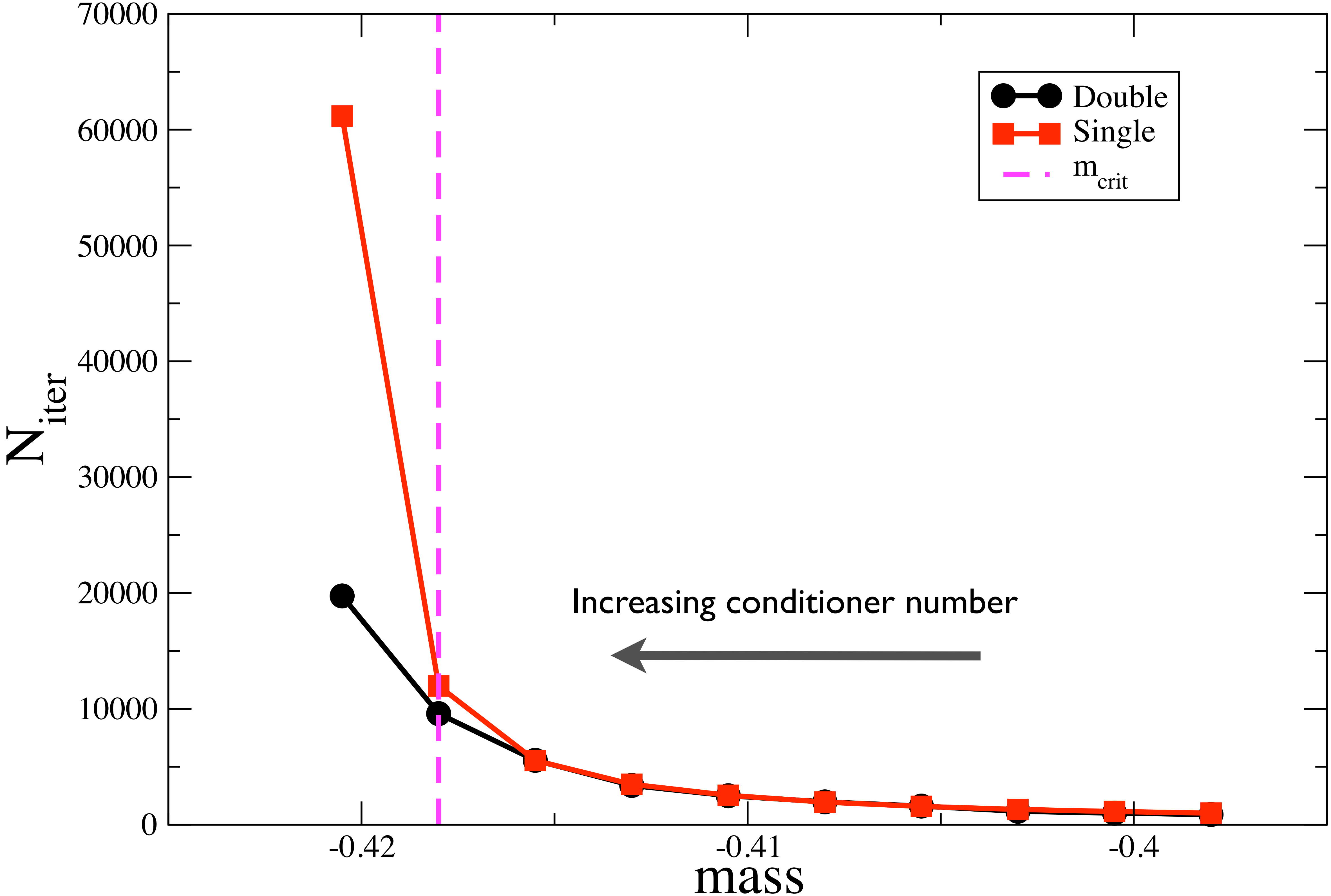
- Require solver tolerance beyond limit of single precision
- e.g., Use defect-correction

Double precision mat-vec and accumulate

$$
\begin{array}{l}
\text{while } (|\mathbf{r}_k| > \varepsilon) \ \{ \\
\quad \mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k \\
\quad \mathbf{p}_k = A^{-1}\mathbf{r}_k \\
\quad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k \\
\}
\end{array}
$$

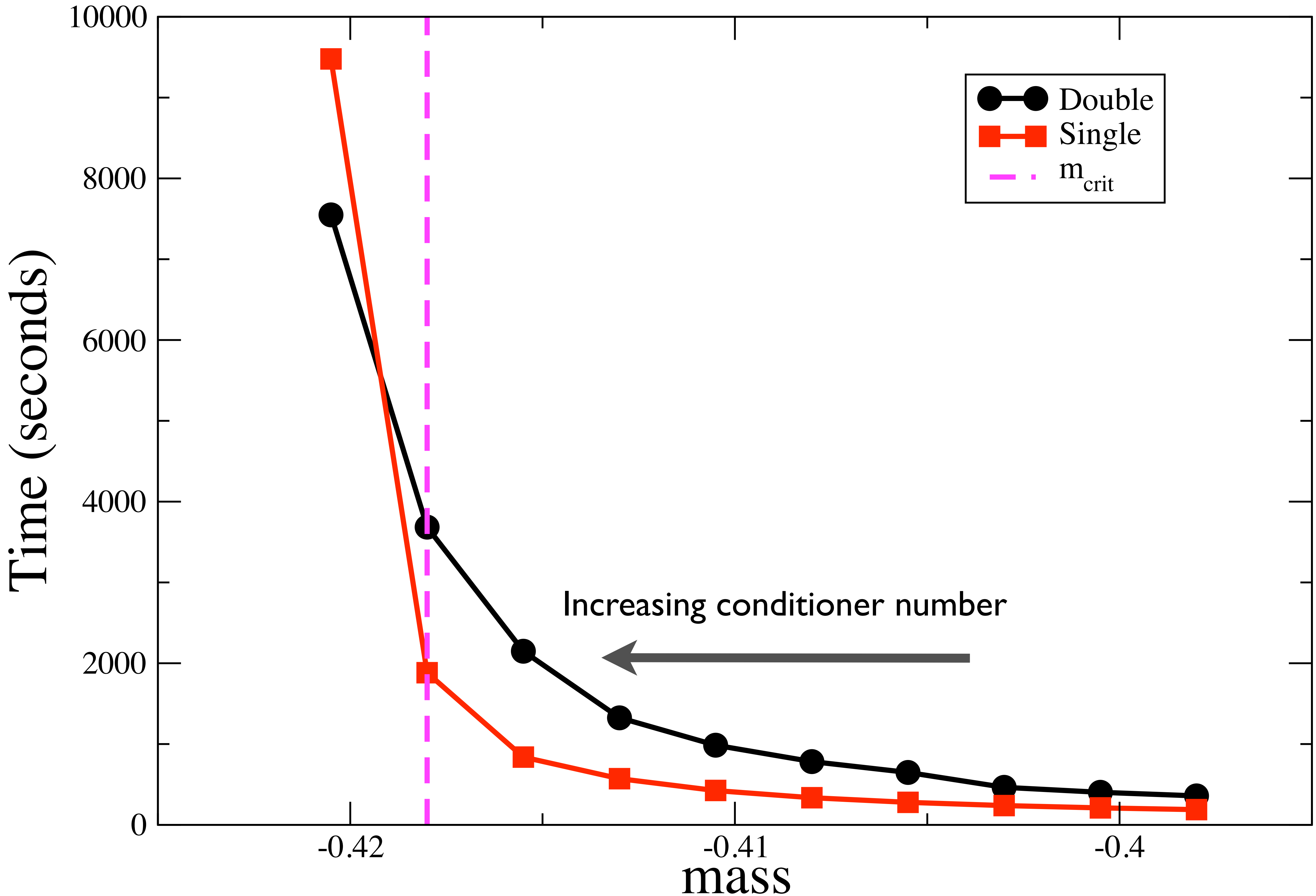
Inner single precision solve

- Double precision can be done on CPU or GPU
  - Can always check GPU gets correct answer
- Disadvantage is each new single precision solve is a restart
  - Use Reliable Updates (Sleijpen and Van der Worst 1996)



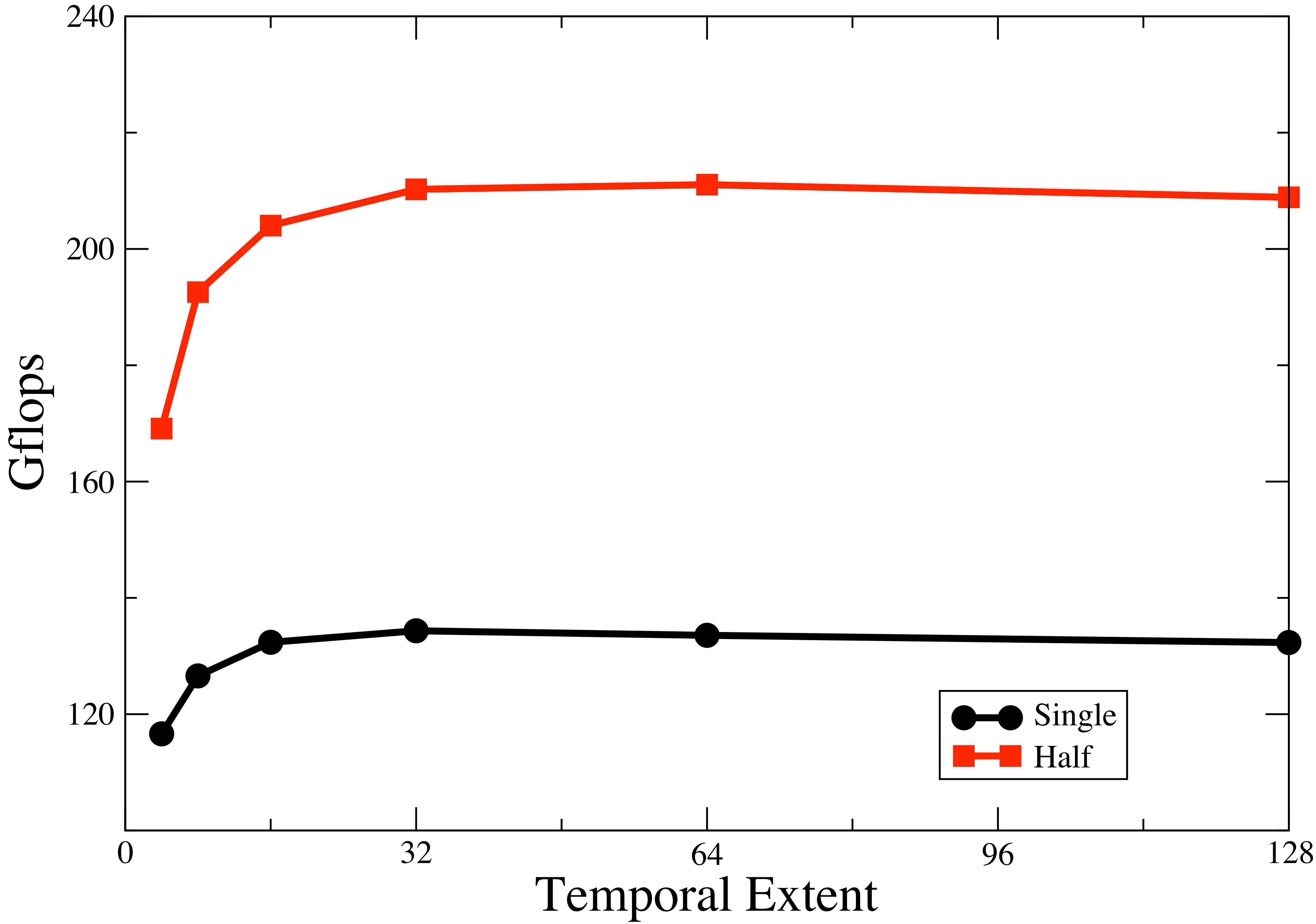
Wilson Inverter Iterations

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



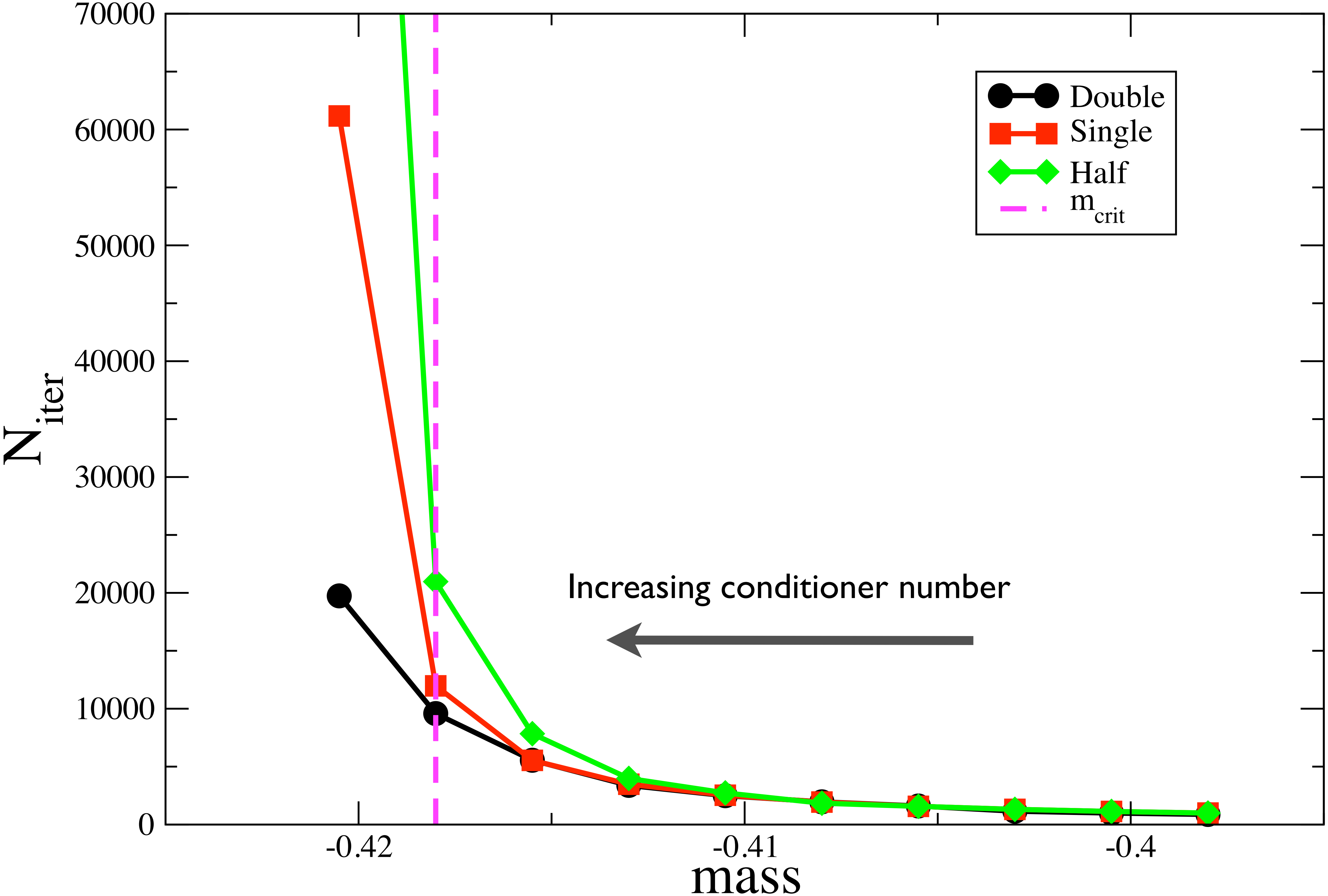
Wilson Inverter Time to Solution

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



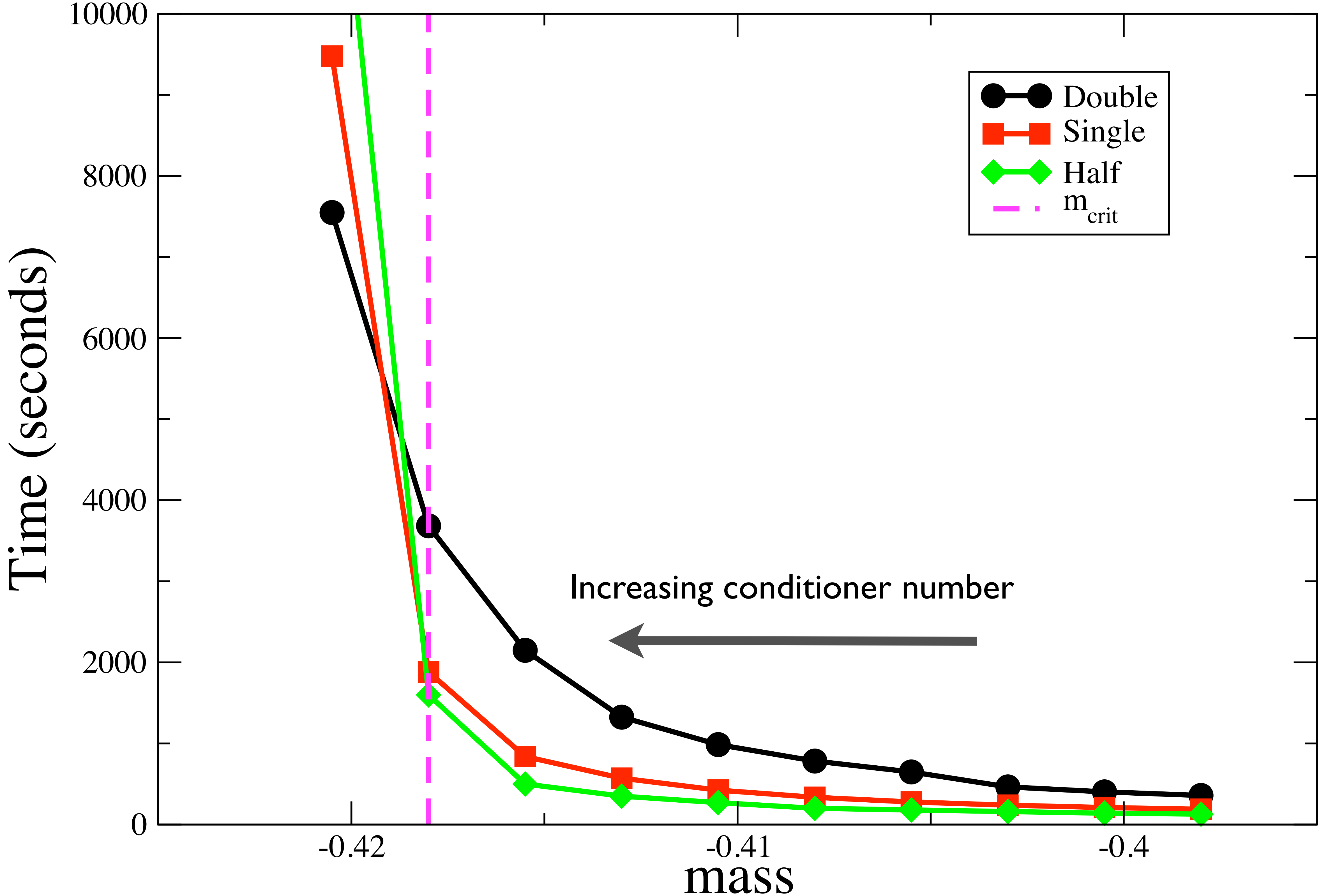
Wilson Matrix-Vector Performance

Half Precision (V = 24^3xT)



Wilson Inverter Iterations

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

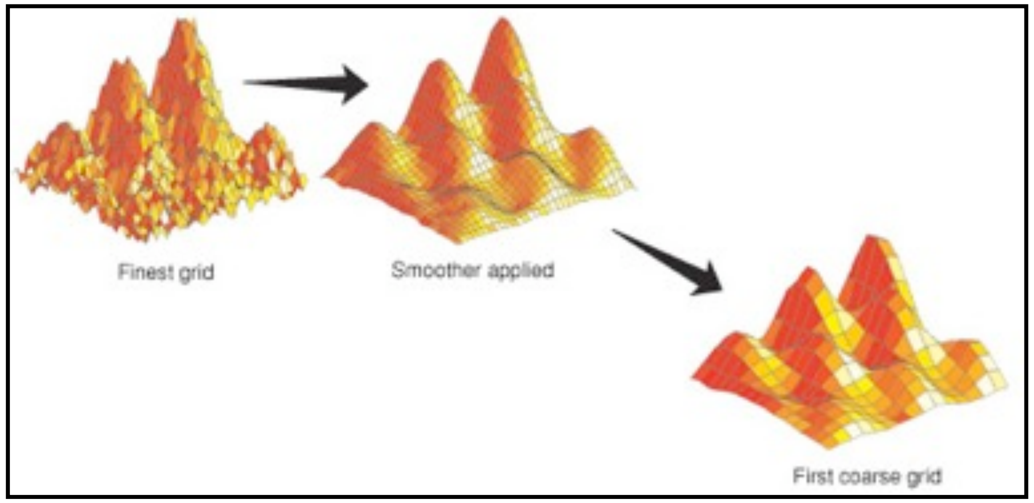


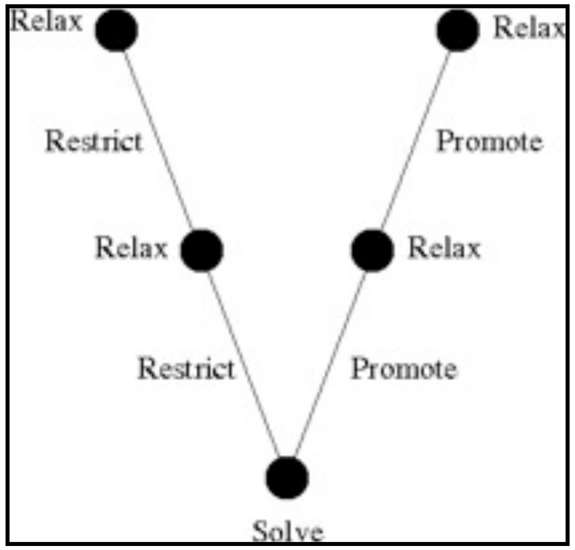
Wilson Inverter Time to Solution

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

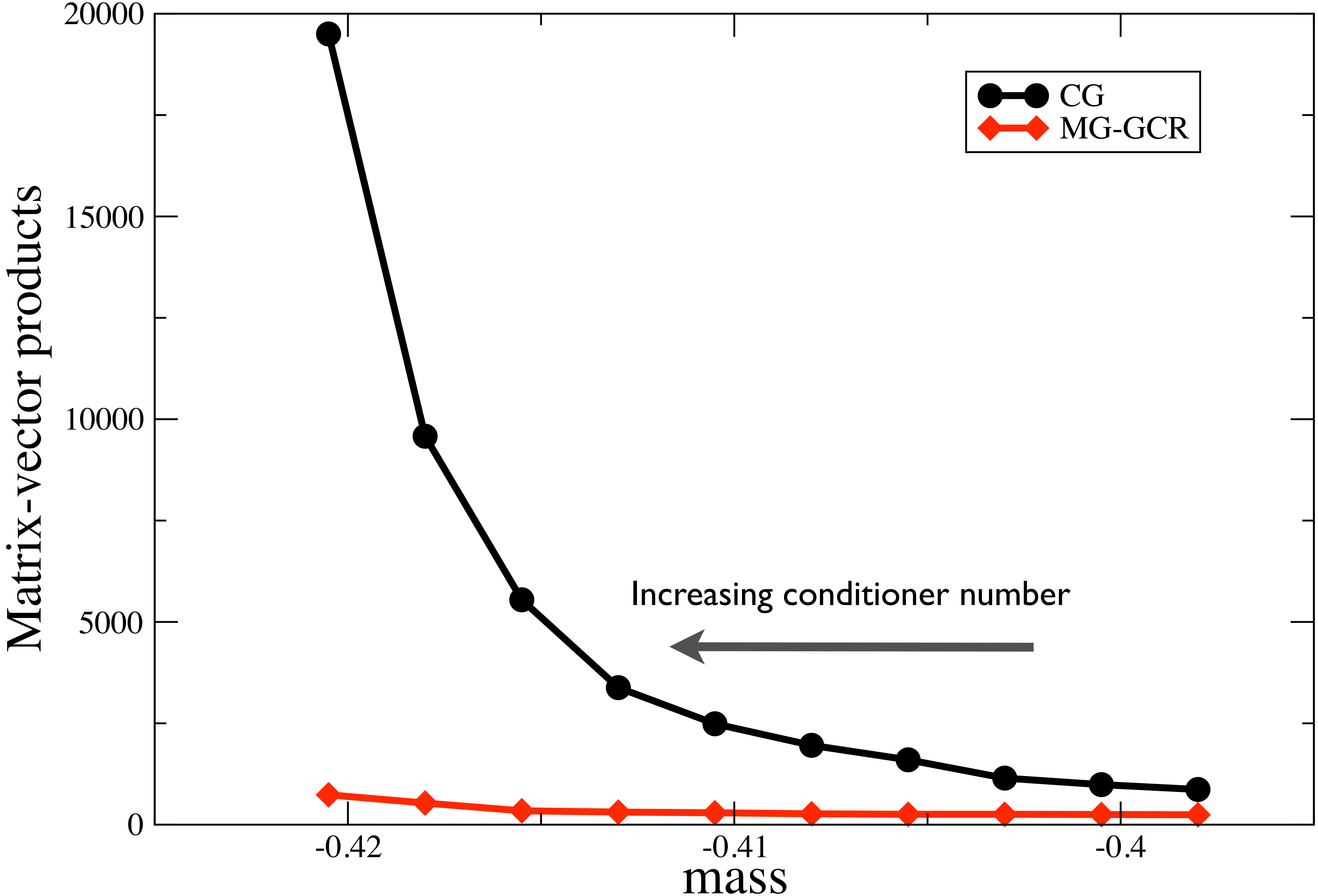
Multigrid Solver

- Use solution on coarse grid to accelerate the solver





- Iterate this process until exact solve is possible (V-cycle)
- Multigrid methods are optimal
  - O(N) scaling
  - No condition number dependence



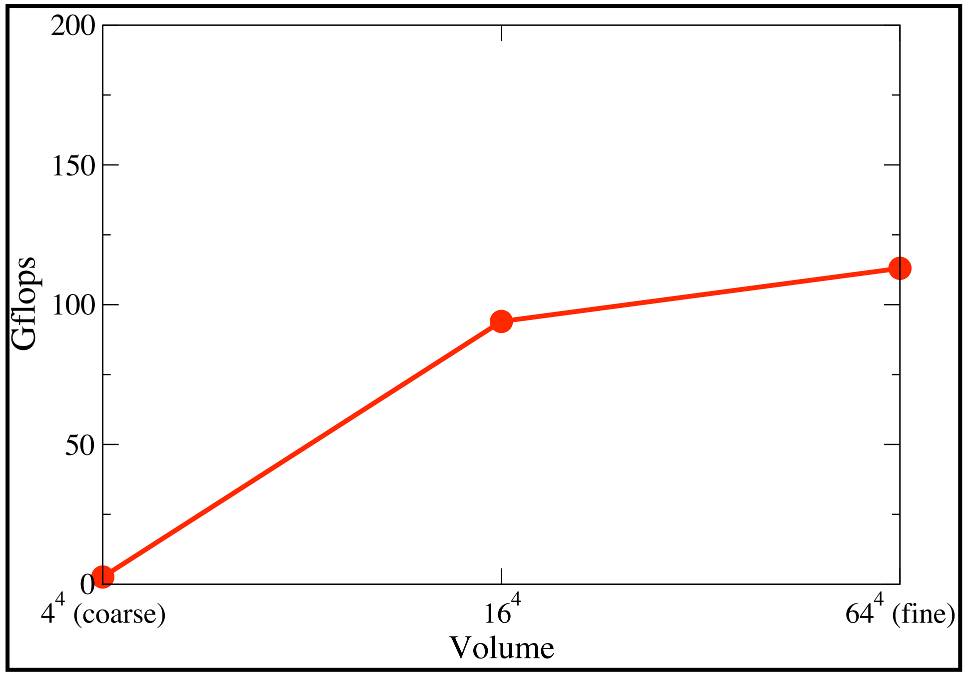
Iterations Until convergence: MG vs CG

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

Multigrid on a GPU

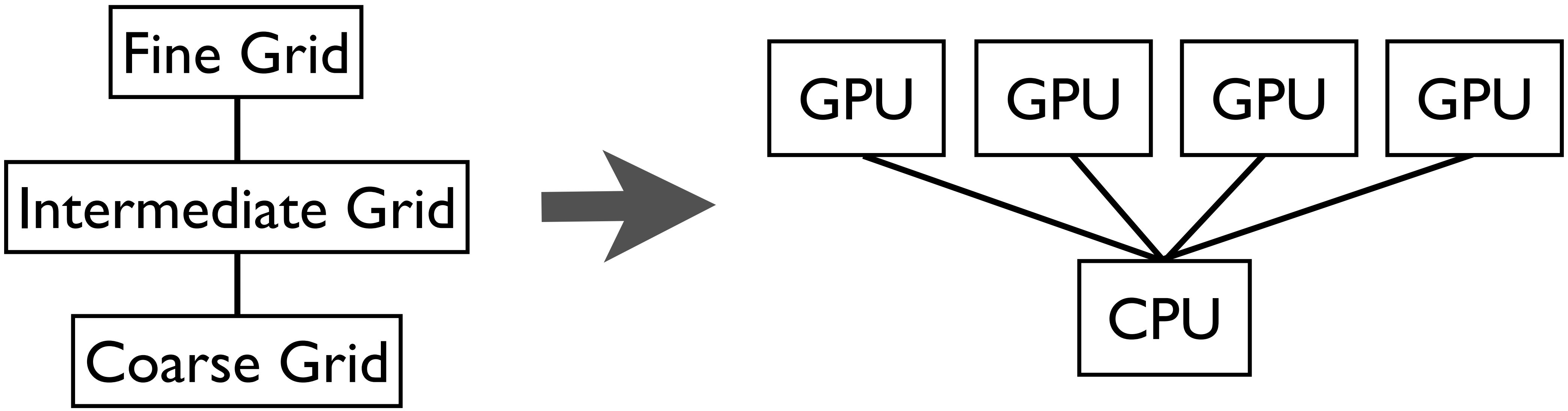
- Very difficult to obtain high performance on parallel architectures
- E.g., $V=64^4$, 3 level multigrid algorithm, 4^4 coarsening

	Fine Grid	Intermediate Grid	Coarse Grid
Volume	64^4	16^4	4^4
Surface / Volume	0.125	0.5	2



- More cores than degrees of freedom
- Efficient multi-GPU impossible on the coarse grid
- Heterogenous Algorithm

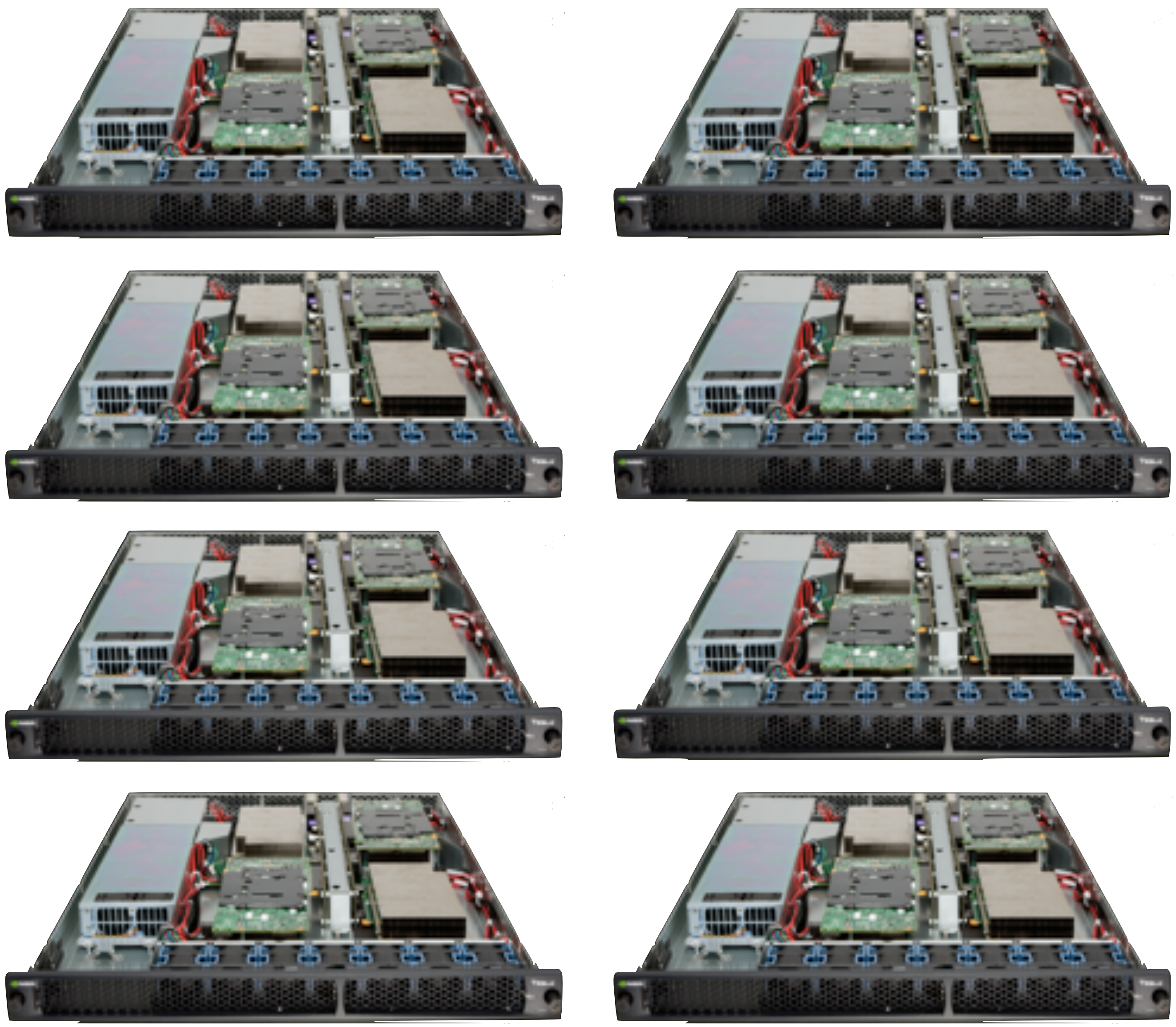
Multigrid on a GPU



- Heterogenous Algorithm => Heterogenous Architecture
  - Fine and intermediate grid operations performed on GPU
  - Coarse grid operators performed on CPU
  - GPU + CPU combination ideal for multigrid
- Mixed precision possible
  - Single/Half precision for multigrid preconditioner
  - Double precision for outer Krylov wrapper

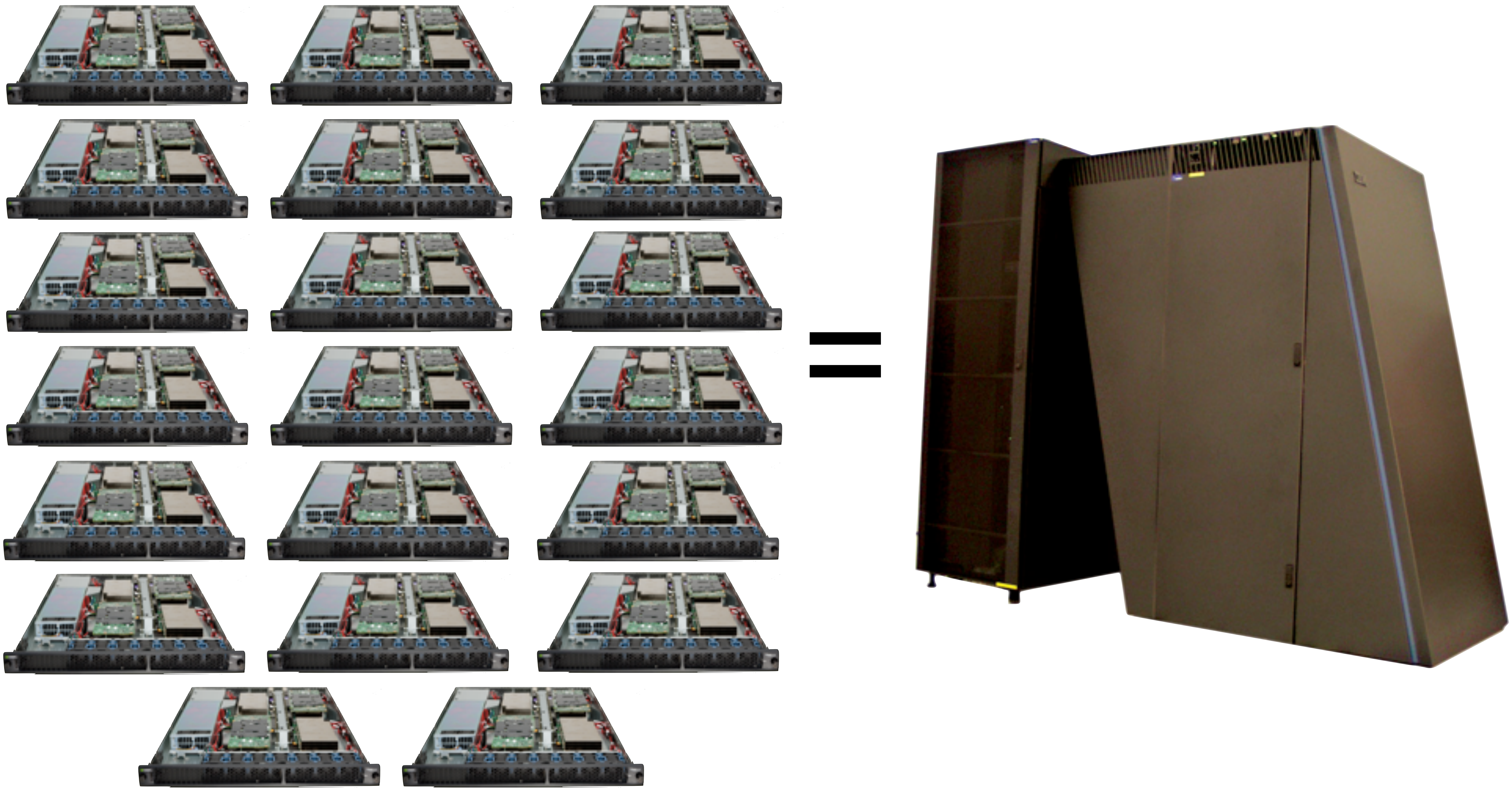
How Fast is Fast?

Performance Per MFLOP

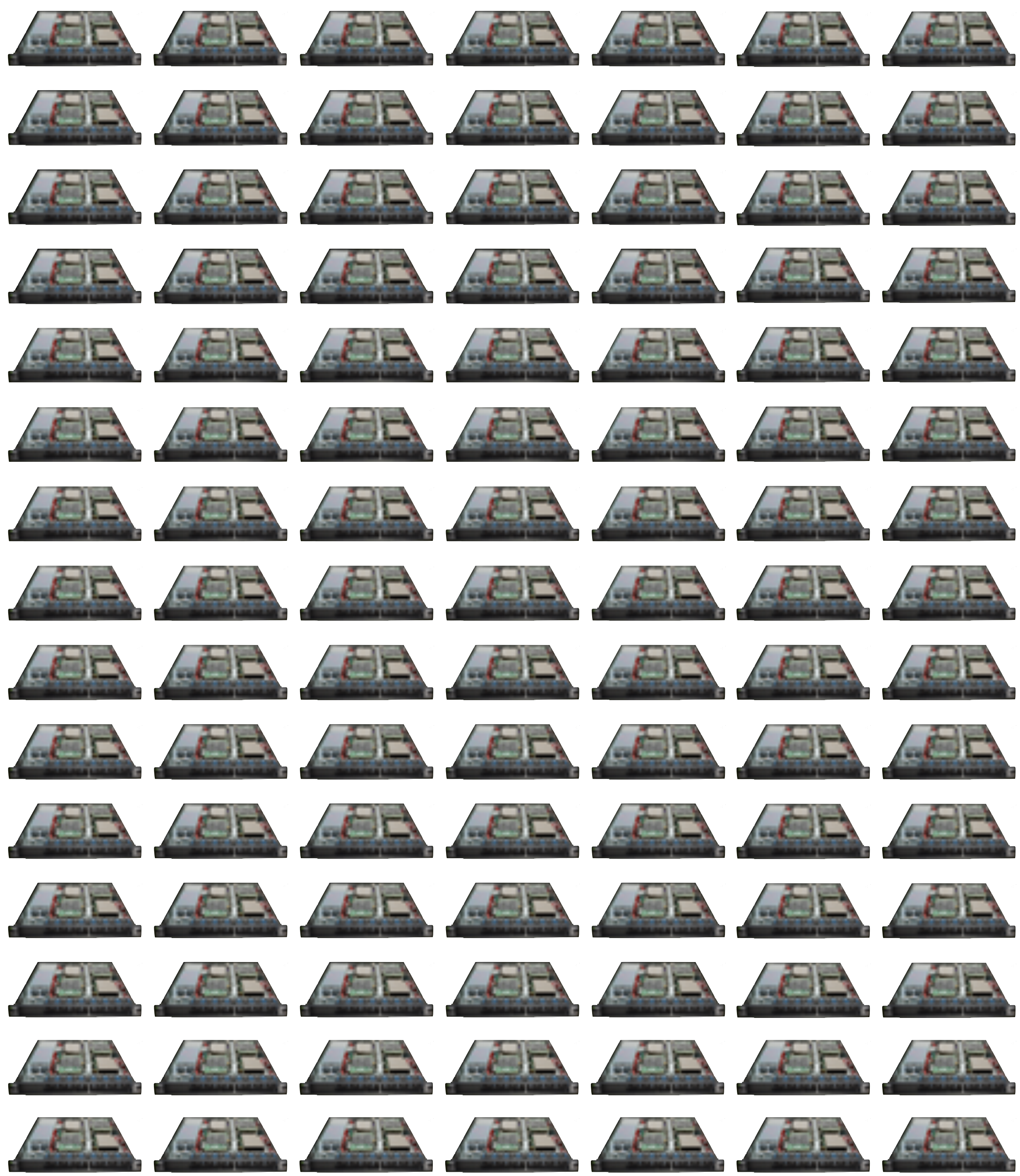


=

Performance Per Watt



Performance Per $



=

Conclusions

- Fantastic algorithmic performance obtained on today GPUs
  - Flops per Watt
  - Flops per $
- Some work required to get best performance
  - Standard libraries are not an option
  - Knowledge of the problem required
  - Reduce memory traffic at all costs
- Algorithm design critical component
  - Bandwidth constraints force complete rethink of problem
- Future work: scale to many GPUs

QCD on Fermi?

- Better double precision
  - Factor of 2 penalty vs. single precision
- More bandwidth
  - Current code will scale with bandwidth improvement
- More shared memory
  - Store spinors in shared memory to reduce memory traffic
  - Super-linear speedup over bandwidth
- Larger address space
  - Bigger problems on a single GPU,
- ECC memory
  - Deploy non-iterative code on GPU