Improving HISQ Propagator solves using deflation

Leon Hostetler[1], Kate Clark[2], Carleton DeTar[3], Steven Gottlieb[2], Evan Weinberg[2]

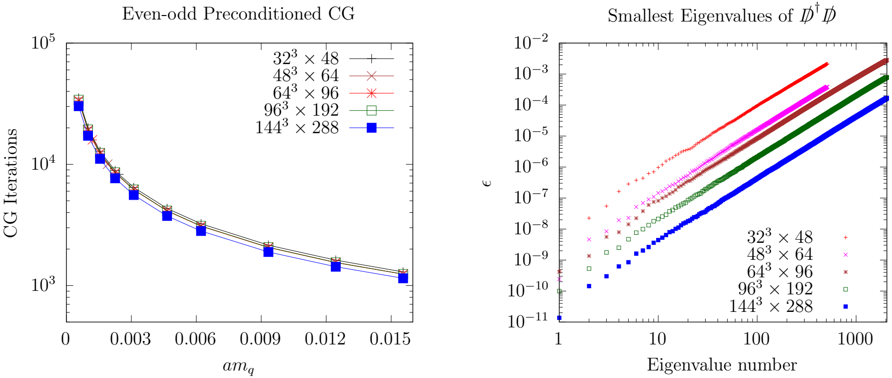
[1] Indiana University
[2] NVIDIA
[3] The University of Utah

August 1, 2024

Outline

Outline

Critical slowing down in propagator calculation



1. The CG iterations needed to compute propagators blows up as quark mass is decreased
2. CG iterations depends on the condition number of the Dirac matrix

$$\kappa = \frac{\lambda_{max}}{\lambda_{\min}}, \qquad \text{with } \lambda_{max} \approx 23 \text{ and } \lambda_{min} = \epsilon_{min} + 4m^2$$

Status of HISQ Multigrid

1. 2018: Multigrid for 2D Schwinger model by Brower, Weinberg, Clark, and Strelchenko (PRD 97, 114513)
2. Multigrid in 4D support added to QUDA
3. 2022: Multigrid-preconditioned GCR for HISQ by Ayyar, Brower, Clark, Wagner, and Weinberg (arXiv: 2212.12559)
   - Critical slowing down nearly eliminated
   - 10x speedup over CG for light quark propagators on $144^3 \times 288$
4. 2023: 4-level multigrid for HISQ by Ayyar and Brower (unpublished)
   - Critical slowing down significantly reduced
   - Lots of tuning needed
   - 4x speedup over CG for light quark propagators on $144^3 \times 288$
5. 2024: HISQ operator added to PETSc

Now, back to deflation!

In this talk...

- We experiment with deflation for Highly Improved **Staggered** Quarks (HISQ)
- Lattice configurations are from **MILC's "physical point" ensembles** with $a \approx 0.15, 0.12, 0.09, 0.06$, and 0.042 fm
- Eigenvectors (EVs) are generated using the `staggered_eigensolve_test` application from **QUDA** (`https://github.com/lattice/quda`)
- Propagators are computed using the `ks_spectrum` application from **MILC** (`https://github.com/milc-qcd/milc_qcd`)
  - Deflation and CG are offloaded to QUDA
- These tests were performed on **Frontier** (HPE Cray EX supercomputer) where each node has one 64-core AMD "Optimized 3rd Gen EPYC" CPU with 512 GB of memory and four AMD MI250X, each with 2 Graphics Compute Dies (GCDs) for a total of 8 GCDs per node
- Reported solve times do not include EV generation or loading times
- CG stopping criterion is a residual $< 10^{-8}$

Outline

HISQ Deflation

1. **Eigensolve:** Generate eigenvectors of

$$\not{D}^\dagger \not{D}$$

   using thick restarted Lanczos method (TRLM)

2. **Propagator solve:** Solve for ψ

$$M^\dagger M \psi = \eta, \qquad M \equiv \not{D} + 2m$$

   using deflated conjugate gradient for the normal equations (CGNE):

   1. Project eigenvectors $|v_i\rangle$ onto source vector

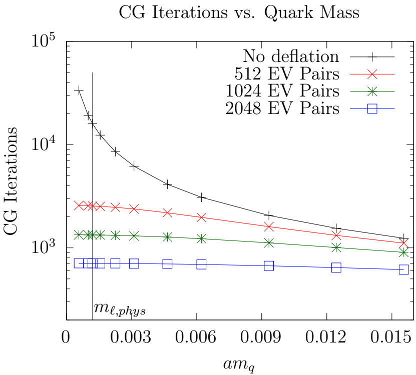
$$x = \sum_i |v_i\rangle \frac{1}{\lambda_i} \langle v_i|\eta\rangle$$

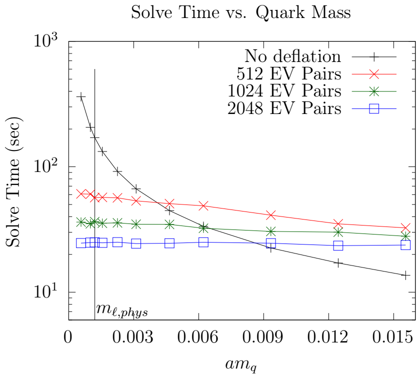
   2. Use x as initial guess to CG solver

HISQ Deflation

1. Deflation:
   - Eigenvectors are used to get an initial guess that has the correct low mode components
   - Then CG only has to deal with the high modes which converge more quickly
2. Critical slowing down is shifted from the CG solve to the eigensolve
3. What's the point then?
   - With undeflated CG, critical slowing down hits us on every solve
   - With deflated CG, critical slowing down hits us once per gauge configuration
   - Amortize the eigensolve cost over multiple propagator solves
4. As V is increased, deflation becomes relatively more costly
5. Eventually, we will need another solution...multigrid
6. But where?

$64^3 \times 96$ (0.09 fm) on 12 nodes



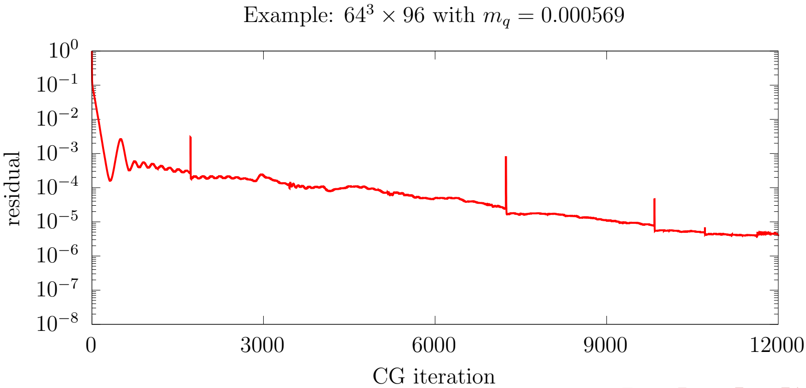


- **Left:** CG iterations versus quark mass. At $m_{\ell,phys}$, the ratio of undeflated vs. deflation with 2048 EVs is 22x
- **Right:** Time to compute two propagators versus quark mass. We see a 6.8x speedup, assuming setup costs can be amortized, at $m_{\ell,phys}$ with 2048 EVs

Outline

Deflation with Sloppy Eigenvectors

- With double precision eigenvectors, it is challenging to scale deflation to large volumes due to the size of the eigenvectors
  - Single parity storage
  - Single precision eigenvectors
  - Half precision for the inner CG solves
- Result: Deflated CG performs well at first but then stagnates:

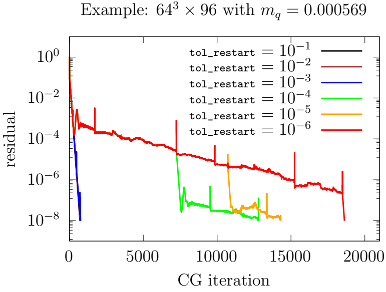


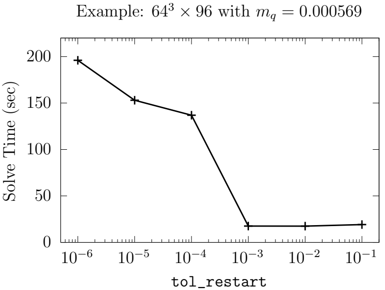
Multi-Deflation with Sloppy Eigenvectors

1. Solution: Restart the CG and re-apply the initial deflation when residual drops by some factor
2. Try different values for QUDA's `tol_restart` parameter
3. Find optimal value by looking at the solve time

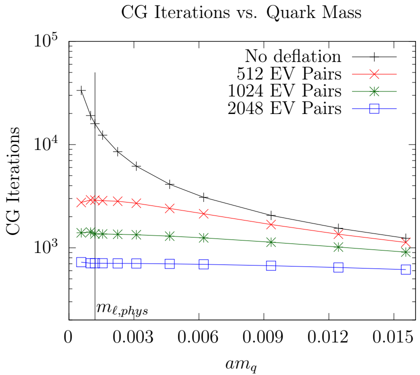
Result:

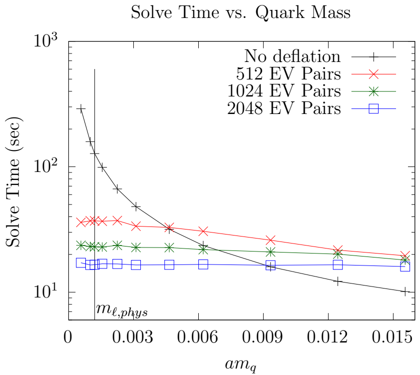
- CG convergence similar to single deflation with precise EVs
- Memory savings lead to 3x reduction in number of nodes needed in this example
- Further increases solve speedup





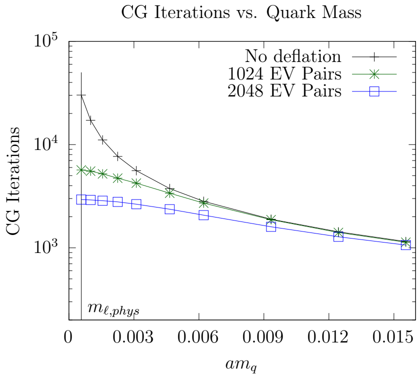
$64^3 \times 96$ (0.09 fm) on 4 nodes

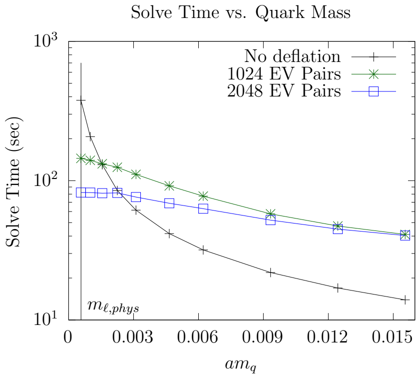




- **Left:** CG iterations versus quark mass. At $m_{\ell,phys}$, the ratio of undeflated vs. deflation with 2048 EVs is 22x
- **Right:** Time to compute two propagators versus quark mass. We see a 7.7x speedup, assuming setup costs can be amortized, at $m_{\ell,phys}$ with 2048 EVs

$144^3 \times 288$ (0.042 fm) on 192 nodes





- **Left:** CG iterations versus quark mass. At $m_{\ell,phys}$, the ratio of undeflated vs. deflation with 2048 EVs is 10x
- **Right:** Time to compute two propagators versus quark mass. We see a 4.6x speedup, assuming setup costs can be amortized, at $m_{\ell,phys}$ with 2048 EVs

Outline

Outlook for HISQ Deflation

1. Deflation is a viable solution to the critical slowing down problem for contemporary lattice sizes
    - Periodically restarting the CG and re-applying the deflation allows to use imprecise eigenvectors
    - Significant solve time speedups with room for further improvement
2. Further improvements for HISQ deflation are in progress:
    - Multiple right-hand side solves (Recall Tuesday talk by Kate Clark and poster by Evan Weinberg)
    - QUDA memory usage
    - Testing half-precision eigenvectors
    - Block TRLM to reduce eigensolve cost
    - Eigenvector compression
3. Head-to-head comparisons with MG-GCR and 4-level MG on $144^3 \times 288$ lattices

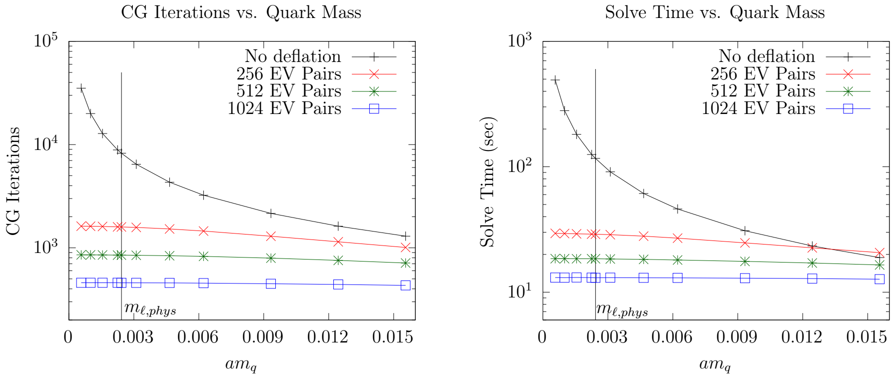
Thank you!

Additional Slides:

A note on results...

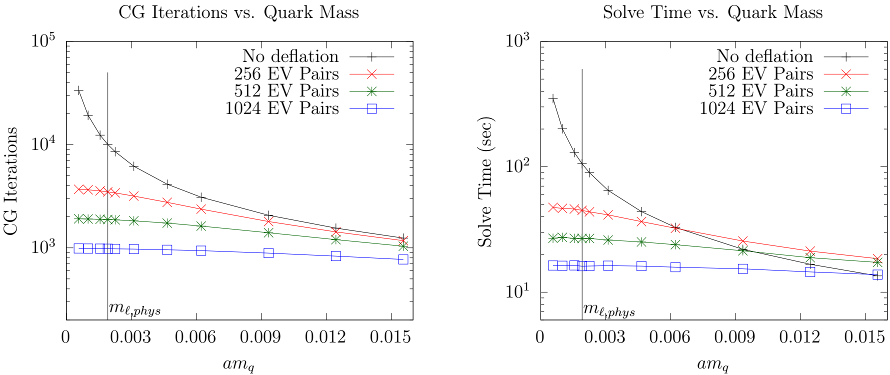
- "Solve times" reported here are actually for 2 propagators $\times$ 3 colors $= 6$ total solves
- Solve times reported here do not include EV generation or loading times
- To take advantage of of QUDA's autotuning as we would in production running, we do a pre-tuning run to save the tune cache, and report timing from a second run that reads the cached parameters.
- CG stopping criterion is a residual $< 10^{-8}$

$32^3 \times 48$ (0.15 fm) on 2 GCDs



- **Left:** CG iterations versus quark mass. At $m_{\ell,phys}$, the ratio of undeflated vs. deflation with 1024 EVs is 18x
- **Right:** Time to compute two propagators versus quark mass. We see a 9.0x speedup, assuming setup costs can be amortized, at $m_{\ell,phys}$ with 1024 EVs

$48^3 \times 64$ (0.12 fm) on 2 nodes



- **Left:** CG iterations versus quark mass. At $m_{\ell,phys}$, the ratio of undeflated vs. deflation with 1024 EVs is 10x
- **Right:** Time to compute two propagators versus quark mass. We see a 6.6x speedup, assuming setup costs can be amortized, at $m_{\ell,phys}$ with 1024 EVs

Challenges of Going to Larger Volumes

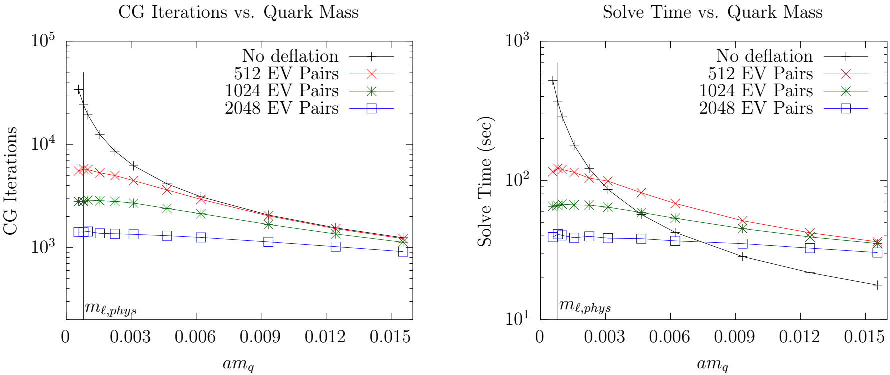
Double precision EVs take up a lot of space!

- Disk space: For example, 2.4TB for 2048 EVs of $64^3 \times 96$
- IO time: ~ 30 minutes to load these EVs from disk
- Memory: Requires 12 nodes whereas CG without deflation can run on 1 node!

Solutions:

- File size reduced by half when using EVs in *single parity* format and reduced by another half when saved in single precision
- IO improved by orders of magnitude when saving EVs in *partfile* format
- Memory usage halved by using single precision

$96^3 \times 192$ (0.06 fm) on 27 nodes



- **Left:** CG iterations versus quark mass. At $m_{\ell,phys}$, the ratio of undeflated vs. deflation with 2048 EVs is 17x
- **Right:** Time to compute two propagators versus quark mass. We see a 8.9x speedup, assuming setup costs can be amortized, at $m_{\ell,phys}$ with 2048 EVs

Setup Costs

- The focus of this study was purely on solve time and not on optimizing setup costs
- Here are the (unoptimized) setup costs that I saw:

Size	Nodes	Total File Size	Generating 2048 EVs[1]	Loading 2048 EVs[2]
$64^3 \times 96$	4	590GB	4014s	61s
$96^3 \times 192$	27	3.98TB	6668s	62s
$144^3 \times 288$	192	20.2TB	10470s	45s

- Expect a 2-3x reduction in EV generation time once we start using Block TRLM
- Knobs to tune include Chebyshev parameters (min, max, and polynomial degree) and the size of the “batched rotation” space

[1] Includes everything—loading gauge field, computing fat and long links, the eigensolve, and saving EVs to disk

[2] Assumes partfile (with $8\times$Nodes $=$ MPI ranks) and single-parity storage with EVs in single precision

HISQ Deflation I

The procedure used by `ks_spectrum` to compute the propagators:

$$M^\dagger M \psi = \eta,$$

is as follows:

1. MILC: Preconditions even and odd sites
$$y = M^\dagger \eta$$
2. MILC: Prepares even site source y_e and passes it off to QUDA
3. QUDA: Loads the eigenvectors (previously done on CPU)
4. QUDA: Performs the deflation (previously done on CPU)
5. QUDA: Performs the CG solve
$$\psi_e = \left(M^\dagger M\right)^{-1} y_e$$

HISQ Deflation II

6. MILC: Receives the even site solution ψ_e from QUDA
7. MILC: Reconstructs the odd site solution ψ_o

$$\psi_o = \frac{1}{2} m \left(D_{oe} \psi_e + \eta_o \right)$$

8. QUDA: Polishes the odd site solution using one or more CG iterations

$$\psi_o = \left(M^\dagger M \right)^{-1} y_o$$

9. Repeat all of the above for the other two colors

HISQ Deflation III

Note:

- Odd part v_o of an eigenvector can be reconstructed from the even part v_e

$$v_o = \frac{i}{\lambda} D_{oe} v_e$$

  See, e.g. arXiv:1710.07219
- Single parity format: Storage need (disk and memory) is reduced by half when we use only even part v_e
- Since odd site solution ψ_o is explicitly reconstructed from even site solution, there is no need for deflation for the odd sites
- Thus no need for us to ever compute or store the odd part v_o of the eigenvectors.

Speedups at $m_{\ell,phys}$ with 2048 EVs

Real-time speedups given *identical* resources:

Size	$m_{\ell,phys}$	Without Deflation Nodes	Without Deflation Solve Time	With Deflation Nodes	With Deflation Solve Time	Speedup
$64^3 \times 96$	0.0012	4	127.3s	4	16.55s	7.7x
$96^3 \times 192$	0.0008	27	366.6s	27	41.12s	8.9x
$144^3 \times 288$	0.000569	192	377.5s	192	82.23s	4.6x

Cost comparisons using *minimal* resources:

Size	Without Deflation Nodes	Without Deflation Solve (s)	Without Deflation Cost (N-s)	With Deflation Nodes	With Deflation Solve (s)	With Deflation Cost (N-s)	Efficiency
$64^3 \times 96$	1	316.2	316.2	4	16.55	66.20	4.8x
$96^3 \times 192$	3	1182	3546	27	41.12	1110	3.2x
$144^3 \times 288$	12	1777	21,320	192	82.23	15,790	1.4x