ALGORITHMS FOR LATTICE QCD

Kostas Orginos
William and Mary / JLab

INT EXASCALE WORKSHOP, June 27-July 1
OUTLINE

• Lattice formulation of QCD

• Computation
  • Configuration generation
    • Hybrid Monte Carlo
  • Reweighting
  • Correlation functions
    • Linear Solvers

• Outlook
LATTICE QCD

In continuous Euclidian space:

\[ Z = \int DqD\bar{q}DA_\mu \ e^{-S[\bar{q},q,A_\mu]} \]

\[ \langle \mathcal{O} \rangle = \frac{1}{Z} \int DqD\bar{q}DA_\mu \ \mathcal{O}(\bar{q},q,A_\mu) \ e^{-S[\bar{q},q,A_\mu]} \]

Lattice regulator:

Gauge sector:

\[ U_\mu(x) = e^{-i\alpha A_\mu(x + \frac{\hat{\mu}}{2})} \]

Fermion sector:

Things get nasty!
In continuous Euclidian space:

\[ Z = \int Dq D\bar{q} D A_\mu \ e^{-S[\bar{q}, q, A_\mu]} \]

\[ \langle \mathcal{O} \rangle = \frac{1}{Z} \int Dq D\bar{q} D A_\mu \ \mathcal{O}(\bar{q}, q, A_\mu) \ e^{-S[\bar{q}, q, A_\mu]} \]

Lattice regulator:

Gauge sector:

\[ U_\mu(x) = e^{-i\alpha A_\mu (x + \frac{\mu}{2})} \]

Fermion sector:

Things get nasty!

**Fermion doubling**
In continuous Euclidian space:

\[
Z = \int DqD\bar{q}DA_\mu e^{-S[\bar{q},q,A_\mu]}
\]

\[
\langle \mathcal{O} \rangle = \frac{1}{Z} \int DqD\bar{q}DA_\mu \mathcal{O}(\bar{q},q,A_\mu) e^{-S[\bar{q},q,A_\mu]}
\]

Lattice regulator:

Gauge sector:

\[
U_\mu(x) = e^{-iaA_\mu(x+\frac{\mu}{2})}
\]

Fermion sector:

Things get nasty!

**Fermion doubling**

**Chiral symmetry breaking**
Gauge sector:

\[ S_g(U) = \beta \sum_p \left( 1 - \frac{1}{3} \text{ReTr} U_p \right) \rightarrow \frac{1}{4} F_{\mu\nu}^2 \]

Fermion sector:

\[ S_f(\bar{q}, q, U) = \bar{q} D(U) q \]

- \( D(U) \) sparse matrix
- Wilson fermions
- Kogut-Susskind fermions
- Domain Wall
- Overlap: Not a sparse matrix

\[ \mathcal{Z} = \int \mathcal{D}[U] \mathcal{D}[\bar{\psi}] \mathcal{D}[\psi] \ e^{-\bar{\psi} D(U) \psi - S_g(U)} \]
Gauge sector:

\[ S_g(U) = \beta \sum_p \left( 1 - \frac{1}{3} \text{ReTr} U_p \right) \rightarrow \frac{1}{4} F_{\mu\nu}^2 \]

Fermion sector:

\[ S_f(\bar{q}, q, U) = \bar{q} D(U) q \]

- $D(U)$ sparse matrix
- Wilson fermions
- Kogut-Susskind fermions
- Domain Wall
- Overlap: Not a sparse matrix

\[ \mathcal{Z} = \int D[U] \det(D(U)) e^{-S_g(U)} \]
\[
\langle \mathcal{O} \rangle = \frac{1}{Z} \int \prod_{\mu, x} dU_\mu(x) \mathcal{O}[U, D(U)^{-1}] \det (D(U)^\dagger D(U))^{n_f/2} e^{-S_g(U)}
\]
\[ \langle \mathcal{O} \rangle = \frac{1}{Z} \int \prod_{\mu, x} dU_\mu(x) \quad \mathcal{O}[U, D(U)^{-1}] \quad \det (D(U)\dagger D(U))^{n_f/2} \quad e^{-S_g(U)} \]
\[
\langle \mathcal{O} \rangle = \frac{1}{Z} \int \prod_{\mu,x} dU_\mu(x) \left[ \mathcal{O}[U, D(U)^{-1}] \right] \left( \text{det} (D(U)^\dagger D(U)) \right)^{n_f/2} e^{-S_g(U)}
\]

Matrix Inversion: Iterative Solvers

\[ D(U) \chi = \psi \]

Monte Carlo integration
Hybrid Monte Carlo: No determinant evaluation
\[ \langle \mathcal{O} \rangle = \frac{1}{Z} \int \prod_{\mu, x} dU_\mu(x) \left( \mathcal{O}[U, D(U)^{-1}] \right) \det (D(U)^\dagger D(U))^{n_f/2} e^{-S_g(U)} \]

- Solution of linear system: significant CPU time
- HMC needs matrix inversions
  - Continuously changing U
  - Fixed right hand side \( \psi \)
- Correlation functions:
  - Fixed U
  - Large number of orthogonal right hand sides \( \psi \)
WHAT DOES IT TAKE?

- Hadronic scale
  - characteristic length $\sim \Lambda_{QCD} \sim 220\text{MeV}$
  - $1\text{fm} = 1 \times 10^{-13}\text{cm}$

- The lattice spacing
  - $a \ll 1\text{fm}$

- The lattice size
  - $La \gg 1\text{fm}$

- Reasonable choices
  - $a = .1\text{fm}$, ... $La = 3\text{fm}$

- Degrees of freedom
  - $2 \times 3 \times 4 \times 32^4 = 2.5 \times 10^7$
    - Algorithm scaling: $\sim \frac{1}{a^7}$
HYBRID MONTE CARLO


\[ \langle \mathcal{O} \rangle = \frac{1}{Z} \int \prod_{\mu,x} dU_\mu(x) \, \mathcal{O}[U, D(U)^{-1}] \, \det(D(U)^\dagger D(U))^{n_f/2} \, e^{-S_g(U)} \]

Generate gauge fields with probability:

\[ P(U) = \det(D(U))^{n_f} e^{-S_g(U)} \]

Then expectation values become averages:

\[ \langle \mathcal{O} \rangle = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}(U_i, \frac{1}{D(U_i)}) \]

Need an update of U that:

- Detailed Balance
- Ergodicity
- Avoids the computation of the determinant
The two flavor case:

\[ D(U)\dagger = \gamma_5 D(U)\gamma_5 \quad \text{det}(D(U))^2 = \text{det}(D(U)\dagger D(U)) \]

Using bosonic fields:

\[ \text{det}(D(U)\dagger D(U)) = \int d\phi^\dagger d\phi e^{-\phi^\dagger \frac{1}{D(U)\dagger D(U)} \phi} \]

Add conjugate momenta to the gauge fields with gaussian action:

\[ P_\mu(x) \leftrightarrow U_\mu(x) \quad S_p = \frac{1}{2} \sum_{\mu,x} P_\mu(x)^2 \]

Hamiltonian evolution:

\[ \{P, U\} \leftrightarrow \{P', U'\} \]

\[ \mathcal{H} = \frac{1}{2} \sum_{\mu,x} P_\mu(x)^2 + S_g(U) + \phi^\dagger \frac{1}{D(U)^\dagger D(U)} \phi \]
In continuous fictitious evolution time:

\[ \dot{U} = \frac{\partial H}{\partial P} \quad \dot{P} = -\frac{\partial H}{\partial U} \]

The algorithm satisfies:
- Detailed balance
- Ergodicity

Need numerical reversible integration algorithm

Leapfrog Integrator
Omelyan Integrator

\[ \{P, U\} \rightarrow \{P', U'\} \]

Metropolis accept reject to correct energy violations

Use multiple time steps. Isolate sources of large force and evolve them at smaller time steps.

Use multiple time steps. Isolate sources of large force and evolve them at smaller time steps.

Example: Gauge action generates larger force than the fermion action.

MULTIPLE TIME STEPS


- Split up the Hamiltonian

\[ \mathcal{H} = \frac{1}{2} \sum_{x, \mu} P_\mu(x)^2 + S_1 + S_2 \]

\[ T_U(\epsilon) : \quad U \rightarrow e^{i\epsilon P} U \]

\[ T_P^1(\epsilon) : \quad P \rightarrow P + \epsilon F_1 \]
\[ T_P^2(\epsilon) : \quad P \rightarrow P + \epsilon F_2 \]

- Two evolutions

\[ T_1 = T_P^1(\epsilon_1/2) T_U(\epsilon_1) T_P^1(\epsilon_1/2) \]
\[ T_2 = T_P^1(\epsilon_2/2) T_1^{N_1}(\epsilon_1) T_P^1(\epsilon_2/2) \]

- Full trajectory \( \tau \):

\[ [T_2]^{N_2} \]

- Time steps fulfill: \( N_2 = \tau / \epsilon_2 \quad N_1 = \epsilon_1 / \epsilon_2 \)
MULTIPLE TIME STEPS


- Split up the Hamiltonian

\[ \mathcal{H} = \frac{1}{2} \sum_{x,\mu} P_{\mu}(x)^2 + S_1 + S_2 \]

\[ T_U(\epsilon) : \quad U \rightarrow e^{i\epsilon P} U \]

\[ T^1_P(\epsilon) : \quad P \rightarrow P + \epsilon F_1 \]

\[ T^2_P(\epsilon) : \quad P \rightarrow P + \epsilon F_2 \]

- Two evolutions

\[ T_1 = T^1_P(\epsilon_1/2)T_U(\epsilon_1)T^1_P(\epsilon_1/2) \]

\[ T_2 = T^1_P(\epsilon_2/2)T^N_1(\epsilon_1)T^1_P(\epsilon_2/2) \]

- Full trajectory \( \tau \):

\[ [T_2]^{N_2} \]

- Time steps fulfill: \( N_2 = \tau / \epsilon_2 \quad N_1 = \epsilon_1 / \epsilon_2 \)
MULTIPLE TIME STEPS


Split up the Hamiltonian

\[ \mathcal{H} = \frac{1}{2} \sum_{x, \mu} P_\mu(x)^2 + S_1 + S_2 \]

\[ T_U(\epsilon) : \quad U \rightarrow e^{i\epsilon P} U \]

\[ T_P^1(\epsilon) : \quad P \rightarrow P + \epsilon F_1 \]

\[ T_P^2(\epsilon) : \quad P \rightarrow P + \epsilon F_2 \]

Two evolutions

\[ T_1 = T_P^1(\epsilon_1/2) T_U(\epsilon_1) T_P^1(\epsilon_1/2) \]

\[ T_2 = T_P^1(\epsilon_2/2) T_1^{N_1}(\epsilon_1) T_P^1(\epsilon_2/2) \]

Full trajectory \( \tau \):

\[ [T_2]^{N_2} \]

Time steps fulfill:

\[ N_2 = \frac{\tau}{\epsilon_2} \quad N_1 = \frac{\epsilon_1}{\epsilon_2} \]
- This allows fast evolution
- Small energy violation
- Large acceptance
- This allows fast evolution
- Small energy violation
- Large acceptance
• This allows fast evolution
• Small energy violation
• Large acceptance
- This allows fast evolution
- Small energy violation
- Large acceptance
THE FERMION FORCE

Most challenging

Need to solve:
\[ \chi = \frac{1}{D^+(U)D(U)}\phi \]

- Harder as the quark mass gets smaller
- Fermion force dominates at small quark masses
PRECONDITIONED HMC

Idea: Split up the fermion force

\[ \det \left( D(U)^\dagger D(U) \right) = \det \left( M(U)^\dagger M(U) \right) \det \left( \frac{1}{M(U)} D(U)^\dagger D(U) \frac{1}{M(U)} \right) \]

- Use two boson fields (pseudo-fermions)

- \( M(U) \) Preconditioner that generates cheap but large force

- The correction term the gives small force

- Preconditioner need not be good solver preconditioner
PRECONDITIONED HMC

- UV spectrum of $D(U)$: Large force
- IR spectrum of $D(U)$: Small force
HEAVY MASS PRECONDITIONING

\[ \chi = M^\dagger(U)[m_h] \frac{1}{D^\dagger(U)[m_l]D(U)[m_l]} M(U)[m_h] \phi \]

Cheap Large force: small time step

\[ \chi' = \frac{1}{M^\dagger(U)[m_h]M(U)[m_h]} \phi' \]

Expensive small force: Large time step

\[ \det(D(U)\dagger D(U)) = \det(M(U)\dagger M(U)) \det\left( \frac{1}{M(U)\dagger} D(U)\dagger D(U) \frac{1}{M(U)} \right) \]
Use Chebyshev polynomial approximation to the UV spectrum

\[ M(U)^{-1} = P(\lambda, D^\dagger D) \quad \text{Approximation good in } [\lambda, 1] \quad \lambda \sim 0.3 \]

Polynomium degree is small (n \sim 16)
Most of the Fermion force comes from this limited part of the spectrum
Most eigenvalues are in this range!
Force calculation is cheap (No matrix inversion needed)

Golub Ruiz Touhmi 2005: Use this preconditioner for multiple right hand sides
Schwarz-Preconditioner

DDHMC


- M(U): No links to neighboring Blocks
- M(U): UV physics
- Correction term needs noise on the surface only
- Correction term: IR physics
- Factor of $\sim 10$ speed up at small quark masses

Tuesday, June 28, 2011
DEFLATED-DD-HMC

[Uses an GCR with an AMG preconditioner]

[Chronology reduces the refresh of the preconditioner]

[Nearly removes critical slowing down]

[Lattice $32^3 \times 64$ $a = 0.08$fm]

> plot by M. Luscher 1002.4232v2

Tuesday, June 28, 2011
RATIONAL HMC

[Clark and Kennedy]

\[
\det(D(U)\,^\dagger\, D(U)) = \det(R(D(U)\,^\dagger\, D(U))^{-2})
\]

\[
R(x) = \frac{P_n(x)}{Q_m(x)} = \sum_{k=1}^{m} \frac{a_k}{x + b_k} \rightarrow \frac{1}{x^{1/2}}
\]

- Very accurate
- Generated using Remez algorithm
- Multi shift solver (Real roots \(b_k > 0\))
- Can be used to work with odd number of flavors (strange quark)
RHMC AND N\textsuperscript{TH} ROOT TRICK

\[
\det(D(U)^\dagger D(U)) = \det([D(U)^\dagger D(U)]^{1/n})^n
\]

\begin{itemize}
  \item Use different pseudo fermions for each factor
  \item Force 1/n the original
  \item Evolution can go faster
\end{itemize}
RHMC MULTI TIME SCALE


- Use different time scale small and large roots.
- Force 1/n the original
WORLD HMC PERFORMANCE

TFlop-years vs. $m_\pi$

10000 MD units

- DWF(0.11)
- DWF(0.08)
- AuxDet(0.14)
- Asqtad(0.09)
- Asqtad(0.06)
- HISQ(0.09)
- Ani. Clover(0.04/0.125)
- Clover(0.06)
- Clover(0.09)
- tmWilson(0.08)
- CI(0.15)

- 2 flavor Clover Defl. - DDHMC arXiv:1002.4232v2

figure by C. Jung arXiv:001.0941v1

Tuesday, June 28, 2011
ISOSPIN BREAKING

• Gauge field configurations are generated with the degenerate up and down quark masses

• In nature $m_{\text{up}} \sim 2$ MeV and $m_{\text{down}} \sim 5$ MeV

• Precision calculations will need to non-degenerate light quark masses

• Nuclear physics: Fine tuned

• We need an efficient way to do calculations to slightly vary parameters in the action
Reweighting is a method used to perform calculations using an ensemble that does not have the action parameters we want.

- Gauge configurations are generated with $m_{up} = m_{down}$.
- Observables with $m_{up} \neq m_{down}$ can be calculated.
Starting ensemble

\[ \langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \mathcal{D}[U] \det(D^\dagger(U)D(U))^{N_f/2} \mathcal{O}(D(U)^{-1}, U) \ e^{-S_g(U)} \]

\[ \mathcal{Z} = \int \mathcal{D}[U] \det(D(U)D^\dagger(U))^{N_f/2} \ e^{-S_g(U)} \]

Target ensemble

\[ \langle \mathcal{O}' \rangle = \frac{1}{\mathcal{Z}'} \int \mathcal{D}[U] \det(D'^\dagger(U)D'(U))^{N_f/2} \mathcal{O}(D'(U)^{-1}, U) \ e^{-S_g(U)} \]

\[ \mathcal{Z}' = \int \mathcal{D}[U] \det(D'(U)D'^\dagger(U))^{N_f/2} \ e^{-S_g(U)} \]

Modify the fermion action
\[
\langle \mathcal{O} \rangle' = \frac{1}{Z'} \int \mathcal{D}[U] \ e^{\frac{N_f}{2} \left[ \text{Tr} \log(D'^\dagger(U)D'(U)) - \text{Tr} \log(D^\dagger(U)D(U)) \right]} \ \mathcal{O}(D'(U)^{-1}, U) \ e^{-S(U)}
\]

\[
Z' = \frac{1}{Z} \int \mathcal{D}[U] \ e^{\frac{N_f}{2} \left[ \text{Tr} \log(D'^\dagger(U)D'(U)) - \text{Tr} \log(D^\dagger(U)D(U)) \right]} \ e^{-S(U)}
\]

Computational task: Evaluate the trace log of a sparse positive definite matrix

- Use pseudofermions just like HMC
- Compute inverses of the Dirac Matrix
- Use Gaussian quadrature
- Lanczos iteration
- Converges faster than solving a linear system (with ex. CG)

Hassenfratz et. al. arXiv:0805.2369 ;
RBC arXiv:1011.0892 ;
PACS-CS arXiv:0911.2561

[Golub & Meurant ’93; Bai, Fahey & Golub ’96]
GAUSSIAN QUADRATURE

[Golub & Meurant ’93; Bai, Fahey & Golub ’96]

\[ Tr \log(A) \approx \frac{1}{N} \sum_{k=1}^{N} \eta_k^\dagger \log(A) \eta_k \]

\( \eta \) are vectors whose components are random \( Z_4 \) noise

Gaussian quadrature evaluates \( \eta_k^\dagger \log(A) \eta_k \)

\[ \eta^\dagger f(A) \eta = \eta^\dagger Q^\dagger f(\Lambda)Q \eta = u^\dagger f(\Lambda)u = \sum_i u_i^* f(\lambda_i)u_i \]

With \( Q \) the eigenvector matrix and \( \lambda_i \) the eigenvalues of \( A \)
\[ I[f] = \eta^\dagger f(A)\eta = \sum_i u_i^* f(\lambda_i)u_i = \int_a^b d\lambda \sum_{i=1}^n u_i^* u_i \delta(\lambda - \lambda_i) f(\lambda) = \int_a^b d\mu(\lambda) f(\lambda) \]

\[ \mu(\lambda) = \begin{cases} 
0, & \text{if } \lambda < a = \lambda_1 \\
\sum_j^i u_j^* u_j, & \text{if } \lambda_i \leq \lambda < \lambda_{i+1} \\
\sum_j^n u_j^* u_j, & \text{if } b = \lambda_n \leq \lambda 
\end{cases} \]

To calculate the integral use Gaussian Quadrature integration with the orthogonal polynomial defined by the Lanczos recursion relation

\[ I[f] \approx \sum_{i=1}^k \omega_i^2 f(\theta_i) \]

\( \theta_i \) are the eigenvalues and \( \omega_i \) the squares of the first elements of the normalized eigenvectors of the Lanczos matrix \( T_k \)

We apply this method to reweighting: [A. Rehim W. Detmold KO]
average thin plaquette, $m_{\text{sea}} = -0.170 \longrightarrow -0.175$

Thin plaquette type

Average

plot by A. Rehim

Tuesday, June 28, 2011
average thin plaquette, \( m_{\text{sea}} = -0.170 \rightarrow -0.180 \)

\[ m_{\text{sea}} = -0.170 \]

\[ m_{\text{sea}} = -0.180 \]

\[ \text{reweighted } m_{\text{sea}} = -0.170 \]

plot by A. Rehim
REWEIGHTING

- Is already very useful and may become even more so in the near future
- Linear algebra methods for determining the reweighting factor work well
- Reweighted observables agree with exact results provided that the shifts in the action parameters are small
- Can we find further improvements? Do multi-grid like approaches exist?
CORRELATION FUNCTIONS

\[ H_n = 1 (\xi, t) = A_{10} (t) \]

\[ H_n = 2 (\xi, t) = A_{20} (t) - (2 \xi)^2 C_{20} (t) \]

\[ P, S | O | P, S \]

\[ C_{2pt}(\vec{p}, t) = \langle J_{\vec{p}}(t) J(0) \rangle \]

\[ C_{3pt}(\vec{p}, \vec{q}; t, \tau) = \langle J_{\vec{p}}(t) O(\vec{q}, \tau) J(0) \rangle \]

Spectrum

Structure

\[ C_{mp} = \langle J_1(t) J_2(t) J_1(0) J_2(0) \rangle \]

Interactions and multi-particle Spectrum
DEFLATION

- Iterative solvers slow down when the matrix has very low eigenvalues
- Basic Idea: Project out the low modes
- Computing eigenvectors is expensive
  - 1 eigenvector roughly costs as much as solving one linear system
- Deflation can work well if the cost of constructing a sufficient basis spanning the low eigenvector space is small
- Two approaches:
  - Krylov methods: Computing the basis while solving a linear system
    - EigCG: symmetric problem [Stathopoulos and KO arXiv:0707.0131]
The eigCG algorithm

Developed with A. Stathopoulos (W&M)

Basic goals:

• Let CG do its job in solving the system

• Slowly accumulated few low eigenvectors of our matrix $A$ by interrupting CG without restarting it

• Use are “recurrence” relation that improves eigenvector convergence

• Use limited memory i.e. do not store all residual vectors that CG produces
The eigCG algorithm ($\text{Nev}, m$)

$k = 0; j = 0; x_0 = 0; r_0 = b$

while $r_k \neq 0$

$k = k + 1$

if ($k = 1$)

$p_1 = r_0$

else

$\beta_k = \frac{r_{k-1}^\dagger r_{k-1}}{r_{k-2}^\dagger r_{k-2}}$

$p_k = r_{k-1} + \beta_k p_{k-1}$

end

$\alpha_k = \frac{r_{k-1}^\dagger r_{k-1}}{p_k^\dagger A p_k}$

$x_k = x_{k-1} + \alpha_k p_k$

$r_k = r_{k-1} - \alpha_k A p_k$

end

$x = x_k$

$v_j = \frac{r_k}{\|r_k\|}$

update the $T_j$ matrix

if ($j = m$)

diagonalize $T_m \rightarrow Y_m$ keep lowest $\text{Nev}$

diagonalize $T_{m-1} \rightarrow Y_{m-1}$ keep lowest $\text{Nev}$

$QR$ factorize $Y_m, Y_{m-1} \rightarrow Q$

construct $H = Q^\dagger T_m Q$

$H$ is $2N_{\text{ev}} \times 2N_{\text{ev}}$

diagonalize $H \rightarrow Z$

$v_i = \sum_{n=1}^{m} (QZ)_{ni} v_n \ {i = 1..2N_{\text{ev}}}$

$j = 2N_{\text{ev}}$

rebuild $T_j$

end

Tuesday, June 28, 2011
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]

Tuesday, June 28, 2011
The eigCG algorithm

• Iterate the CG algorithm
• Save the residual vectors and fill in the tridiagonal matrix
• When max number of vectors reached: Diagonalize
• Keep only few low eigenpairs
• Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

• Iterate the CG algorithm

• Save the residual vectors and fill in the tridiagonal matrix

• When max number of vectors reached: Diagonalize

• Keep only few low eigenpairs

• Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m = 9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

• Iterate the CG algorithm
• Save the residual vectors and fill in the tridiagonal matrix
• When max number of vectors reached: Diagonalize
• Keep only few low eigenpairs
• Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2$  $m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

\[ N_{ev} = 2 \quad m=9 \]
The Incremental eigCG

For the first \( s_1 \) right hand sides do:

\[
U = [], H = [] \quad \% \text{accumulated Ritz vectors}
\]

for \( i = 1 : s_1 \)

\[
x_0 = U H^{-1} U^H b_i \quad \% \text{for } s_1 \text{ initial rhs}
\]

\[
[x, V, M] = \text{eigCG}(\text{nev}, m, A, x_0, b_i) \quad \% \text{eigCG with initial guess } x_0
\]

\[
\tilde{V} = \text{orthonormalize } V \text{ against } U \quad \% \text{not strictly necessary}
\]

\[
W = A \tilde{V}, H = \begin{bmatrix} H & U^H W \\ W^H U & \tilde{V}^H W \end{bmatrix} \quad \% \text{Add } \text{nev} \text{ rows to } H
\]

Set \( U = [U, \tilde{V}] \quad \% \text{Augment } U
\]

end

• Most of the cost is in Rayleigh-Ritz

• Needs \( \text{nev} \) MatVec operations and several dot products

• Ultimate size of \( U \) is determined by how much cost you can amortize for a given number of right hand sides

• Lots of the flops can be done efficiently using level 3 BLAS
EigCG for QCD

$N_{ev}=10, m=100$

- Valence Quark mass equals to sea quark mass

- Convergence of 48 successive linear systems
  Incremental eigCG on the first 24, then Init–CG with restarting at 2.5e–5

Case: $16^3 \times 64$. Mass = $-0.4125$

Case: $24^3 \times 64$. Mass = $-0.4125$

- Number of iterations (matrix-vector operations)
- Linear system residual norm

Tuesday, June 28, 2011
Critical slowing down (or lack off)

**EigCG** [Stathopoulos and KO arXiv:0707.0131]

### Case: $16 \times 16 \times 16 \times 64 \times 12$

![Graph showing critical quark mass and Matvecs for Case 16]

### Case: $24 \times 24 \times 24 \times 64 \times 12$

![Graph showing critical quark mass and Matvecs for Case 24]

For sufficiently large number of right hand sides:

- Small volume factor of 8 speed up at $m_{\text{sea}} = m_{\text{val}}$
- Large volume factor of 6 speed up at $m_{\text{sea}} = m_{\text{val}}$

Numerical tests ran at NERSC (franklin) and the cyclades cluster at W&M

Tuesday, June 28, 2011
EIGENVECTOR ACCURACY

\[ R = \| A e - \lambda e \| \]

- For light masses we get O(50) eigenvectors to single precision accuracy
Large Lattice Tests

$32^2 \times 96 \, \text{mq} = -0.4125 \, 256 \, \text{vecs}$
Large Lattice Tests

$32^3 \times 96 \text{mq} = -0.4125 \ 512 \ \text{vecs}$

![Graph showing residual vs. iteration for large lattice tests with various line styles and colors. The y-axis represents the residual on a logarithmic scale ranging from $10^{-8}$ to $10^2$, and the x-axis represents the iteration number from 0 to 2500.](#)
Large Lattice Tests

$32^3 \times 96 \text{mq} = -0.4125 \text{ 1024 vecs}$

![Graph showing residual vs. iter for large lattice tests]
Large Lattice Tests

Lattice: $32^3 96 \ m_q = -0.4125$

- Deflation space size 512
- Deflation space size 1024
- Deflation space size 256

Large Lattice Tests
Large Lattice Tests

Lattice: $32^3 96 \ m_q = -0.4125$

- Deflation space size 256
- Deflation space size 1024
- Deflation space size 512

integrated time (sec)

No rhs
EigBiCG

Developed with A. Rehim  A. Stathopoulos (W&M)

• Same idea as EigCG applied to BiCG

• Exploits the bi-Lanczos algorithm to built a deflation subspace while solving linear systems

• Use BiCGstab for steady state inverter

• BiCGstab has good performance for certain LQCD problems
EigBiCG on a $12^4$ lattice ($N=2.5\times10^5$)

plot by A. Rehim
OTHER DEFLATION ALGORITHMS


- Luscher’s Schwarz pre-conditioner (Domain-Decomposition) see talk by Balint Joo

- **Multigrid** see talks by Brower, Falgout, and Cohen
CONCLUSIONS

• Numerical linear algebra algorithms play an important role in Lattice QCD calculations

• Improvements by orders of magnitude have been made by careful tuning and innovative ideas

• Still a lot needs to be done

• In some cases reformulation of the problem might become more important than the implementation details of existing methodologies