Extending the QUDA library with the eigCG solver

Alexei Strelchenko

Scientific Computing Division @ Fermilab

Lattice 2014, Columbia University, June 23
Outline

- The Incremental eigCG
- Implementation details
- Numerical experiments
- Conclusion
The eigCG(k, m) algorithm


0 \hspace{1cm} i = 0; V^m = []; \hspace{1cm} r_0 = b - A x_0; \hspace{1cm} //\text{search vectors and residual}

1 \hspace{1cm} \text{for } j = 0, 1, \ldots \text{ until } ||r_j|| / ||r_0|| < tol:

2 \hspace{1cm} \text{Inside standard CG iteration:}

3 \hspace{1cm} \text{update the Lancz. matrix } T_m \text{ and the Lancz. vector } V^m_i

4 \hspace{1cm} \text{if } i == m: \hspace{1cm} \text{restart } V^{(2k)}, \text{ set } i = 2 \times nev

5 \hspace{1cm} i = i + 1

6 \hspace{1cm} \text{Update residual and solution}

7 \hspace{1cm} \text{end for}
Eigenvector computing in eigCG

Given $V^m$ and $T_m = V^{m+}AV^m$ and $T_{m-1} = V^{m-1+}AV^{m-1}$:

1. Solve for lowest $k$ eigenpairs: $T_m u_i = \lambda_i u_i$, $T_{m-1} \bar{u}_i = \bar{\lambda}_i \bar{u}_i$
2. Orthogonalize $\bar{U}$ against $U$: $Q = orth[U, \bar{U}]$
3. Set $H = Q^+T_mQ$ and solve: $H z_j = \mu_j z_j$
4. Compute $2k$ Ritz vectors: $y_j = V^m Q z_j$
5. Restart $V^{2k}$: $V^{2k} = [y_1, \ldots, y_{2k}]$
6. Rebuild $T_m$
Improving eigenvec. accuracy: the Incremental eigCG


1. \( U = [], \quad H = [] \)  // accum. Ritz vectors
2. \( \textbf{for} \quad s = 1, \ldots, s_1 : \)  // for \( s_1 \) RHS
3. \( x_0 = U H^{-1} U^H b_s \)  // Galerkin proj.
4. \([x_i, V, H] = \text{eigCG}(nev, m, A, x_0, b_i) \)  // eigCG part
5. \( \tilde{V} = \text{orthogonalize } V \text{ against } U \)  // (not strictly needed)
6. \([U, H] = \text{RayleighRitz}[U, \tilde{V}] \)
7. \( \textbf{end for} \)
Incremental eigCG framework: summary

- **Incremental phase**

  For the first \( s_1 \) RHS: call \( \text{eigCG}(k, m) \) solver, add \( k \) eigenvectors to a separate subspace after each RHS.

- **Init-CG phase**

  For all subsequent RHS (i.e., \( s > s_1 \)): apply Galerkin oblique projection to deflate an initial guess with Ritz vectors generated by the Incremental eigCG, then call standard CG.
eigCG(nev, m) implementation in the QUDA library

create an eigenvector set: \( V = [0 : m] \)

start CG iterations

an extra iteration index: \( i = 0 \)

load the Lanczos vectors: \( V[i] \leftarrow r_i / \| r_i \| \)

construct the Lanczos matrix: \( T_m \)

if \( i == m \):

   apply RR on \( T_m, T_{m-1} \rightarrow Y_m, Y_{m-1} \) (nev lowest eigenpairs)

   QR factorize \( Y_m, Y_{m-1} \rightarrow Q = \text{orth}[Y_m, Y_{m-1}] \)

   set \( H = Q^\dagger T_m Q \) and apply RR on \( H : HZ = Z\Lambda \)

   restart \( V : V = V(QZ) \)

   reset \( i = 2 \times \text{nev} \) and rebuild \( T_m \)

end if

continue CG iterations until the next restart (\( m - 2\text{nev} \) iters)
Eigenvectors in QUDA

- Main requirement is to keep QUDA functionality:
  - application of $\mathcal{D}$
  - blas operations provided by QUDA

- Added extra attributes and members in spinor field classes:
  - ColorSpinorParam
  - ColorSpinorField
  - cudaColorSpinorField

- Allows to work with both the whole eigenvector set and individual eigenvectors
Eigenvectors in QUDA: cont.

```cpp
class ColorSpinorParam : public LatticeFieldParam {

    ...,
    int spinorset_dim;
    int spinorset_id;
    ...
};

class ColorSpinorField : public ColorSpinorParam {

    ...,
    int spinorset_dim;
    int spinorset_id;
    int spinorset_volume;
    ...
    std::vector<ColorSpinorField*> spinorset;
    ...
};

class ColorSpinorField : public ColorSpinorParam {

    ...
    cudaColorSpinorField& SpinorsetItem(const int idx) const;
    ...
};
```
To create an eigenvector set:

```c++
cudaParam.create = QUDA_ZERO_FIELD_CREATE;
cudaParam.spinorset_dim = m;
cudaColorSpinorField *evects = new cudaColorSpinorField(cudaParam);
...```

To work with an individual eigenvector:

```c++
DiracMdagM m(dirac);

m(..., evect→SpinorsetItem(i), ...);
...

CDotProductCuda(evect→SpinorsetItem(i), evect→SpinorsetItem(j));
...```
LA routines for the eigCG solver

- currently relies on MAGMA GPU library:
  - highly optimized lapack-like routines,
    `magma_zgeqrf_gpu(...)`, `magma_zunmqr(...)`, etc.
  - but no multi-process support
What kind of LA operations do we need?

- RR block:

```plaintext
if i == m :

1. $T_m Y = Y \Lambda$, $T_{m-1} \bar{Y} = \bar{Y} \Lambda$ (at most $m$-dim eigenproblem)
2. $Q = \text{orth}[Y, \bar{Y}]$ (2 * nev $m$-component vectors)
3. $H = Q^\dagger T_m Q$ (2nev $\times$ 2nev output matrix)
4. $HZ = Z\Lambda$ (2nev-dim eigenproblem)
5. $Q = (QZ)$ (m $\times$ 2nev output matrix)
6. $V = VQ$ (here we need multi-gpu!)

endif
```
What kind of LA operations do we need? cont.

RR block:

\[
\text{if } i == m : \\
1. \quad T_m Y = Y \Lambda, \ T_{m-1} \tilde{Y} = \tilde{Y} \Lambda \quad (\text{magma\textunderscore zheev\textunderscore gpu()}) \\
2. \quad Q = \text{orth}[Y, \tilde{Y}] \quad (\text{magma\textunderscore zgeqrf\textunderscore gpu()}) \\
3. \quad H = Q^\dagger T_m Q \quad (\text{magma\textunderscore zunmr\textunderscore gpu()}) \\
4. \quad HZ = Z \Lambda \quad (\text{magma\textunderscore zheev\textunderscore gpu()}) \\
5. \quad Q = (QZ) \quad (\text{magma\textunderscore zgemm()}) \\
6. \quad V = VQ \quad (\text{here we need multi-gpu!}) 
\]

endif
NVIDIA Kepler anatomy

K110B micro-architecture highlights
(Tesla K40m)

- 12GB RAM, BW up to 288GB/s
- 15 SMX units, 2880 cores (192 per SMX)
- $P_{\text{theor.}} = 1.43/4.29\text{GFlops}$
- Dynamic parallelism, Hyper-Q, GPUdirect

Increased memory size $\rightarrow$ very essential for the deflated solvers!
Twisted mass fermion action

Lattice volume: $24^3 \times 48$

Two configurations:
- $\kappa = 0.161231, \mu = 0.0085$
- $\kappa = 0.163270, \mu = 0.0040$

eigCG parameters: $nev = 8, m = 128, \, tol = 10^{-10}, \, tol_{rest} = 5e^{-7}$

Used 4-GPU K40m node @ JLAB and 2-GPU K40m node @ FNAL
Incremental eigCG convergence

The degenerate twisted mass fermions, $\kappa = 0.163270, \mu = 0.0040$

- InitCG double-single mixed precision

![Convergence of 48 successive linear systems, L=24,T=48](image)

Tesla K40m
@ rec 8 gauge

A. Strelchenko
Extending the QUDA library with the eigCG solver
Incremental eigCG convergence

The degenerate twisted mass fermions, \( \kappa = 0.163270, \mu = 0.0040 \)

- InitCG double-half mixed precision
Incremental eigCG convergence

The degenerate twisted mass fermions, $\kappa = 0.161231, \mu = 0.0085$

- InitCG double-single mixed precision
Incremental eigCG convergence

The degenerate twisted mass fermions, $\kappa = 0.161231, \mu = 0.0085$

- InitCG double-half mixed precision
Eigenvectors accuracy

The degenerate twisted mass fermions, \( \kappa = 0.163270, \mu = 0.0040 \)
Eigenvector accuracy

The degenerate twisted mass fermions, $\kappa = 0.161231$, $\mu = 0.0085$
Incremental eigCG: performance summary

- Typical timings for the incremental stage ($\kappa = 0.163270, \mu = 0.0040$):
  - $[44.4, 42.6, 39.6, 29.3, 27.9, 24.7, 23.5, 21.7, 20.9, 19.8, 16.8, 15.1, \ldots ]$

- Average DS CG exec. time:
  - non-deflated CG: 15 secs
  - deflated CG: 2.42 secs

- Average DH CG exec. time:
  - non-deflated CG: does not converge
  - deflated CG: 1.84 secs
Deflated CG : need for restarting

- No restart

- Restart
Conclusion

- **Incremental eigCG efficiency:**
  - ×8 speedup in terms of iterations
  - ×8 speedup in execution time for initCG stage
  - requires reliable updates with Reighley-Ritz for eigCG stage

- **Future work:**
  - eigBiCGstab
  - GMRES-DR