Can Block Iterative Eigensolvers Fulfill their Performance Promise?

Jonas Thies  Melven Röhrig-Zöllner  Achim Basermann

DLR Simulation and Software Technology
Distributed Systems and Component Software
High Performance Computing Group

project ESSEX
Motivation

Mathematical problem

- Find $O(10)$ eigenpairs
  
  $$Ax_i = \lambda_i x_i$$

  of a large, sparse matrix $A$.

- Interior or extreme $\lambda_i$,
- symmetric or general $A$.

Memory gap

- Small memory bandwidth vs. high Peak Flop rate

→ Increase the compute intensity
Regular vs. block JDQR

1: start from initial spaces
   \[ V^H V = I, \quad V^A := AV, \quad H = V^H AV \]
2: while ... do
3:   (sorted) Schur decomp.,
   \[ H = S T S^H, S^H S = I \]
4:   select shift \( \theta = T_{1,1} \)
5:   \[ u = Vs_1, u^A = V^A s_1 \]
6:   \[ r = u^A - \theta u \]
7:   (lock converged eigenpairs in \( Q \))
8:   (shrink subspaces if necessary)
9:   \[ \tilde{Q} = [Q \ u] \]
   solve approximately for \( t \perp \tilde{Q} \):
10: \[ (I - \tilde{Q} \tilde{Q}^H)(A - \theta I)(I - \tilde{Q} \tilde{Q}^H)t = -r \]
11: orthogonalize \( t \) against \( V \)
12: extend \( V = [V \ \tilde{t}] \), \( V^A, H = V^H V^A \)
13: end while
Regular vs. block JDQR

1: start from initial spaces
\[ V^H V = I, \ V^A := AV, \ H = V^H AV \]

2: while ... do
3: (sorted) Schur decomp.,
\[ H = STS^H, \ S^H S = I \]
4: select shift \( \theta = T_{1,1} \)
5: \( u = Vs_1, u^A = V^As_1 \)
6: \( r = u^A - \theta u \)
7: (lock converged eigenpairs in \( Q \))
8: (shrink subspaces if necessary)
9: \( \tilde{Q} = [Q \ u] \)
solve approximately for \( t \perp \tilde{Q} \):
10: \( (I - \tilde{Q} \tilde{Q}^H)(A - \theta I)(I - \tilde{Q} \tilde{Q}^H)t = -r \)
11: orthogonalize \( t \) against \( V \)
12: extend \( V = [Vt, \ V^A, \ H = V^H V^A] \)
13: end while

(\ldots)
(\ldots)

while ... do
(\ldots)
(\ldots)
select shifts \( \theta_i = T_{i,1}, i = 1 \ldots n_b \)
\( u = VS_{:,1:n_b}, u^A = V^AS_{:,1:n_b} \)
\( r_{:,i} = u^A_{:,i} - \theta_i u_{:,i} \)
(\ldots)
(\ldots)
\( \tilde{Q} = [Q \ u] \)
solve \( n_b \) independent systems
\( (I - \tilde{Q} \tilde{Q}^H)(A - \theta_i I)(I - \tilde{Q} \tilde{Q}^H)t_{:,i} = -r_{:,i} \)
block-orthogonalize \( t \) against \( V \),
extend subspaces (by \( n_b \) vecs)
end while
Numerical behavior

Block size 2

Block size 4

relative overhead (# spMVM)

# eigenvalues found

Andrews

cfd1

cry10000

dw8192

torsion1

cfinan512

relative overhead (# spMVM)

# eigenvalues found

Andrews

cfd1

cry10000

dw8192

torsion1

cfinan512
Kernels needed for BJDQR

- spMMVM: sparse Matrix times Multiple Vector Multiplication
- block projection, \( y \leftarrow (I - VV^H)x \)
- block vector orthogonalization
  given \( V, W : V^H V = I \), find \( Q, R : W = QR, Q^H Q = I, V^H Q = 0. \)
- subspace transformation for ‘thick restart’
  \( V_{:,1:k} \leftarrow V_{:,1:m} \cdot S, S \in \mathbb{C}^{m \times k}, k < m \)
- preconditioning operation approximating \((A - \tau I)^{-1}x\)
  (so far unpreconditioned blocked Krylov \(\rightarrow\) ESSEX-II)
Row major vs. column major storage

\begin{align*}
y_j & \leftarrow (A - \theta_j)x_j \\
S & \leftarrow Q^T Y \\
Y & \leftarrow Y - QS
\end{align*}

Tpetra/TBB, col major
Row major vs. column major storage

\[ y_j \leftarrow (A - \theta_j)x_j \]
\[ S \leftarrow Q^T Y \]
\[ Y \leftarrow Y - QS \]

Tpetra/TBB, col major

GHOST, col major

Block Sparse Solvers

• consequences for overall implementation (avoid strides!)

SELL-C- \( \sigma \) most helpful if SIMD/SIMT width > \( n_b \)
Row major vs. column major storage

\[ y_j \leftarrow (A - \theta_j)x_j \]
\[ S \leftarrow Q^T Y \]
\[ Y \leftarrow Y - QS \]

<table>
<thead>
<tr>
<th>block size ( n_b )</th>
<th>runtime [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

- **Tpetra/TBB, col major**
- **GHOST, col major**
- **GHOST, row major**

**consequences for overall implementation:** Avoid strides!

**SELL-C:** Most helpful if SIMD/SIMT width > \( n_b \)
Row major vs. column major storage

- consequences for overall implementation (avoid strides!)
- SELL-C-σ most helpful if SIMD/SIMT width > \( n_b \)
Overall block speedup (strong scaling)

Setup

- Non-symmetric matrix from 7-point 3D PDE discretization ($n \approx 1.3 \cdot 10^8$, $n_{nz} \approx 9.4 \cdot 10^8$)
- Seeking 20 eigenvalues
- Ivy Bridge Cluster

Results

- $n_b = 2$: significantly faster
- $n_b = 4$: no further improvement

See Röhrig-Zöllner et al SISC 37(6), 2015
Block orthogonalization schemes

- Given orthogonal vectors \((v_1, \ldots, v_k) = V\)
- For \(X \in \mathbb{R}^{n \times nb}\) find orthogonal \(Y \in \mathbb{R}^{n \times \tilde{nb}}\) with

\[
YR_1 = X - VR_2, \quad \text{and} \quad V^TY = 0
\]

Two phase algorithms

**Phase 1** Project: \(\tilde{X} \leftarrow (I - VV^T)X\)

**Phase 2** Orthogonalize: \(Y \leftarrow f(\tilde{X})\)

- communication optimal \(f\):
  - CholQR or SVQB (Stathopoulos and Wu, SISC 2002)
  - TSQR (Demmel et al., SISC 2012)
- Each phase messes with the accuracy of the other. \(\rightarrow\) iterate
Raising the computational intensity

Kernel fusion

Phase 2 \[ \tilde{X} \leftarrow XB, \quad C \leftarrow V^T\tilde{X} \]
Phase 1 \[ \tilde{X} \leftarrow X - VC, \quad \tilde{B} \leftarrow \tilde{X}^T\tilde{X} \]
Phase 3 \[ \tilde{X} \leftarrow XB, \quad \tilde{B} \leftarrow \tilde{X}^T\tilde{X} \]
⇒ use SVQB

Increased precision

Idea Calculate value and error of each arithmetic operation
- Store intermediate results as double-double (DD) numbers
- Based on arithmetic building blocks (2Sum, 2Mult)
  Muller et al.: Handbook of Floating-Point Arithmetic, Springer 2010
- Exploit FMA operations (AVX2)
Results: accuracy after one iteration

Error in $V^TY$ vs. condition number $\kappa(X, V)$

Error in $Y^TY$ vs. condition number $\kappa(X)$

- Project + TSQR
- Project / SVQB
- DD Project / SVQB

Different $\kappa(X, W)$

Breakdown!
Results: runtime to orthogonality

- Iterated project/SVQB
- " " with fused operations
- " " with fused DD operations

Diagram shows runtime to orthogonality for different block sizes and projection sizes.
Faster through

- block algorithms
- kernel fusion
- increased precision arithmetic (not data)
Summary

Lessons learned

• don’t use BLAS-3 for ‘tall, skinny matrices’
• row-major storage
• implement algorithms accordingly (avoid strides)
• use ‘free’ operations to increase accuracy/reduce iterations

Outlook: Peta and beyond

• reduction of synchronization kicks in
• increasing memory gap favors block methods
• crucial for ESSEX-II: matrix reordering and preconditioning