Towards High Performance Algorithms for the Symmetric Eigenvalue Problems

Azzam Haidar

ICL January 14, 2011
General Overview: the Eigenproblem algorithms

Two-stages approach:

1. Non Symmetric EVP
   • Hessenberg Reduction + QR iteration.

2. Symmetric EVP
   • Tri-Diagonalization Reduction + Divide-and-Conquer.

3. Singular Value Decomposition
   • Bi-Diagonalization Reduction + Divide-and-Conquer.
General Overview: the Eigenproblem algorithms

Two-stages approach:

1. Non Symmetric EVP
   • Hessenberg Reduction + QR iteration.

2. Symmetric EVP
   • Tri-Diagonalization Reduction + Divide-and-Conquer.

3. Singular Value Decomposition
   • Bi-Diagonalization Reduction + Divide-and-Conquer.
The standard Tridiagonal reduction xSYTRD

* LAPACK SYTRD:

1. Apply left-right transformations $Q \mathbf{A} Q^*$ to the panel $
   \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix}
   
2. Update the remaining submatrix $A_{33}$

$$
\begin{pmatrix}
T_{11} & T_{21}^T & 0 \\
T_{21} & A_{22} & A_{32}^T \\
0 & A_{32} & A_{33}
\end{pmatrix}
\equiv
\begin{pmatrix}
T_{11} & T_{21}^T & 0 \\
T_{21} & A_{22} & A_{32}^T \\
0 & A_{32} & A_{33}
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
T_{11} & T_{21}^T & 0 \\
T_{21} & A_{22} & A_{32}^T \\
0 & T_{23} & A_{33}
\end{pmatrix}
$$

where $A_{33} = A_{33} - YW^T - WY^T$

---

**phase 1**

**phase 2**

\textbf{step $k$:} $Q \mathbf{A} Q^*$ then update $\Rightarrow$ \textbf{step $k+1$}
The standard Tridiagonal reduction \texttt{xSYTRD}

\begin{itemize}
  \item \textbf{Characteristics}
  \begin{enumerate}
    \item Phase 1 requires:
      \begin{itemize}
        \item 4 panel vector multiplications,
        \item 1 symmetric matrix vector multiplication with $A_{33}$,
        \item Cost $2(n-k)^2b$ Flops.
      \end{itemize}
    \item Phase 2 requires:
      \begin{itemize}
        \item Symmetric update of $A_{33}$ using \texttt{SYRK},
        \item Cost $2(n-k)^2b$ Flops.
      \end{itemize}
  \end{enumerate}
  \item \textbf{Observations}
  \begin{itemize}
    \item Too many Blas-2 op,
    \item Relies on panel factorization,
    \item Total cost $4n^3/3$
    \item \textbf{Bulk sync phases},
    \item \textbf{Memory bound algorithm}.
  \end{itemize}
\end{itemize}
The tile reduction to Band

**Tile Band tridiagonal:**

**Characteristics**
- Split operation into tiles/panel op,
- Most op relies on Blas-3/Blas-2,
- Extract parallelism/sync phases,
- Scheduling task execution,
- BDL data layout, overlap between data transfer computation/memory bound algo.
The tile reduction to Band

nz = 512
Parallel Performance on a quad-socket quad-core Intel Xeon 2.4 GHz processors with MKL BLAS V10.0.1.
**Observations**

- **Reduction achieved ONLY to band forms.**

- **Need to go to FULL reduction?**

1. **Bulge Chasing:**
   - Relies on Blas-1 operations,
   - Most of the existing techniques consists on sequential operations,
   - Expensive, memory bound and lack of efficiency,

2. **Investigate research on band solvers:**
   - Re-think about techniques such as Band Divide and Conquer,
   - Open question...
Observations

- Reduction achieved **ONLY** to band forms.

- **Need to go to FULL reduction?**

1. **Bulge Chasing:**
   - Relies on Blas-1 operations,
   - Most of the existing techniques consists on sequential operations,
   - Expensive, memory bound and lack of efficiency,

2. **Investigate research on band solvers:**
   - Re-think about techniques such as Band Divide and Conquer,
   - Open question...
The Divide and Conquer algorithm stage -2-

Target: Tridiagonal matrices:

★ Basic concepts:

1. **Subdivision** of $T$ into independent sets of tridiagonal blocks

2. **Solution** of the each of the “simple” eigenproblem ($T_1$, $T_2$)

3. Recursively, **Merge** each 2 blocks.
The Divide and Conquer algorithm stage -2-

**Target:** Tridiagonal matrices:

**Basic concepts:**

So the tridiagonal matrix $T$ can be expressed as:

$$
T = \alpha vv^T + \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + \alpha vv^T
$$

$$
= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \alpha vv^T
$$

$$
= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + \alpha zz^T \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T
$$

$$
= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix} \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix}^T \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T
$$

The Divide and Conquer algorithm stage -2-

Target: Tridiagonal matrices:

**Basic concepts:**

So the tridiagonal matrix $T$ can be expressed as:

$$T \equiv \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + \alpha \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^{T} \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix}^{T} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^{T}$$
The Band Divide and Conquer algorithm stage -2-

Target: Band matrices:

★ Basic concepts:

1. **Subdivision** of $A$ into independent sets of tridiagonal blocks

2. **Solution** of the each of the "simple" eigenproblem $(A_1, A_2)$

3. Recursively, **Merge** each 2 blocks.

\[
A = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \sigma_1 z_1 z_1^T + \sigma_2 z_2 z_2^T + \cdots + \sigma_b z_b z_b^T
\]

\[
= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{21} & \bar{Q}_{22} \end{pmatrix} \begin{pmatrix} \bar{D}_1 & 0 \\ 0 & \bar{D}_2 \end{pmatrix} \begin{pmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{21} & \bar{Q}_{22} \end{pmatrix}^T \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \sigma_2 z_2 z_2^T + \cdots + \sigma_b z_b z_b^T
\]
### Tridiag v.s. Band: divide and conquer

**Observations:**

<table>
<thead>
<tr>
<th>Tridiagonal D&amp;C</th>
<th>Band D&amp;C</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Inherent natural parallelism,</td>
<td>• Inherent natural parallelism,</td>
</tr>
<tr>
<td>• Have a very low granularity, they scale very well,</td>
<td>• Have a very low granularity, they scale very well,</td>
</tr>
<tr>
<td>• Each merge step requires $1$ rank-one modification, and $1$ GEMV,</td>
<td>• Each merge step requires $k$ rank-one modification and $k$ GEMV,</td>
</tr>
<tr>
<td>• Cost $O(n^2)$</td>
<td>• Cost $6k^2n^2$, where $k$ is the semibandwidth.</td>
</tr>
</tbody>
</table>
Some observations

🌟 In practice:

stage 1: the reduction to band is efficient for \( k = 200 \) and breakdown for \( k < 128 \)

stage 2: D&C cost is \( 6k^2 n^2 \)

NO performance is achieved
Some observations

* In practice:

  stage 1: the reduction to band is efficient for \( k = 200 \)
  and breakdown for \( k < 128 \)

  stage 2: D&C cost is \( 6k^2 n^2 \)

  NO performance is achieved

Need tricks to obtain performance?
# First tricks

Timing (top) and number of tasks $10^3$ (bottom) for the reduction stage -1-

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>4000</th>
<th>8000</th>
<th>12000</th>
<th>16000</th>
</tr>
</thead>
<tbody>
<tr>
<td>k=200</td>
<td>2.6</td>
<td>13</td>
<td>40</td>
<td>90</td>
</tr>
<tr>
<td>k=100</td>
<td>2.5</td>
<td>13</td>
<td>42</td>
<td>96</td>
</tr>
<tr>
<td>k= 50</td>
<td>8</td>
<td>62</td>
<td>214</td>
<td>524</td>
</tr>
<tr>
<td>k= 25</td>
<td>70</td>
<td>480</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
</tr>
<tr>
<td>k= 20</td>
<td>112</td>
<td>765</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>4000</th>
<th>8000</th>
<th>12000</th>
<th>16000</th>
</tr>
</thead>
<tbody>
<tr>
<td>k=200</td>
<td>3</td>
<td>21</td>
<td>72</td>
<td>170</td>
</tr>
<tr>
<td>k=100</td>
<td>21</td>
<td>170</td>
<td>575</td>
<td>1,365</td>
</tr>
<tr>
<td>k= 50</td>
<td>170</td>
<td>1,365</td>
<td>4,607</td>
<td>10,922</td>
</tr>
<tr>
<td>k= 25</td>
<td>1,365</td>
<td>10,922</td>
<td>36,863</td>
<td>87,380</td>
</tr>
<tr>
<td>k= 20</td>
<td>2,665</td>
<td>21,332</td>
<td>71,992</td>
<td>170,665</td>
</tr>
</tbody>
</table>
First tricks

Analysis:

1. If $k=k/2$: each tile will be visited $2^{l-1}$ more times meaning that each tile will be loaded $2^{l-1}$ more times from memory.

2. The computation on small tile is fast enough, that loading from and to the main memory will lead to huge bus traffic.

3. Small tile, result in loosing the efficient of the BLAS-3 operations.

4. Small tiles, will generate a huge number of tasks, that will decreases the scalability and the efficiency of any dynamic scheduler.
First tricks

Analysis:

1. If $k = k/2$: each tile will be visited $2^{l-1}$ more times meaning that each tile will be loaded $2^{l-1}$ more times from memory.

2. The computation on small tile is fast enough, that loading from and to the main memory will lead to huge bus traffic.

3. Small tile, result in loosing the efficient of the BLAS-3 operations.

4. Small tiles, will generate a huge number of tasks, that will decreases the scalability and the efficiency of any dynamic scheduler.

Idea: Use grouping techniques for tasks and data
First tricks

Idea: Use grouping techniques for tasks and data

**Characteristics:**

1. More BLAS-3 operations,
2. Computing time is large enough to overlap loading from memory,
3. Decrease significantly the number of tasks, the scheduler is efficient again,
4. Tiles size vary during each step.
First tricks

Idea: Use grouping techniques for tasks and data

without grouping

with grouping
## First tricks

Timing (top) and number of tasks $10^3$ (bottom) for the reduction stage -1-

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>without grouping</th>
<th></th>
<th>with grouping</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4000</td>
<td>8000</td>
<td>12000</td>
<td>16000</td>
</tr>
<tr>
<td>k=200</td>
<td>2.6</td>
<td>13</td>
<td>40</td>
<td>90</td>
</tr>
<tr>
<td>k=100</td>
<td>2.5</td>
<td>13</td>
<td>42</td>
<td>96</td>
</tr>
<tr>
<td>k=50</td>
<td>8</td>
<td>62</td>
<td>214</td>
<td>524</td>
</tr>
<tr>
<td>k=25</td>
<td>70</td>
<td>480</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
</tr>
<tr>
<td>k=20</td>
<td>112</td>
<td>765</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>without grouping</th>
<th></th>
<th>with grouping</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4000</td>
<td>8000</td>
<td>12000</td>
<td>16000</td>
</tr>
<tr>
<td>k=200</td>
<td>3</td>
<td>21</td>
<td>72</td>
<td>170</td>
</tr>
<tr>
<td>k=100</td>
<td>21</td>
<td>170</td>
<td>575</td>
<td>1,365</td>
</tr>
<tr>
<td>k=50</td>
<td>170</td>
<td>1,365</td>
<td>4,607</td>
<td>10,922</td>
</tr>
<tr>
<td>k=25</td>
<td>1,365</td>
<td>10,922</td>
<td>36,863</td>
<td>87,380</td>
</tr>
<tr>
<td>k=20</td>
<td>2,665</td>
<td>21,332</td>
<td>71,992</td>
<td>170,665</td>
</tr>
</tbody>
</table>
Second tricks

🌟 Analysis:

The solution of the Eigensystem of a symmetric matrix requires:
1. the transformation of the matrix into a nice form (band matrix) stage -1-,
2. The computation of the Eigenvalue of the nice matrix stage -2-.

🌟 Bottleneck:

1. Bulk-synchronization between these two stages.

Idea: Remove the synchronization
Second tricks

Idea: Remove the synchronization

reduction phase  reduction + divide and conquer  divide and conquer
Experimental results
Tuning

effect of the bandwidth size (matrix size = 16000 on 16 threads)

- Band reduction
- Band Divide and Conquer
- Total Eigensolver

Elapsed time in seconds on a quad-socket quad-core Intel Xeon 2.4 GHz processors with MKL V10.2.4
Performance Comparison

Comparison between open-source (OS) and Vendors (V) libraries

Matrix size

Elapsed time in seconds on a quad-socket quad-core Intel Xeon 2.4 GHz processors with MKL V10.2.4
Performance Comparison

Testing matrices – type 1

Testing matrices – type 2

Testing matrices – type 3

Testing matrices – type 4

Elapsed time in seconds on a quad-socket quad-core Intel Xeon 2.4 GHz processors with MKL V10.2.4
Performance Comparison

Elapsed time in seconds on a quad-socket quad-core Intel Xeon 2.4 GHz processors with MKL V10.2.4
Performance Comparison

Elapsed time in seconds on a quad-socket quad-core Intel Xeon 2.4 GHz processors with MKL V10.2.4
Future work

Road map and open questions:

• Study the possibility of applying once a rank-k modification?
• Develop similar approach for SVD,
• Offloading of the most-expensive operations into the GPU.
• Think about reduction from band to tridiagonal.
• Evaluate the bulge chasing techniques, and the possibility to accelerate those sequential approaches?
• Effort might be made on parallelizing those sequential algorithm.
Future work

🌟 Reduction achieved **ONLY** to band forms.

🌟 Need to go to **FULL** reduction?

1. **Bulge Chasing:**
   - Relies on Blas-1 operations,
   - Most of the existing techniques consists on sequential operations,
   - Expensive, memory bound and lack of efficiency,

2. **Investigate research on band solvers:**
   - Re-think about techniques such as Band Divide and Conquer,
   - Open question...
Future work

🌟 Reduction achieved **ONLY** to band forms.

🌟 Need to go to **FULL** reduction?

1. **Bulge Chasing:**
   - Relies on Blas-1 operations,
   - Most of the existing techniques consists on sequential operations,
   - Expensive, memory bound and lack of efficiency,

2. **Investigate research on band solvers:**
   - Re-think about techniques such as Band Divide and Conquer,
   - Open question...
Bulge Chasing - The algorithm

Grouping to increase cache reuse

Thread based locality

nz = 75
Bulge Chasing - Preliminary results

Performance of the full reduction

Matrix size

GFLOPS

- Preliminary algo
- D&C
- MKL-SBR
- MKL-LPK
- REF-LPK

Elapsed time in seconds on a quad-socket quad-core Intel Xeon 2.4 GHz processors with MKL V10.2.4
Thank you for your attention