A Memory-Efficient Algorithm for Large-Scale Symmetric Tridiagonal Eigenvalue Problem on Multi-GPU Systems

Hyunsu Cho and Peter A. Yoon
Trinity College, Hartford, CT, USA
Symmetric Eigenvalue Problem

$$Ax = \lambda x$$

where $A$ is symmetric

Many interesting applications require eigenvectors
Divide and Conquer

Yields **full spectrum** of eigenvalues and eigenvectors
Is numerically stable
Gives rise to **independent subproblems**
Often faster than $O(n^3)$ due to deflation
Divide and Conquer

Apply **orthogonal similarity transformation** to reduce $A$ to tridiagonal form

$$Q^T A Q = A'$$

where

$A'$ is symmetric tridiagonal
and $Q$ is orthogonal

Existing work on single-node, multi-GPU:
MAGMA (UTK)
Divide and Conquer

- Solve subproblems
- Merge solutions
- Repair
Divide and Conquer
Merging solutions

Suppose

\[ A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} + \begin{bmatrix} b_m \\ b_m \end{bmatrix} \]

where

\[ A_1 = Q_1 D_1 Q_1^T \]  (subproblem #1)
\[ A_2 = Q_2 D_2 Q_2^T \]  (subproblem #2)
Merging solutions

Then

\[ A = QDQ^T + \begin{bmatrix} b_m & b_m \\ b_m & b_m \end{bmatrix} \]

where

\[ Q = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} \]
Merging solutions

Then

\[ A = Q D Q^T + \begin{bmatrix} b_m & b_m \\ b_m & b_m \end{bmatrix} \]

where

\[ Q = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} \]

Rank-one modifier

\[ H = b_m \begin{bmatrix} e_m \\ e_1 \end{bmatrix} \begin{bmatrix} e_m \\ e_1 \end{bmatrix}^T \]
Rank-one update

\[ H = b_m \begin{bmatrix} \frac{e_m}{e_1} \\ e_1 \end{bmatrix} \begin{bmatrix} \frac{e_m}{e_1} \\ e_1 \end{bmatrix}^T \]

\[ A = Q D Q^T + H = Q (D + b_m z z^T) Q^T \]

where

\[ z = Q^T \begin{bmatrix} e_m \\ e_1 \end{bmatrix} = \begin{bmatrix} \text{last column of } Q_1^T \\ \text{first column of } Q_2^T \end{bmatrix} \]
Rank-one update

\[ H = b_m \left[ \frac{e_m}{e_1} \right] \left[ \frac{e_m}{e_1} \right]^T \]

\[ A = QDQ^T + H = Q(D + b_m z z^T)Q^T \]

where

\[ z = Q^T \left[ \frac{e_m}{e_1} \right] = \left[ \begin{array}{c} \text{last column of } Q_1^T \\text{first column of } Q_2^T \end{array} \right] \]

Need eigen-decomposition of inner system
Decompose $D + b_m^T z z^T$

1. Sort entries in $D$; permute $z$ likewise
2. Filter some entries in $D$ and $z$ via deflation (next slide)
Decompose $D + b_m z z^T$

1. Sort entries in $D$; permute $z$ likewise
2. Filter some entries in $D$ and $z$ via deflation (next slide)
3. Compute all roots of the **secular equation** [1]

\[
1 + b_m \sum_{i=1}^{n} \frac{d_i^2}{z_i - \lambda} = 0,
\]

...giving the $m$ eigenvalues.

4. Compute corresponding eigenvectors stably [2]

[1] Li 1994
Decompose $D + b_m zz^T$

1. Sort entries in $D$; permute $z$ likewise
2. Filter some entries in $D$ and $z$ via deflation (next slide)
3. Compute all roots of the **secular equation** [1]
   
   $$1 + b_m \sum_{i=1}^{n} \frac{d_i^2}{z_i - \lambda} = 0,$$
   
   giving the $m$ eigenvalues.

4. Compute corresponding eigenvectors stably [2]

5. Multiply each eigenvector by $Q$

Recall: $A = Q(D + b_m zz^T)Q^T$

[1] Li 1994
Deflation

Recall:

\[ D = \begin{bmatrix} D_1 & \mid & D_2 \end{bmatrix} \]

Entries of \( D \) are eigenvalues of two subproblems
If two entries are nearly identical, we throw one away

**Fewer columns** when multiplying eigenvectors by \( Q \)
Same thing for small entries in \( z \)

Reduce work complexity to \( O(n^{2.3}) \)
GPU computing

General-purpose computation on GPUs
Bulk parallelism w/ many small threads
Cost effective; widely available
Mapping work to GPU

1. Sort entries in $D$; permute $z$ likewise
2. Filter some entries in $D$ and $z$ via deflation
3. Compute all roots of the secular equation, giving the $m$ eigenvalues.
4. Compute corresponding eigenvectors stably
5. Multiply each eigenvector by $Q$  
   → Done in bulk via DGEMM

Parallel but not as work-intense
GPU memory

High-bandwidth dedicated memory
Separate from main memory
Limited in size
Memory requirement

Eigenvectors are dense → $O(n^2)$ storage

Intermediate workspace: eigenvectors of inner system

<table>
<thead>
<tr>
<th>Matrix dimension</th>
<th>Memory required</th>
</tr>
</thead>
<tbody>
<tr>
<td>8192</td>
<td>1.5 GB</td>
</tr>
<tr>
<td>16384</td>
<td>5.8 GB</td>
</tr>
<tr>
<td>32768</td>
<td>23.4 GB</td>
</tr>
<tr>
<td>36000</td>
<td>28.2 GB</td>
</tr>
<tr>
<td>50000</td>
<td>54.4 GB</td>
</tr>
</tbody>
</table>
Our contribution

Overcome limitation in GPU memory while retaining adequate performance
Strategies

1. Use multiple GPUs
Strategies

2. Keep most of workspace in main memory (out-of-core approach)
Strategies

3. **Shape work** to fit GPU workspaces
Block matrix multiplication
Use a fine partition to fit submatrices into GPU memory

\[ Q \times \text{Eigenvectors of } D + b_m z z^T = \text{Eigenvectors of } A \]
Hybrid computation

Allocate subproblems to both GPUs and CPUs
Model performance as a power function

Profiler fits parameters using least-squares

\[ R^2 = 0.9566 \]
Hybrid computation

Solve many subproblems in parallel
Hybrid computation

Solve each subproblem by parts
Results

Scales to 50k * 50k matrix
With 4 GB of GPU memory

Per-GPU peak memory usage

Main memory: 64 GB
GPU memory: 5 GB per GPU
Results

Performance: vs. multicore CPU

CPU: dual Intel® Xeon® E5-2620
GPU: 4 Nvidia Tesla® K20c
Conclusion

Out-of-core approach overcomes memory limitation on the GPU

Hybrid computation with profiling delivers reasonable performance
Acknowledgment

Trinity College, Student Research Program
Nvidia Corporation, CUDA Teaching Center Program
Any questions?