Extreme-Scale Eigenvalue Reordering in the Real Schur Form

Lars Karlsson, Mirko Myllykoski, Björn Adlerborn, Carl Christian Kjelgaard Mikkelsen, and Bo Kågström

Department of Computing Science and HPC2N
Umeå University

SIAM-CSE-2017
The problem
The kernel

- Solve tiny Sylvester equation
  \[ AX - XB = \gamma C. \]
- Compute small QR factorization
  \[
  \begin{bmatrix}
  -X \\
  \gamma I
  \end{bmatrix} = QR
  \]
- Apply orthogonal transformation
Scalar code (DTRSEN, LAPACK)

- Slide each eigenvalue up the diagonal as in bubble-sort
- Low arithmetic intensity
- Poor cache utilization
- Move eigenvalues within small window
- Apply transform to block rows and columns
- Slide window up and repeat
Parallel blocked code (PBDTRSEN, SCALAPACK)
Task based programming

Figure: Move a single block to the top
<table>
<thead>
<tr>
<th>symbol</th>
<th>operation</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Swap adjacent blocks</td>
<td>$S(3, 4)$</td>
</tr>
<tr>
<td>R</td>
<td>Right update of block column</td>
<td>$R(i &lt; 3, 3 : 4)$</td>
</tr>
<tr>
<td>L</td>
<td>Left update of block row</td>
<td>$L(3 : 4, 3 &lt; j)$</td>
</tr>
</tbody>
</table>
Compressed DAG

$R(i < 4, 4:5)$

$S(4,5)$

$R(i < 3, 3:4)$

$S(3,4)$

$R(i < 2, 2:3)$

$S(2,3)$

$L(3:4, 4 < j)$

$L(2:3, 3 < j)$

$L(1:2, 2 < j)$

start

end
An advantage of task based programming ...
<table>
<thead>
<tr>
<th>BDTRSEN courtesy of Kressner - Thank you very much!</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Blocked code</td>
</tr>
<tr>
<td>• Move eigenvalues within windows</td>
</tr>
<tr>
<td>• Delays update of matrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PBDTRSEN (ScaLAPACK) (Granat, Kressner, Kågström)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Parallel block code</td>
</tr>
<tr>
<td>• Uses multiple windows</td>
</tr>
<tr>
<td>• Global synchronization after row/column updates</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task based code running under StarPU (Mirko Myllykoski)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Uses multiple windows</td>
</tr>
<tr>
<td>• Uses BDTRSEN to process windows</td>
</tr>
<tr>
<td>• Low level synchronization among threads</td>
</tr>
</tbody>
</table>
Real Schur forms are built from a seed and parameters

- $n$, the dimension of the problem

  \[ n \in \{10000, 20000, 30000, 40000\} \]

- $k$, the number of 2 by 2 real blocks

  \[ 2k = \frac{n}{2} \]

- $p$, the probability of choosing any diagonal block

  \[ p \in \{0.05, 0.15, 0.35, 0.50\} \]

Eigenvalues are drawn from a grid of well separated points

- Ensures that “all” Sylvester equations well conditioned
### Machine K* (operational 2016)

- Intel Xeon E5-2690v4
- 2 NUMA nodes per node
- 14 cores per NUMA node
- Each core has its own FPU
- Cores drawn from list 0:1:27

*Standard compute node on Kebnekaise at High Performance Computing Center North (HPC2N)*
Accuracy

- For all $\lambda \in \lambda(A)$, we have
  \[
  \frac{|\lambda - \tilde{\lambda}|}{|\lambda|} \lesssim 900u.
  \]

- In addition, we have
  \[
  \frac{\|Q^T AQ - \tilde{A}\|_F}{\|A\|_F} \lesssim 190u
  \]
  and
  \[
  \frac{\|Q^T Q - I\|_F}{\|I\|_F} \lesssim 315u.
  \]
Time to solve

MPI versus StarPU / Kebnekaise / 5% selected

MPI (28 ranks)  StarPU (28 workers)

Runtime [s]
Matrix dimension

Runtime [s]
Matrix dimension
Time to solve

MPI versus StarPU / Kebnekaise / 15% selected

MPI (28 ranks)
StarPU (28 workers)
Time to solve

MPI versus StarPU / Kebnekaise / 35% selected

MPI (28 ranks) | StarPU (28 workers)
---|---
18 | 44
Time to solve

MPI versus StarPU / Kebnekaise / 50% selected

Runtime [s] vs Matrix dimension for MPI (28 ranks) and StarPU (28 workers). The graph shows a comparison of the runtime across different matrix dimensions, with a focus on the selected 50% configuration.
Best serial code?

StarPU runtime / Kebnekaise / 5% selected

- BDTRSEN
- 1 worker
- 4 workers
- 12 workers
- 20 workers
- 28 workers

Runtime [s] vs. Matrix dimension
Best serial code?

StarPU runtime / Kebnekaise / 15% selected

BDTRSEN
1 worker
4 workers
12 workers
20 workers
28 workers

0
500
1000
1500
2000
2500
10000 20000 30000 40000

Runtime [s]

Matrix dimension

21 / 44
Best serial code?

![StarPU runtime / Kebnekaise / 35% selected](image_url)

- **BDTRSEN**: purple
- **1 worker**: green
- **4 workers**: light blue
- **12 workers**: orange
- **20 workers**: yellow
- **28 workers**: dark blue

- **Matrix dimension**
- **Runtime [s]**

![Runtime and Matrix dimension graph](image_url)
Best serial code?

StarPU runtime / Kebnekaise / 50% selected

- BDTRSEEN
- 1 worker
- 4 workers
- 12 workers
- 20 workers
- 28 workers

Matrix dimension vs. Runtime [s]
Strong scalability, StarPU

![Graph showing efficiency vs. CPU cores/StarPU workers for different values of N (10000, 20000, 30000, 40000). The graph illustrates the strong scalability of StarPU with Kebnekaise, focusing on 5% selected.](image-url)
Strong scalability, StarPU

Strong scalability / StarPU / Kebnekaise / 15% selected

Efficiency vs. CPU cores / StarPU workers

N = 10000
N = 20000
N = 30000
N = 40000

N = 10000
N = 20000
N = 30000
N = 40000

Efficiency

CPU cores / StarPU workers
Strong scalability, StarPU
Strong scalability, StarPU

![Graph showing strong scalability with StarPU on Kebnekaise with 50% selected. The graph plots efficiency against CPU cores/StarPU workers for different values of N (10,000 to 40,000). The efficiency decreases as the number of CPU cores increases.]
Weak scalability

Weak scalability / StarPU / Kebnekaise

N = 40,000 for 28 cores

Efficiency

CPU cores / StarPU workers

50% selected
35% selected
15% selected
5% selected

0 0.2 0.4 0.6 0.8 1
4 8 12 16 20 24 28

Efficiency

CPU cores / StarPU workers
Flop rate

Floating-point performance / Kebnekaise / 5% selected

CPU cores / StarPU workers

Efficiency

N = 40000
N = 30000
N = 20000
N = 10000

Efficiency vs. CPU cores / StarPU workers
Flop rate

Floating-point performance / Kebnekaise / 15% selected

Efficiency

CPU cores / StarPU workers

N = 40000
N = 30000
N = 20000
N = 10000
Flop rate

Floating-point performance / Kebnekaise / 35% selected

Efficiency

CPU cores / StarPU workers

N = 10000
N = 20000
N = 30000
N = 40000

Efficiency

CPU cores / StarPU workers
Floating-point performance / Kebnekaise / 50% selected

Efficiency vs. CPU cores / StarPU workers for different values of N.

- N = 40000
- N = 30000
- N = 20000
- N = 10000

The graph shows the efficiency as a function of the number of CPU cores and StarPU workers for various N values, with a 50% selected floating-point performance.
Idle time

Runtime distribution / Kebnekaise / 5% selected

Portion of execution time

Overhead
Idle
Executing

Matrix dimension : StarPU workers

10000 20000 30000 40000

0 0.2 0.4 0.6 0.8 1
Idle time

Runtime distribution / Kebnekaise / 15% selected

Portion of execution time

Overhead
Idle
Executing

Matrix dimension : StarPU workers

10000 20000 30000 40000
Idle time

Runtime distribution / Kebnekaise / 35% selected

Portion of execution time

Overhead
Idle
Executing

Matrix dimension : StarPU workers

10000 20000 30000 40000

Idle time
Idle time

Runtime distribution / Kebnekaise / 50% selected

Portion of execution time

Matrix dimension : StarPU workers

Overhead
Idle
Executing
The end is near!

Thank you for your attention
Additional material and figures
Strong scalability, MPI

Graph showing strong scalability with MPI, using Kebnekaise, and selecting 5%.

Axes:
- Y-axis: Efficiency
- X-axis: CPU cores / MPI ranks

Lines for different values of N:
- N = 10000
- N = 20000
- N = 30000
- N = 40000
Strong scalability, MPI

The figure shows the strong scalability of MPI on the Kebnekaise system for different datasets sizes (N = 10000, 20000, 30000, 40000). The efficiency is plotted against the number of CPU cores/mpi ranks.

- For N = 10000, the efficiency decreases sharply with increasing CPU cores/mpi ranks.
- For N = 20000, the efficiency decreases at a slower rate compared to N = 10000.
- For N = 30000 and 40000, the efficiency decreases gradually, indicating good scalability.

The graph suggests that the system scales well up to a certain point, beyond which the efficiency drops, possibly due to resource limitations or communication overhead.
Strong scalability, MPI

![Graph showing strong scalability with MPI on Kebnekaise with 35% selected.](image-url)
Strong scalability, MPI

Strong scalability / MPI / Kebnekaise / 50% selected

Efficiency

CPU cores / MPI ranks

N = 10000
N = 20000
N = 30000
N = 40000

Efficiency

CPU cores / MPI ranks
Additional figures
Multiwindow DAG