Introduction to StarNEig
A Task-based Library for Solving
Nonsymmetric Eigenvalue Problems

Mirko Myllykoski and Carl Christian Kjelgaard Mikkelsen

in collaboration with
Angelika Schwarz, Lars Karlsson, Bo Kågström and
Mahmoud Eljammaly

Department of Computing Science
Umeå University

PPAM 2019
Eigenvalue problem

- StarNEig library aims to implement a complete stack of algorithms for solving **dense nonsymmetric** eigenvalue problems.
- Both standard
  \[ Ax_i = \lambda_i x_i \]
  and generalized
  \[ Ax_i = \lambda_i Bx_i \]
eigenvalue problems are considered.
Eigenvalue problem (algorithm stack)

Figure: An illustration of the complete algorithm stack in standard case.
Motivation (eigenvalue reordering)

- In some cases, we want to reorder the Schur form $S$ such that a **selected cluster of eigenvalues** appears in the leading diagonal blocks of the updated Schur form $\tilde{S}$.
- Gives an orthonormal basis for a desired invariant subspace.

**Figure:** An illustration of the reordering process in standard case.
Motivation (accumulated transformations)

- A modern algorithm
  - groups a set of orthogonal transformations together and
  - initially applies them only within a small diagonal window.
- The transformations are accumulated and later propagated with level 3 BLAS operations.

Figure: An illustration of accumulated transformations.
Motivation (concurrent windows)

- Multiple diagonal windows can be active **concurrently**.
- The level 3 BLAS updates **must be propagated in a sequentially consistent order**.
  - Requires careful coordination!

**Figure:** An illustration of two concurrent windows.
Motivation (ScaLAPACK-style approach)

- Eigenvalue reordering is implemented in ScaLAPACK\(^1\).
- With \( p \) cores, we can have up to \( \sqrt{p} \) concurrent windows.
- The transformation are broadcasted and applied in parallel.
  - Theoretically possible degree of parallelism is \( p \).
  - Only if we have \( \sqrt{p} \) concurrent windows.

Figure: An illustration of a ScaLAPACK-style algorithm.

---

Motivation (task-based approach and task graphs)

- Computational work is cut into self-contained tasks.
- A runtime system
  - derives the task dependences and
  - schedules the tasks to computational resources.
- The task dependencies can be visualized as a task graph.

Figure: A simplified task graph arising from eigenvalue reordering.
Motivation (more opportunities for concurrency)

- Real live task graphs are much more complex.
  - But enclose more opportunities for increased concurrency.
- The runtime system unrolls the task graph.
  - **No global synchronization.**
  - Computational steps are allowed overlap and merge.

---

**Figure:** An illustration of a task-based algorithm\(^2\).

Motivation (GPUs, distributed memory, other benefits)

- Other benefits of the task-based approach include
  - better load balancing,
  - task priorities,
  - accelerators support (GPUs) with performance models,
  - automatic data transfers between memory spaces and
  - implicit MPI communications.

Figure: An illustration of implicit MPI communications.
StarNEig library (overview)

- Designed and implemented at Umeå University as a part of NLAFET project.
- Runs on top of the **StarPU** task-based runtime system.
- Targets both
  - **shared memory** and
  - **distributed memory** machines.
- Some components of the library support **GPUs**.
- Real arithmetic supported, complex arithmetic planned.
- Beta release (v0.1-beta.2) available at https://github.com/NLAFET/StarNEig.
## StarNEig library (current status)

<table>
<thead>
<tr>
<th></th>
<th>Standard case</th>
<th>Generalized case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SM</td>
<td>DM</td>
</tr>
<tr>
<td>Hessenberg</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Schur</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reordering</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Eigenvectors</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

- **✓** Ready
- **✓** Experimental
- **✓** LAPACK or ScaLAPACK wrapper
- ✗ In progress
- — Not planned
Distributed memory (data distribution)

- StarNEig distributes matrices in **rectangular blocks of a uniform size**.

- User has three options:
  1. Use the default data distribution.
  2. Use a **two-dimensional block cyclic distribution**.
  3. Define a **data distribution function** $d : \mathbb{Z}^+ \times \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ that maps the block indices to the MPI rank space.

![Figure: Examples of various data distributions.](image-url)
Distributed memory (block size)

- StarNEig divides the distributed blocks into **square tiles**.
- Tile size is closely connected to task granularity.
  - Tiny tile size
    ⇒ fine-grained task granularity
    ⇒ large scheduling overhead.
- **Distributed blocks should be relatively large.**
- Many ScaLAPACK-style codes are designed for / perform optimally with smaller block sizes.

**Figure:** An illustration of how the block are divided into tiles.
Distributed memory (CPU core mapping)

- StarPU manages a set of worker threads.
  - Usually one thread per CPU core / GPU + MPI thread.
- **One process per node** (1ppn) configuration required.
  - A node can be, e.g., a full compute node or a NUMA island.
  - Many ScaLAPACK-style codes are designed for / perform optimally in *one process per core* (1ppc) configuration.

![Illustrations of CPU core mappings and data distributions.](image)

**Figure:** Illustrations of CPU core mappings and data distributions.
Distributed memory (ScaLAPACK compatibility)

> StarNEig is compatible with ScaLAPACK and provides a ScaLAPACK compatibility layer:

```c
// create a 2D block cyclic data distribution (pm X pn process mesh)
starneig_distr_t distr =
    starneig_distr_init_mesh(pm, pn, STARNEIG_ORDER_DEFAULT);

// create a n X n distributed matrix (bn X bn blocks)
starneig_distr_matrix_t dA =
    starneig_distr_matrix_create(n, n, bn, bn, STARNEIG_REAL_DOUBLE, distr);

// convert the data distribution to a BLACS context
starneig_blacs_context_t context = starneig_distr_to_blacs_context(distr);

// convert the distributed matrix to a BLACS descriptor and a local buffer
starneig_blacs_descr_t descr_a;
double *local_a;
starneig_distr_matrix_to_blacs_descr(dA, context, & descr_a, (void **) & local_a);

// ScaLAPACK subroutine for reducing general distributed matrix to upper Hessenberg form
extern void pdgehrd_(int const *, int const *, int const *, double *,
    int const *, int const *, starneig_blacs_descr_t const *, double *,
    double *, int const *, int *);
pdgehrd_(&n, &ilo, &ihi, local_a, &ia, &ja, &descr_a, tau, ...);
```
Computational experiments were performed on the Kebnekaise system, HPC2N, Umeå University.

- **Regular compute node:** 28 Intel Xeon E5-2690v4 Broadwell cores. 128 GB memory. FDR Infiniband.
- **V100 GPU node:** 28 Intel Xeon Gold 6132 Skylake cores. 192 GB memory. **Two NVIDIA Tesla V100 GPUs.**

The results are extracted from

Schur reduction (distributed memory performance)

(a) Standard case\(^3\).

(b) Generalized case\(^4\).

Figure: StarNEig versus ScaLAPACK-style approach (relative run-time improvement).

\(^3\) https://github.com/NLAFET/SEVP-PDHSEQR-Alg953/.
\(^4\) https://github.com/NLAFET/GEVP-PDHGEQZ.
Schur reduction (distributed memory scalability)

Figure: Standard case, 28 cores / node, max 700 cores.
Eigenvalue reordering (distributed memory performance)

(a) Standard case\(^5\).

(b) Generalized case\(^6\).

Figure: StarNEig versus ScaLAPACK-style approach, 35% selected.

\(^5\) [Link to ScaLAPACK-PDTRSEN](http://www.netlib.org/scalapack/explore-html/d8/db0/pdtrsen_8f.html).

\(^6\) [Link to NLAFET GEVP-PDHGEQZ](https://github.com/NLAFET/GEVP-PDHGEQZ).
Eigenvalue reordering (distributed memory scalability)

Figure: Standard case, 35% selected, 28 cores / node, max 700 cores.
Figure: Standard case, 35% selected, NVIDIA V100.
Summary

- Task-based algorithms for most steps in the algorithm chain:

<table>
<thead>
<tr>
<th></th>
<th>Standard case</th>
<th>Generalized case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SM</td>
<td>DM</td>
</tr>
<tr>
<td>Hessenberg</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Schur</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reordering</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Eigenvectors</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Support for shared and distributed memory; and GPUs.
- Increased parallelism through expressing algorithms as DAGs.
- Better (heterogeneous) scheduling and load balancing.
- Overlapping communications and computations.

- Parallel, efficient and robust algorithm for computing eigenvectors.
  - See preceding presentation from Carl Christian (Parallel Robust Computation of Generalized Eigenvectors of Matrix Pencils).
Extra (Hessenberg reduction, GPU performance)

Figure: StarNEig versus MAGMA, NVIDIA V100.