Parallel Robust Computation of Generalized Eigenvectors of Matrix Pencils

Carl Christian Kjelgaard Mikkelsen[0000−0002−9158−1941] and Mirko Myllykoski[0000−0002−3689−0899]
Department of Computing Science and HPC2N
Umeå University
90187 Umeå, Sweden
{spock,mirkom}@cs.umu.se

Abstract. In this paper we consider the problem of computing generalized eigenvectors of a matrix pencil in real Schur form. In exact arithmetic, this problem can be solved using substitution. In practice, substitution is vulnerable to floating-point overflow. The robust solvers xtgevc in LAPACK prevent overflow by dynamically scaling the eigenvectors. These subroutines are scalar and sequential codes which compute the eigenvectors one by one. In this paper, we discuss how to derive robust algorithms which are blocked and parallel. The new StarNEig library contains a robust task-parallel solver Zazamoukh which runs on top of StarPU. Our numerical experiments show that Zazamoukh achieves a super-linear speedup compared with dtgevc for sufficiently large matrices.

Keywords: Generalized eigenvectors · Overflow protection · Task-parallelism

1 Introduction

Let \( A \in \mathbb{R}^{m \times m} \) and \( B \in \mathbb{R}^{m \times m} \). The matrix pencil \((A, B)\) consists of all matrices of the form \( A - \lambda B \) where \( \lambda \in \mathbb{C} \). The set of (generalized) eigenvalues of the matrix pencil \((A, B)\) is given by

\[
\lambda(A, B) = \{ \lambda \in \mathbb{C} : \det(A - \lambda B) = 0 \}.
\]

We say that \( x \in \mathbb{C}^m \) is a (generalized) eigenvector of the matrix pencil \((A, B)\) if and only if \( x \neq 0 \) and

\[
Ax = \lambda Bx.
\]

The eigenvalues of \((A, B)\) can be computed by first reducing \((A, B)\) to real Schur form \((S, T)\). Specifically, there exist orthogonal matrices \( Q \) and \( Z \) such that \( S = Q^T A Z \) is quasi-upper triangular and \( T = Q^T B Z \) is upper triangular. It is clear that

\[
\lambda(A, B) = \lambda(S, T).
\]

Moreover, \( y \in \mathbb{C}^m \) is a generalized eigenvector of \((S, T)\) corresponding to the eigenvalue \( \lambda \), if and only if \( x = Z y \) is a generalized eigenvector of \((A, B)\) corresponding to the eigenvalue \( \lambda \).
In this paper, we consider the parallel computation of eigenvectors of a matrix pencil in real Schur form. In exact arithmetic, this problem can be solved using substitution. However, substitution is very vulnerable to floating-point overflow.

In LAPACK [3] there exists a family \texttt{xtgevc} of subroutines which compute the generalized eigenvectors of a matrix pencil in Schur form. They prevent overflow by dynamically scaling the eigenvectors. These subroutines are scalar codes which compute the eigenvectors one by one. In this paper we discuss the construction of algorithms which are not only robust, but blocked and parallel.

Our paper is organized as follows. In Section 2 we briefly review past work on robust algorithms for solving equations of triangular type. In Section 3 we consider the problem of computing the eigenvectors of a matrix pencil in real Schur form using real arithmetic. This problem is equivalent to solving a homogeneous matrix equation of the form

\[ \text{SVD} - T \text{VE} = 0, \quad (1) \]

which respect to \( V \). The matrix \( D \) is diagonal and the matrix \( E \) is block diagonal with diagonal blocks which are either 1-by-1 or 2-by-2. In Section 4 we present a blocked algorithm for solving this matrix equation. In Section 5 we discuss how to prevent overflow in this algorithm. The concept of an augmented matrix is central to this discussion. A robust task-parallel solver \texttt{Zazamoukh} has been developed and integrated into the new StarNEig library for solving non-symmetric eigenvalue problems [1, 9]. The performance of \texttt{Zazamoukh} is compared to LAPACK in Section 7. We suggest directions for future work in Section 8.

2 Related Work on Robust Algorithms

LAPACK contains several robust routines for solving equations involving triangular matrices. These routines include \texttt{xtrevc} (standard eigenvectors), \texttt{xtgevc} (generalized eigenvectors), and \texttt{xtrsyl} (Sylvester matrix equations). They prevent overflow by scaling the right hand-side dynamically. The underlying principles were originally derived by Anderson and implemented in \texttt{xlatrs} [2]. This family of subroutines apply to triangular linear systems

\[ T x = b \quad (2) \]

with a single right-hand side. Mikkelsen and Karlsson [7] formalized the work of Anderson and derived a robust blocked algorithm for solving equation (2). In particular, Mikkelsen and Karlsson isolated two functions \texttt{ProtectDivision} and \texttt{ProtectUpdate} which can be used to prevent overflow in scalar divisions \( y \leftarrow b/t \) and general linear updates of the form \( Y \leftarrow Y - TX \). These two functions have the following key properties:

1. If \( t \neq 0 \) and \( |b| \leq \Omega \), and

\[ \xi = \text{ProtectDivision}(|b|,|t|) \]


then $\xi \in (0, 1]$ and $|\xi b| \leq |t|\Omega$. It follows that the scaled division

$$y \leftarrow \frac{(\xi b)}{t}$$

cannot exceed $\Omega$.

2. If $Z = Y - TX$ is defined, with

$$\|Y\|_\infty \leq \Omega, \quad \|T\|_\infty \leq \Omega, \quad \|X\|_\infty \leq \Omega,$$

and

$$\xi = \text{ProtectUpdate}(\|Y\|_\infty, \|T\|_\infty, \|X\|_\infty)$$

then $\xi \in (0, 1]$ and

$$\xi(\|Y\|_\infty + \|T\|_\infty \|X\|_\infty) \leq \Omega.$$  

It follows that

$$Z \leftarrow (\xi Y) - T(\xi X) = (\xi Y) - (\xi T)X.$$  

can be computed without any component of any intermediate or final result exceeding $\Omega$.

Mikkelsen, Schwarz, and Karlsson have derived a robust blocked algorithm for solving triangular linear systems

$$TX = B \tag{3}$$

with multiple right-hand sides. Their task-parallel implementation (Kiya) is significantly faster than dlatrs when numerical scaling is necessary and not significantly slower than dtrsm when numerical scaling is not required [8]. This paper also contains a formal proof of the correctness of ProtectUpdate and ProtectDivision.

3 Real Arithmetic

In this section we show that the problem of computing generalized eigenvectors is equivalent to solving a real homogeneous matrix equation of the type given by equation (1).

Let $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{m \times m}$ be given. The set of generalized eigenvalues of the matrix pencil $(A, B)$ can be computed by first reducing $(A, B)$ to generalized real Schur form. Specifically, there exist orthogonal matrices $Q \in \mathbb{R}^{m \times m}$ and $Z \in \mathbb{R}^{m \times m}$ such that

$$S = Q^T AZ = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1p} \\ S_{21} & S_{22} & \cdots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ S_{p1} & S_{p2} & \cdots & S_{pp} \end{bmatrix}, \quad T = Q^T BZ = \begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1p} \\ T_{21} & T_{22} & \cdots & T_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ T_{p1} & T_{p2} & \cdots & T_{pp} \end{bmatrix}$$
are upper block-triangular and \( \dim(S_{jj}) = \dim(T_{jj}) \in \{1, 2\} \). It is clear that
\[
\lambda(S, T) = \bigcup_{j=1}^{p} \lambda(S_{jj}, T_{jj}).
\]
We follow the standard convention and represent eigenvalues \( \lambda \) using an ordered pair \((\alpha, \beta)\) where \( \alpha \in \mathbb{C} \) and \( \beta \geq 0 \). If \( \beta > 0 \), then \( \lambda = \alpha/\beta \) is a finite eigenvalue.

The case of \( \alpha \in \mathbb{R} \setminus \{0\} \) and \( \beta = 0 \), corresponds to an infinite eigenvalue. The case of \( \alpha = \beta = 0 \) corresponds to an indefinite eigenvalue problem.

Let \( n_j = \dim(S_{jj}) = \dim(T_{jj}) \). In order to simplify the current discussion, we will make the following assumptions:

1. If \( n_j = 1 \), then \((S_{jj}, T_{jj})\) has an eigenvalue with is either real or infinite.
2. If \( n_j = 2 \), then \((S_{jj}, T_{jj})\) has two complex conjugate eigenvalues.
3. All eigenvalues are distinct.

By eliminating the possibility of multiple eigenvalues and indefinite problems we are free to formulate robust algorithms for well-defined problems. The question of how to handle problems which are not well-defined is certainly important but outside the scope of this paper.

### 3.1 Computing a Single Eigenvector

It this subsection, we note that the problem of computing a single generalized eigenvector of \((S, T)\) is equivalent to solving a tall homogeneous matrix equation involving real matrices. Let \( \lambda \in \lambda(S_{jj}, T_{jj}) \) and let \( \lambda = (a_j + ib_j)/\beta_j \) where \( \beta_j > 0 \) and
\[
\alpha_j = a_j + ib_j \in \mathbb{C}.
\]

Let \( n_j = \dim(S_{jj}) \) and let \( D_{jj} \in \mathbb{R}^{n_j \times n_j} \) and \( E_{jj} \in \mathbb{R}^{n_j \times n_j} \) be given by
\[
D_{jj} = \begin{bmatrix} \beta_j \\ 0 \end{bmatrix}, \quad E_{jj} = \begin{bmatrix} a_j \\ b_j \end{bmatrix},
\]
when \( n_j = 1 \) (or equivalently \( b_j = 0 \)) and
\[
D_{jj} = \begin{bmatrix} \beta_j & 0 \\ 0 & \beta_j \end{bmatrix}, \quad E_{jj} = \begin{bmatrix} a_j & b_j \\ -b_j & a_j \end{bmatrix},
\]
when \( n_j = 2 \) (or equivalently \( b_j \neq 0 \)). With this notation, the problem of computing an eigenvector is equivalent to solving the homogeneous linear equation
\[
SVD_{jj} - TVE_{jj} = 0
\]
with respect to \( V \in \mathbb{R}^{m \times n_j} \). This follows from the following lemma.

**Lemma 1.** Let \( \lambda \in \lambda(S_{jj}, T_{jj}) \) and let \( \lambda = (a_j + ib_j)/\beta \) where \( \beta > 0 \). Then the following statements are true:

1. If \( n_j = 1 \), then \( x \in \mathbb{R}^m \) is a real eigenvector of \((S, T)\) corresponding to the real eigenvalue \( \lambda \in \mathbb{R} \) if and only if \( V = [x] \) has rank 1 and solves (6).
2. If \( n_j = 2 \), then \( z = x + iy \in \mathbb{C}^m \) is a complex eigenvector of \((S, T)\) corresponding to the complex eigenvalue \( \lambda \in \mathbb{C} \) if and only if \( V = [x \ y] \) has rank 2 and solves (6).
3.2 Computing All Eigenvectors

In this subsection, we note that the problem of computing all generalized eigenvectors of \((S, T)\) is equivalent to solving a homogeneous matrix equation involving real matrices. Specifically, let \(p\) denote the number of 1-by-1 or 2-by-2 blocks on the diagonal of \(S\), let \(D \in \mathbb{R}^{m \times m}\) and \(E \in \mathbb{R}^{m \times m}\) be given by

\[
D = \text{diag}\{D_{11}, D_{22}, \ldots, D_{pp}\}, \quad B = \text{diag}\{E_{11}, E_{22}, \ldots, E_{pp}\},
\]

where \(D_{jj}\) and \(E_{jj}\) are given by equations (4) and (5). Then \(V = [V_1 \ V_2 \ \ldots \ V_p] \in \mathbb{R}^{m \times m}\) solves the homogeneous matrix equation

\[
SVD - TVE = 0, \tag{7}
\]

if and only if \(V_j \in \mathbb{R}^{m \times n_j}\) solves equation (6).

4 A Blocked Algorithm

In this section we present a blocked algorithm for solving the homogeneous matrix equation (7). We begin by redefining the partitioning of \(S\). Let \(S = [S_{ij}], \quad i, j \in \{1, 2, \ldots, M\}\) denote any partitioning of \(S\) into an \(M\) by \(M\) block matrix which does not split any of the 2-by-2 blocks along the diagonal of \(S\). Apply the same partitioning to \(T, D, B,\) and \(V\). The homogeneous matrix equation (7) can now be written as

\[
\begin{bmatrix}
S_{11} & S_{12} & \cdots & S_{1M} \\
S_{21} & S_{22} & \cdots & S_{2M} \\
\vdots & \vdots & \ddots & \vdots \\
S_{M1} & S_{M2} & \cdots & S_{MM}
\end{bmatrix}
\begin{bmatrix}
V_{11} & V_{12} & \cdots & V_{1M} \\
V_{21} & V_{22} & \cdots & V_{2M} \\
\vdots & \vdots & \ddots & \vdots \\
V_{M1} & V_{M2} & \cdots & V_{MM}
\end{bmatrix}
\begin{bmatrix}
D_{11} \\
D_{22} \\
\vdots \\
D_{MM}
\end{bmatrix}
= \begin{bmatrix}
T_{11} & T_{12} & \cdots & T_{1M} \\
T_{21} & T_{22} & \cdots & T_{2M} \\
\vdots & \vdots & \ddots & \vdots \\
T_{M1} & T_{M2} & \cdots & T_{MM}
\end{bmatrix}
\begin{bmatrix}
V_{11} & V_{12} & \cdots & V_{1M} \\
V_{21} & V_{22} & \cdots & V_{2M} \\
\vdots & \vdots & \ddots & \vdots \\
V_{M1} & V_{M2} & \cdots & V_{MM}
\end{bmatrix}
\begin{bmatrix}
E_{11} \\
E_{22} \\
\vdots \\
E_{MM}
\end{bmatrix} \tag{8}
\]

The block columns of \(V\) can be computed concurrently, because \(D\) and \(E\) are block diagonal. It is straightforward to verify that equation (8) can be solved using Algorithm 1. Algorithm 1 can be implemented using three distinct kernels.

1. Kernel 1 solves small homogeneous matrix equations of the form

\[
S_{jj}XD_{jj} = T_{jj}XE_{jj} \tag{9}
\]

with respect to \(X\). This is equivalent to finding eigenvectors for the pencil \((S_{jj}, T_{jj})\).
Algorithm 1: Blocked computation of all generalized eigenvectors

1. for \( j \leftarrow 1, 2, \ldots, M \) do
   2. for \( i \leftarrow 1, 2, \ldots, j \) do
       3. \( V_{ij} \leftarrow 0 \);
   4. for \( j \leftarrow 1, 2, \ldots, M \) do
       5. Solve \( S_{jj} XD_{jj} = T_{jj}XE_{jj} \) with respect to \( X \) and set \( V_{jj} \leftarrow X \);
       6. for \( i \leftarrow j - 1, \ldots, 1 \) do
           7. for \( k \leftarrow 1, 2, \ldots, i \) do
               8. Perform the linear update \( V_{kj} \leftarrow V_{kj} - (S_{k,i+1}V_{i+1,j}D_{jj} - T_{k,i+1}V_{i+1,j}E_{jj}) \)
           9. Solve \( S_{ii} XD_{jj} - T_{ii}XE_{jj} = V_{ij} \) with respect to \( X \) and set \( V_{ij} \leftarrow X \);

2. Kernel 2 performs specialized linear updates of the form
   \[ Y \leftarrow Y - (S_{k,i+1}XD_{jj} - T_{k,i+1}XE_{jj}) \] (10)

3. Kernel 3 solves small matrix equations of the form
   \[ S_{ii} XD_{jj} - T_{ii}XE_{jj} = Y \] (11)
   with respect to \( X \).

Once these kernels have been implemented, it is straightforward to parallelize Algorithm 1 using a task-based runtime system such as StarPU [4].

5 Constructing Robust Kernels

Algorithm 1 is not robust. Each of the three kernels are vulnerable to floating point overflow. The kernels needed for Algorithm 1 can be implemented using nested loops, divisions and linear updates. Therefore, it is not surprising that robust kernels can be implemented using the functions \texttt{ProtectDivision} and \texttt{ProtectUpdate} given in Section 2. We will now explain how this can be done without sacrificing the potential for level-3 BLAS operations. We will concentrate on Kernel 2 which executes the vast majority of the arithmetic operations needed for Algorithm 1.

We will use the concept of an \textit{augmented} matrix introduced by Mikkelsen, Schwarz and Karlsson [8].
Definition 1. Let $X \in \mathbb{R}^{m \times n}$ be partitioned into $k$ block columns

$$X = \begin{bmatrix} X_1 & X_2 & \ldots & X_k \end{bmatrix}, \quad X_j \in \mathbb{R}^{m \times n_j},$$

and let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k) \in \mathbb{R}^k$ satisfy $\alpha_j \in (0, 1]$. The augmented matrix $\langle \alpha, X \rangle$ represents the real matrix $Y \in \mathbb{R}^{m \times n}$ given by

$$Y = \begin{bmatrix} Y_1 & Y_2 & \ldots & Y_k \end{bmatrix}, \quad Y_j \in \mathbb{R}^{m \times n_j}, \quad Y_j = \alpha_j^{-1}X_j.$$ 

This is a trivial extension of the original definition which considered the case of $k = n$. The purpose of the scaling factors $\alpha_j$ is to extend the normal representational range of our floating-point numbers.

Now consider the problem of computing without overflow a specialized linear update of the form needed for Kernel 2, i.e., an update of the form

$$Y \leftarrow (Y - S(XD)) + T(XE).$$

(12)

where $S$ and $T$ are general dense matrices, $D$ is diagonal and $E$ is block diagonal with diagonal blocks that are 1-by-1 or 2-by-2. The parentheses are used indicated the order of evaluation. A representation of the matrices $Z_1 = XD$ and $Z_2 =XE$ can be obtained using augmented matrices and Algorithm 2. Similarly, it is possible to obtain a representation of the matrices $Z_3 = Y - SZ_1$ and $Y = Z_3 + TZ_2$ using augmented matrices and Algorithm 3.

**Algorithm 2: Right updates with block diagonal matrix**

**Data:** An augmented matrix $\langle \alpha, X \rangle$ where

$$X = \begin{bmatrix} X_1 & X_2 & \ldots & X_k \end{bmatrix}, \quad X_j \in \mathbb{R}^{m \times n_j}, \quad \|X_j\|_\infty \leq \Omega,$$

and a block diagonal matrix matrix

$$F = \text{diag}(F_1, F_2, \ldots, F_k), \quad F_j \in \mathbb{R}^{n_j \times n_j}, \quad \|F_j\|_\infty \leq \Omega.$$

**Result:** An augmented matrix $\langle \beta, Y \rangle$ where

$$Y = \begin{bmatrix} Y_1 & Y_2 & \ldots & Y_k \end{bmatrix}, \quad Y_j \in \mathbb{R}^{m \times n_j}, \quad \|Y_j\|_\infty \leq \Omega,$$

such that

$$\beta_j^{-1}Y_j = (\alpha_j^{-1}X_j)F_j,$$

and $Y$ can be computed without exceeding $\Omega$.

1. for $j = 1, 2, \ldots, k$
2. $\gamma_j = \text{ProtectUpdate}(0, \|X_j\|_\infty, \|F_j\|_\infty)$;
3. $Y_j = (\gamma_j X_j)F_j$;
4. $\beta_j = \alpha_j \gamma_j$;
5. return $\langle \beta, Y \rangle$;
Algorithm 3: Left update with dense matrix

Data: A dense matrix $T \in \mathbb{R}^{m \times l}$ and augmented matrices $\langle \alpha, X \rangle$ and $\langle \beta, Y \rangle$

where

\[
X = \begin{bmatrix} X_1 & X_2 & \ldots & X_k \end{bmatrix}, \quad X_j \in \mathbb{R}^{l \times n_j}, \quad \|X_j\|_\infty \leq \Omega,
\]

\[
Y = \begin{bmatrix} Y_1 & Y_2 & \ldots & Y_k \end{bmatrix}, \quad Y_j \in \mathbb{R}^{m \times n_j}, \quad \|Y_j\|_\infty \leq \Omega.
\]

Result: An augmented matrix $\langle \zeta, Z \rangle$ where

\[
Z = \begin{bmatrix} Z_1 & Z_2 & \ldots & Z_k \end{bmatrix}, \quad Z_j \in \mathbb{R}^{m \times n_j}, \quad \|Z_j\|_\infty \leq \Omega,
\]

such that

\[
\zeta_j^{-1} Z_j = \beta_j^{-1} Y_j - T(\alpha_j^{-1} X_j),
\]

and $Z$ can be computed without exceeding $\Omega$.

1. for $j = 1, \ldots, k$ do
2. \hspace{1em} $\gamma_j = \min\{\alpha_j, \beta_j\}$;
3. \hspace{1em} $\delta_j = \text{ProtectUpdate}(\gamma_j/\beta_j, \|Y_j\|_\infty, \|T\|_\infty, (\gamma_j/\alpha_j)\|X_j\|_\infty)$;
4. \hspace{1em} $X_j \leftarrow \delta_j(\gamma_j/\alpha_j) X_j$;
5. \hspace{1em} $Y_j \leftarrow \delta_j(\gamma_j/\beta_j) Y_j$;
6. \hspace{1em} $\zeta_j = \gamma_j \delta_j$;
7. $Z \leftarrow Y - TX$;
8. return $\langle \zeta, Z \rangle$

We cannot escape the fact that the right updates, i.e., the calls to Algorithm 2 have low arithmetic intensity because we are essentially scaling the columns of the input matrix. However, each right update can be followed by a left update, i.e., a call to Algorithm 3 acting on the same data. Algorithm 3 consists of some light pre-processing (the for-loop spanning lines 1-6) which has low arithmetic intensity, but concludes with a regular level-3 BLAS operation (line 7) which contains the overwhelming majority of the necessary arithmetic operations. This explains why robustness can be combined with good performance.

In order to execute all linear updates needed for a robust variant of Algorithm 1 we require norms of certain submatrices of $S$ and $T$. In particular, we need the infinity norms of all super-diagonal blocks of $S$ and $T$. Moreover, we require the infinity norm of certain submatrices of $V$. These submatrices consists of either a single column (segment of real eigenvector) or two adjacent columns (segment of complex eigenvector). The infinity norm must be computed whenever a submatrix has been initialized or updated. \text{ProtectUpdate} requires that the input arguments are bounded by $\Omega$ and failure is possible if they are not.

6 Zazamoukh - a Task-Parallel Robust Solver

The new StarNEig library runs on top of StarPU and can be used to solve dense non-symmetric eigenvalue problems [1, 9]. A robust variant of Algorithm
1 has been implemented in StarNEig. This implementation (Zazamoukh) uses augmented matrices and scaling factors which are integer powers of 2. Zazamoukh can compute eigenvectors corresponding to a subset of $\lambda(S, T)$ which is closed under complex conjugation. Zazamoukh is not subject to the assumptions made in Section 3. In particular, Zazamoukh can handle the case of multiple eigenvalues. Zazamoukh is currently limited to shared memory, but an extension to distributed memory is under development.

6.1 Memory Layout

Given block sizes $mb$ and $nb$ Zazamoukh partitions $S$, $T$ and $V$ conformally by rows and columns. In the absence of any 2-by-2 diagonal blocks on the diagonal of $S$ the tiles of $S$ and $T$ are $mb$ by $mb$ and the tiles of $V$ are $mb$ by $nb$. The only exceptions can be found along the right and lower boundaries of the matrices. This default configuration is adjusted minimally to prevent splitting any 2-by-2 block of $S$ or separating the real part and the imaginary part of a complex eigenvector into separate tile columns.

6.2 Tasks

Zazamoukh relies on four types of tasks:

1. Pre-processing tasks which compute all quantities needed for robustness. This includes the infinity norm of all super-diagonal tiles of $S$ and $T$ as well as all norms needed for the robust solution of equations of the type (11). If necessary, the matrices $S$ and $T$ are scaled minimally.
2. Robust solve tasks which use dtgevc to compute the lower tips of eigenvectors, i.e., equation (9)) and a robust solver based on dlaln2 to solve equations of the type given by equation (11).
3. Robust update tasks which execute updates of the type given by (10).
4. Post-processing tasks which enforce a consistent scaling on all segments of all eigenvectors.

6.3 Task insertion order and priorities

Zazamoukh is closely related to Kiya which solves triangular linear systems with multiple right-hand sides. Apart from the pre-processing and post-processing tasks, the main task graph is the disjoint union of $p$ task-graphs, one for each block column of the matrix of eigenvectors. Zazamoukh uses the same task insertion order and priorities as Kiya to process each of the $p$ disjoint sub-graphs. Specifically, tasks corresponding to blocks of $(S, T)$ on the main block diagonal are assigned the highest possible priority. Tasks corresponding to blocks of $(S, T)$ on the $j$th superdiagonal are assigned priority $q - j$ until the number of distinct priorities are exhausted. The rationale behind this choice is to guide the scheduler towards rapid progress on the critical path.
7 Numerical Experiments

In this section we give the result of a set of experiments involving tiny \(m \leq 10,000\) and small \(m \leq 40,000\) matrices. Each experiment consisted of computing all eigenvectors of the matrix pencil. The run time was measured for \texttt{dtgevc} from LAPACK and \texttt{Zazamoukh}. Results related to somewhat larger matrices \((m \leq 80,000)\) can be found in the NLAFET Deliverable 2.7 [10].

The experiments were executed on an Intel Xeon E5-2690v4 (Broadwell) node with 28 cores arranged in two NUMA islands with 14 cores each. StarNEig was compiled with OpenBLAS 0.3.2 (includes LAPACK) and StarPU 1.2.8. We used the StarNEig test-program \texttt{starneig-test} to generate reproducible experiments. The default parameters produce matrix pencils where approximately 1 percent of all eigenvalues are zeros, 1 percent of all eigenvalues are infinities and there are no indefinite eigenvalues. \texttt{Zazamoukh} used the default tile size \(mb = nb\) which is 1.6 percent of the matrix dimension for matrix pencils with dimension \(m \geq 1000\).

All experiments were executed with exclusive access to a complete node (28 cores). LAPACK was run in sequential mode, while \texttt{Zazamoukh} used 28 StarPU workers and 1 master thread. The summary of our results are given in Figure 1. The timings include all overhead needed to achieve robustness. The speedup of \texttt{Zazamoukh} over LAPACK is initially very modest as there is not enough tasks to keep 28 workers busy, but it picks up rapidly and \texttt{Zazamoukh} achieves a super-linear speedup over \texttt{dtgevc} when \(m \geq 10,000\). This is an expression of the fact that \texttt{Zazamoukh} uses a blocked algorithm, whereas \texttt{dtgevc} computes the eigenvectors one by one.

8 Conclusion

Previous work by Mikkelsen, Schwarz and Karlsson has shown that triangular linear systems can be solved without overflow and in a blocked and parallel manner using augmented matrices. In this paper we have shown that the eigenvectors of a matrix pencil can be computed without overflow and in a blocked and parallel manner using augmented matrices. Certainly, robust algorithms are slower than non-robust algorithms when numerical scaling is not required, but robust algorithms will always return a result which can be evaluated in the context of the user’s application. To the best of our knowledge StarNEig is the only library which contains a parallel robust solver for computing the generalized eigenvectors of a dense non-symmetric matrix pencil. The StarNEig solver (\texttt{Zazamoukh}) runs on top of StarPU and uses augmented matrices and scaling factors with are integer powers of 2 to prevent overflow. It achieves super-linear speedup compared with \texttt{dtgevc} from LAPACK (OpenBLAS 0.3.2). In the immediate future we expect to pursue the following work:

1. Extend \texttt{Zazamoukh} to also compute left eigenvectors. Here the layout of the loops is different and we must use the 1-norm instead of the infinity norm when executing the overflow protection logic.
Table 1. Comparison between sequential dtgevc and task-parallel Zazamoukh using 28 cores. The run-times are given in milli-seconds (ms). The last column gives the speedup of Zazamoukh over LAPACK. Values above 28 correspond to super-linear speedup. All eigenvectors were computed with a relative residual less than $2u$, where $u$ denotes the double precision unit roundoff.

| dimension | eigenvalue analysis | run time (ms) | SpeedUp |
|-----------|---------------------|---------------|---------|
|           | m zeros inf. indef. | LAPACK StarNEig |         |
| 1000      | 11 13 0             | 295 175       | 1.6857  |
| 2000      | 25 16 0             | 1598 409      | 3.9071  |
| 3000      | 24 30 0             | 6182 929      | 6.6545  |
| 4000      | 42 49 0             | 15476 1796    | 8.6169  |
| 5000      | 54 37 0             | 30730 2113    | 14.5433 |
| 6000      | 61 64 0             | 53700 2637    | 20.3641 |
| 7000      | 67 64 0             | 84330 3541    | 23.8153 |
| 8000      | 56 69 0             | 122527 4769   | 25.6924 |
| 9000      | 91 91 0             | 171800 6189   | 27.5789 |
| 10000     | 108 94 0            | 242466 7821   | 31.0019 |
| 20000     | 175 197 0           | 2034664 49823 | 40.8378 |
| 30000     | 306 306 0           | 7183746 162747| 44.1406 |
| 40000     | 366 382 0           | 17713267 380856| 46.5091 |

2. Extend Zazamoukh to distributed memory machines.
3. Extend Zazamoukh’s solver to use recursive blocking to reduce the run-time.
   The solve tasks all lie on the critical path of the task graph.
4. Extend Zazamoukh to complex data types. This case is simpler than real arithmetic because there are no 2-by-2 blocks on the main diagonal of $S$.
5. Revisit the complex division routine dladiv [6] which is the foundation for the dlapln2 routine used by Zazamoukh’s solve tasks. In particular, the failure modes of xladiv have not been characterized [5].

Acknowledgments

This work is part of a project (NLAFET) that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 671633. This work was supported by the Swedish strategic research programme eSSENCE. We thank the High Performance Computing Center North (HPC2N) at Umeå University for providing computational resources and valuable support during test and performance runs.

References

1. StarNEig. https://nlafet.github.io/StarNEig/
2. Anderson, E.: LAPACK Working Note No. 36: Robust Triangular Solves for Use in Condition Estimation. Tech. Rep. CS-UT-CS-91-142, University of Tennessee, Knoxville, TN, USA (August 1991)

3. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users’ Guide. SIAM, Philadelphia, PA, 3rd edn. (1999). https://doi.org/10.1137/1.9780898719604

4. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. CCPE - Special Issue: Euro-Par 2009 23, 187–198 (2011). https://doi.org/10.1002/cpe.1631

5. Baudin, M.: Personal correspondence to C. C. Kjelgaard Mikkelsen

6. Baudin, M., Smith, R.L.: Robust Complex Division in Scilab. CoRR abs/1210.4539 (2012), http://arxiv.org/abs/1210.4539

7. Kjelgaard Mikkelsen, C.C., Karlsson, L.: Blocked Algorithms for Robust Solution of Triangular Linear Systems. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (eds.) Parallel Processing and Applied Mathematics. pp. 68–78. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-78024-5_7

8. Kjelgaard Mikkelsen, C.C., Schwarz, A.B., Karlsson, L.: Parallel Robust Solution of Triangular Linear Systems. CCPE (2018). https://doi.org/10.1002/cpe.5064

9. Myllykoski, M., Kjelgaard Mikkelsen, C.C.: Introduction to StarNEig — A Task-based Library for Solving Nonsymmetric Eigenvalue Problems. Accepted by PPAM-2019

10. Myllykoski, M., Kjelgaard Mikkelsen, C.C., Schwarz, A., Kågström, B.: D2.7 eigenvalue solvers for nonsymmetric problems. Tech. rep., Umeå University (2019), http://www.nlafet.eu/wp-content/uploads/2019/04/D2.7-EVP-solvers-evaluation-final.pdf