UMEÅ UNIVERSITY

Towards Efficient Overflow-Free Solvers for Systems of Triangular Type

Angelika Beatrix Schwarz

Licentiate Thesis, May 2019
Department of Computing Science
Umeå University
Sweden

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

angies@cs.umu.se

ISBN 978-91-7855-084-5
ISSN 0348-0542
UMINF 19.05

Printed by City Print i NorrAB, Umeå, 2019.

Abstract

Triangular linear systems are fundamental in numerical linear algebra. A triangular linear system has a straight-forward and efficient solution strategy, namely forward substitution for lower triangular systems and backward substitution for upper triangular systems. Triangular systems, or, more generally, systems of triangular type occur frequently in algorithms for more complex problems. This thesis addresses three systems that involve linear systems of triangular type.

The first system concerns quasi-triangular matrices. Quasi-triangular matrices are block triangular with 1-by-1 and 2-by-2 blocks on the diagonal. Quasi-triangular systems arise in the computation of eigenvectors from the real Schur form for the non-symmetric eigenvalue problem. This thesis contributes two algorithms for the eigenvector computation, which solve shifted quasi-triangular linear systems in an efficient and scalable way.

The second system addresses scaled triangular linear systems. During the solution of a triangular linear system, the entries of the solution can grow. This growth can exceed the representable range of floating-point numbers. Such an overflow can be avoided by solving a scaled triangular system. The solution is scaled prior to every operation that would otherwise result in an overflow. After scaling, the operations can be executed safely. This thesis analyzes the scalability of a recently developed tiled, robust solver for scaled triangular systems, which ensures that at no point in the computation the overflow threshold is exceeded.

The third system tackles the scaled continuous-time triangular Sylvester equation, which couples two quasi-triangular matrices. The solution process is prone to overflow. This thesis contributes a robust, tiled solver and demonstrates its practicability.

These three systems can be addressed with a variation of forward or backward substitution. Compared to the highly optimized and scalable implementations of standard forward and backward substitution available in HPC libraries, the existing implementations of these three systems run at a smaller fraction of the peak performance. This thesis presents techniques to improve on the performance and robustness of the implementations of the three systems.

Acknowledgements

This project would never have reached this stage without the support by numerous people. I thank everybody who provided invaluably constructive criticism, friendly advice or guidance during this project. Special thanks go to my supervisors Lars Karlsson, Bo Kågström and Carl Christian Kjelgaard Mikkelsen and my reference person Martin Berggren.

I thank the research group and my colleagues for their support, inspiring discussions and advice whenever needed. You created an amazing working environment. I am thankful for the quick and competent help by the staff at HPC2N.

You, my dear friends, provided support and advice or cheered me up. Special thanks go to Eddie Wadbro. Without all of you, my time here would have looked very differently. I will forever remember the funny moments with you.

I thank my family for their unconditional support and great encouragement throughout my studies. No journey is too long for you.

Financial support has been received from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671633, and by eSSENCE, a collaborative e-Science programme funded by the Swedish Government via the Swedish Research Council (VR). The computational experiments have been done using the resources at High Performance Computing Center North (HPC2N) at Umeå University, which is part of the Swedish National Infrastructure for Computing (SNIC).

List of papers

This thesis is based on the following papers.

Paper I Angelika Schwarz and Lars Karlsson. Scalable Eigenvector Computation for the Non-Symmetric Eigenvalue Problem. In *Parallel Computing*, volume 85, 131–140, Elsevier, 2019.

Paper II Carl Christian Kjelgaard Mikkelsen, Angelika Beatrix Schwarz and Lars Karlsson. Parallel robust solution of triangular linear systems. In *Concurrency and Computation: Practice and Experience*, e5064, Wiley Online Library, 2018.

Paper III Angelika Schwarz, Carl Christian Kjelgaard Mikkelsen. Robust Task-Parallel Solution of the Triangular Sylvester Equation. Submitted, 2019.

List of contributions

Paper I	Joint development and improvement of the presented algorithms, programming, joint analysis of bottlenecks, writing of the first draft, revising the draft jointly, designing and conducting the experiments, joint analysis and presentation of the numerical results.
Paper II	Joint design of the experiments, conducting the experiments and plotting the results, describing the test environment, reading and commenting on the draft.
Paper III	Derivation and programming of the algorithm, writing and revising the draft, designing and conducting the experiments, interpretation and presentation of the numerical results.

Contents

Chapter 1
Introduction

The solution of triangular linear systems is a central building block in many numerical algorithms. The LU decomposition, for example, expresses a matrix as the product of a lower triangular matrix and an upper triangular matrix. This reduces the solution of a general linear system to the solution of two triangular systems. A variation arises from shifted quasi-triangular linear systems, where the matrices exhibit 2-by-2 blocks on the diagonal. These systems can be solved through a modified forward or backward substitution, where a small linear system is solved for every 2-by-2 block. Another variation arises from the protection against floating-point overflow. Overflow protection can be addressed by the solution of scaled linear triangular systems.

This thesis studies three systems of triangular type, whose solution requires variations of forward/backward substitution. We discuss how to modify the base algorithms such that the implementations achieve high performance. For simplicity, the rest of this thesis assumes that the matrices attain an upper (quasi-)triangular shape.

1.1 A high-performance backward substitution

Algorithms that express most of the computation as level-3 BLAS operations [5, 9], i.e., matrix–matrix computations, commonly achieve high performance on modern computer systems. Level-3 BLAS operations process $\mathcal{O}(n^3)$ operations on $\mathcal{O}(n^2)$ data. Facing the widening gap between memory and processor speed, efficient implementations of level-3 BLAS operations hide slow memory accesses behind the $\mathcal{O}(n)$ ratio of computation/data through efficient usage of the cache hierarchy. Many numerical libraries including LAPACK [2], PLASMA [8], and libflame [36] have reformulated algorithms to be rich in level-3 BLAS.

Backward substitution is a central algorithm in numerical analysis. In order to exhibit the $\mathcal{O}(n)$ ratio of computation/data, backward substitution has to process many right-hand sides simultaneously. The corresponding level-3 BLAS library routine is known as `xTRSM`. Highly optimized implementations of `xTRSM` are available in, for instance, OpenBLAS [30] or Intel MKL [23].

Some algorithms require a variation of backward substitution. This includes, for example, when the system matrix is upper quasi-triangular rather

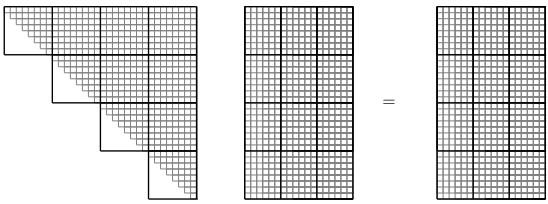

Figure 1.1: Sample tiled triangular system. The 32-by-32 triangular matrix is partitioned into 4-by-4 tiles of size 8-by-8. The 32-by-18 right-hand side matrix is partitioned into 4-by-3 tiles of size 8-by-6. The solution matrix is partitioned analogously to the right-hand side.

than upper triangular. Besides that, backward substitution-based algorithms can encounter floating-point overflow during the solution process. The avoidance of this type of overflow can be addressed with another variation of backward substitution. The implementations of these variations of backward substitution are less optimized than `xTRSM`. This thesis addresses three problems that require a variation of backward substitution, analyzes the bottlenecks in existing implementations and demonstrates how to achieve a high fraction of the peak performance.

We cast these variations of standard backward substitution in the form of tiled algorithms. Tiled algorithms partition the matrices into contiguous submatrices and have been successfully implemented in numerical libraries like PLASMA [8] and libflame [36]. A tiled approach for backward substitution-based problems is advantageous for several reasons. First, tiling, possibly recursively applied, harnesses the cache hierarchy. Second, most of the computation can be cast as matrix–matrix multiplications (`xGEMM`). Hence, the implementation benefits from a highly-optimized `xGEMM` implementation. Third, parallelism can be introduced easily in the form of task parallelism. Each task operates on at least one tile. The tasks are scheduled by a runtime system subject to data flow dependences.

A tiled backward substitution algorithm can be derived from serial backward substitution. We demonstrate this for the system $UX = B$ with the upper triangular m-by-m matrix U and the dense right-hand sides B of size m-by-n. The solution X has the same dimension as B. For simplicity, we assume that m and n are integer multiples of the tile sizes b_m and b_n, respectively[1].

A simple tiled backward substitution can be obtained by tiling the triangular matrix U such that tiles on the diagonal are square. Then the b_m-by-b_m tiles of U are U_{ij}, $i, j = 1, \ldots, m/b_m$. The right-hand side matrix B exhibits a conformal partitioning. Then the b_m-by-b_n tiles of B are B_{jk}, $j = 1, \ldots, m/b_m$, $k = 1, \ldots n/b_n$. The solution matrix X is partitioned analogously to B. A sample partitioning is illustrated in Figure 1.1.

[1] If m and n are not integer multiples of the tile sizes b_m and b_n, the index computation of the tiles is more complicated, but the general structure of the algorithm remains unaltered.

Algorithm 1 presents a tiled backward substitution algorithm. Tiled backward substitution has two nested loops which iterate over the tile columns (line 1) and the tile rows (line 2) of the solution. For each iteration of the second loop a new tile of the solution X is computed (line 3). This can be accomplished with an inner backward substitution algorithm. The newly computed tile X_{jk} is used to update above-lying tiles (lines 4 and 5).

Algorithm 1: Tiled backward substitution solver $UX = B$

```
1 for k ← 1 : n/b_n do
2     for j ← m/b_m : −1 : 1 do
3         Solve U_jj X_jk = B_jk;    // Inner backward substitution
4         for i ← 1 : j + 1 do
5             B_ik ← B_ik − U_ij X_jk ;          // Tile updates
```

Next we consider two variations that require a modification of standard tiled backward substitution. First, the system matrix is upper quasi-triangular. A quasi-triangular matrix is block triangular with 1-by-1 or 2-by-2 blocks on the diagonal. A partitioning avoids the scattering of information by not splitting these 2-by-2 blocks across tiles. The inner backward substitution algorithm has to be modified to handle 2-by-2 blocks on the diagonal. Quasi-triangular matrices occur in Paper I and Paper III. Second, backward substitution and its variation with quasi-triangular matrices are prone to overflow. The algorithms can be modified such that overflow is avoided. This requires dynamic scaling of the solution whenever an operation can exceed the overflow threshold. Paper II and Paper III demonstrate how overflow protection and scaling events can be implemented efficiently.

The following three sections introduce the problems addressed in the papers and summarize how standard backward substitution was modified.

1.2 Computation of Eigenvectors for the Non-Symmetric Standard Eigenvalue Problem

Assume that $A \in \mathbb{R}^{n \times n}$ is dense and non-symmetric. The standard eigenvalue problem $Ay_\ell = \lambda_\ell y_\ell$ aims at computing eigenvalues λ_ℓ and corresponding (right) eigenvector $y_\ell \neq 0$. Although the real-valued A can have real and complex eigenvalues and eigenvectors, all computations can be executed in real arithmetic, saving memory and flops compared to complex arithmetic. Real arithmetic, however, makes the computation more complicated.

We describe two standard approaches to computing eigenvectors. The first approach computes the eigenvectors from the real Schur form [13, 10]. The real Schur form can be computed with the QR algorithm [14, 37]. The QR algorithm in its original form dates back to Francis [11, 12]. The modern version [6, 7]

implemented in LAPACK proceeds as follows. The matrix A is first reduced to upper Hessenberg form $H = Q_0^T A Q_0$ with an orthogonal matrix Q_0. A matrix in upper Hessenberg form has zero entries below the first subdiagonal. In a second step, H is iteratively reduced to the real Schur form $T = Q_1^T H Q_1$, where T is upper quasi-triangular and Q_1 is orthogonal. Together, we obtain the real Schur decomposition $T = Q^T A Q$, where $Q = Q_0 Q_1$. Since all transformations are orthogonal similarity transforms, in exact arithmetic A, H and T have the same eigenvalues. The second step is computed using the multi-shift QR algorithm with aggressive early deflation (AED). The multi-shift QR algorithm is an iterative procedure, which converges to the real Schur form. In real arithmetic, there is no transformation that reduces the 2-by-2 blocks further. Aggressive early deflation is a technique to speed up the convergence of the QR algorithm. It identifies already converged eigenvalues, which can be deflated. The QR algorithm can proceed on a smaller matrix, excluding the already converged eigenvalues.

Provided that all similarity transformations were recorded, an eigenvector of A can be obtained through a two-step procedure from the real Schur form [14, 35]. First, an eigenvector $x \neq 0$ of T corresponding to an eigenvalue λ is computed. This eigenvector satisfies $(T - \lambda I)x = 0$, for readability written assuming complex arithmetic. The eigenvalue λ can be read off from the diagonal of the upper quasi-triangular T; a 1-by-1 block corresponds to a real eigenvalue, a 2-by-2 block corresponds to a pair of complex conjugate eigenvalues. A solution to $(T - \lambda I)x = 0$ can be computed through a variant of backward substitution. Whenever 2-by-2 blocks are encountered on the diagonal of T during the backward substitution, a small linear system has to be solved. Four cases can occur, arising from a real/complex eigenvalue λ encountering a 1-by-1/2-by-2 block. The second step backtransforms x to the original basis $y = Qx$.

The second approach is inverse iteration [24, 38, 14]. Inverse iteration assumes that an approximation $\hat{\lambda}$ to an eigenvalue of A is available. A sequence of vectors $y^{(k)}$ is generated by solving

$$(A - \hat{\lambda} I)y^{(k)} = s^{(k)} y^{(k-1)}, k \geq 1. \tag{1.1}$$

The scalar $s^{(k)}$ serves to normalize $y^{(k)}$. If the sequence of $y^{(k)}$ converges, an eigenvector corresponding to an eigenvalue closest to $\hat{\lambda}$ has been approximated. The requirement of having a good approximation $\hat{\lambda}$ available makes inverse iteration attractive for problems where approximations to the relevant eigenvalues are known. This is, for example, the case for vibration analysis of structures which constitute perturbations of well-known systems [4, 19].

A more general form of inverse iteration allows for a variable shift in the iterations. The linear systems $(A - \hat{\lambda}^{(k)} I)y^{(k)} = s^{(k)} y^{(k-1)}$, $k \geq 1$, can be solved through variants of inverse iterations like for instance Rayleigh quotient iteration [31].

Next we focus on standard inverse iteration (1.1). Practical implementations exploit that all iterations use the same matrix $A - \hat{\lambda} I$. As first proposed

by Wilkinson [38], the LU decomposition with partial pivoting is computed in a preprocessing step as $P(A - \hat{\lambda}I) = LU$. Here, P is a permutation matrix, L is unit lower triangular and U is upper triangular. Then in each iteration $Lc^{(k)} = Py^{(k-1)}$ and $Uy^{(k)} = s^{(k)}c^{(k)}$ are solved.

To reduce the flop count for the solution of the linear system, A can be reduced to upper Hessenberg form through a similarity transformation. If the eigenvalue $\hat{\lambda}$ is complex, the Hessenberg form also reduces the memory requirement. The imaginary parts of the complex LU factors can be stored in the zeroed part of the Hessenberg matrix. Although the LU factors are complex, the computation can be executed in real arithmetic. Solution strategies to efficiently solve shifted Hessenberg systems are discussed in [20, 21]. The upper Hessenberg form is assumed in the inverse iteration implementation `DLAEIN` available in LAPACK 3.7.0. This assumption is particularly justifiable when no approximations of the eigenvalues are known. In that case, A is reduced to Hessenberg form to compute the eigenvalues with the QR algorithm *without* accumulating the orthogonal matrices. Hence, the similar Hessenberg form is already available for the inverse iteration.

Paper I adopts the first approach and computes the eigenvectors from the real Schur form. Aiming for a compute-bound computation, we assume that several eigenvectors corresponding to distinct eigenvalues are sought. Previous work has addressed the compute-bound eigenvector computation of the eigenvectors in complex arithmetic on distributed memory architectures [1, 32]. In complex arithmetic, T is upper triangular and the first step of the eigenvector computation corresponds to a multi-shift backward substitution. Paper I contributes a compute-bound implementation in real arithmetic.

1.3 Robust Solution of Triangular Systems

Consider the system

$$\begin{bmatrix} 0.5 & -1 & & & \\ & 0.5 & -1 & & \\ & & \ddots & \ddots & \\ & & & 0.5 & -1 \\ & & & & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{1.2}$$

The exact solution is

$$x = \begin{bmatrix} 2^m \\ 2^{m-1} \\ \vdots \\ 2^2 \\ 2^1 \end{bmatrix}. \tag{1.3}$$

The component-wise relative condition of a linear system is given by Skeel's condition number [22], which is $2m-1$ for (1.2). The computation can be executed without rounding errors because all intermediate quantities are machine numbers. If $m = 1024$, the solution computed in double-precision arithmetic is

$$x = \begin{bmatrix} \texttt{inf} \\ 2^{1023} \\ \vdots \\ 2^2 \\ 2^1 \end{bmatrix}. \tag{1.4}$$

The first entry of x overflows. An overflow occurs whenever the exponent is too large to be represented in the exponent field. For IEEE binary 64 floating-point numbers ("doubles"), the overflow threshold is $\Omega = 2^{1023}$. Overflow can be avoided by representing the solution x as a scaled vector $x = \gamma^{-1}y$. The scaling factor $\gamma \in (0, 1]$ effectively extends the exponent field. For (1.4), a scaled vector is

$$x = \left(\frac{1}{2}\right)^{-1} \begin{bmatrix} 2^{1023} \\ 2^{1022} \\ \vdots \\ 2^1 \\ 2^0 \end{bmatrix}. \tag{1.5}$$

The scaled representation avoids overflow. We call an algorithm robust if overflow cannot occur at any point during the computation.

Robust algorithms for the solution of triangular systems were developed by Andersson [2]. These algorithms solve the scaled triangular linear systems $Ty = \gamma b$ or $T^T y = \gamma b$. Here, T is a triangular matrix and b is a vector of conforming length. The scaling factor $\gamma \in (0, 1]$ is computed alongside with the solution y. Overflow is avoided by dynamic scaling. Whenever an operation could exceed the overflow threshold, the solution vector y is scaled. After scaling, the operation can be executed safely. Scaling can be required several times. For each scaling event, a scaling factor is computed, by which the current representation of the solution is scaled. The final scaling factor γ corresponds to the product of the scaling factors computed during the solution process.

LAPACK 3.7.0 implements Andersson's robust algorithms for triangular systems as `xLATRS`. Based on upper bounds, `xLATRS` identifies if overflow can occur during the computation. Based on this assessment, different solvers are used. If overflow cannot occur, `xLATRS` falls back to its non-robust counterpart `xTRSV`, which solves a triangular system with one right-hand side. If overflow could occur, a scaled triangular linear system is solved and dynamic scaling prevents overflow.

Since `xLATRS` scales the *entire* solution whenever scaling is triggered, cascades of scaling events incur significant overhead. Kjelgaard Mikkelsen and

Karlsson [28] improve the overflow protection by delimiting scaling events. They partition the right-hand side into tiles. Overflow protection that guards tile updates is introduced. Instead of a global scaling factor, local scaling factors track the scaling level of each tile. Scaling affects only those tiles that are involved in the operation rather than the entire tile column. In a postprocessing step, the local scaling factors are reduced to a global scaling factor. The global scaling factor corresponds to the smallest local scaling factor. The tiles are rescaled w.r.t. the global scaling factor.

Aiming for a compute-bound backward substitution, a robust backward substitution algorithm should process many right-hand sides. For this purpose, let the right-hand sides be arranged as the m-by-n matrix $B = [b_1, \ldots, b_n]$. Each right-hand side b_i is associated with a scaling factor $\gamma_i \in (0, 1]$, $i = 1, \ldots, n$. When the scaling factors are arranged as $\Gamma = \mathrm{diag}(\gamma_1, \ldots, \gamma_n)$, the scaled linear system $TY = B\Gamma$ is solved for Y. This problem demands for a robust version of `xTRSM`. Neither BLAS nor LAPACK comprise a robust counterpart to `xTRSM`.

The library Elemental [29] offers a robust, more general version of `xTRSM` in the form of `SafeMultiShiftTrsm`. This routine solves $TY = B\Gamma + Y\Lambda$, where the shifts $\lambda_1, \ldots, \lambda_n$ are arranged as $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. By setting the shifts to zero, the problem is reduced to $TY = B\Gamma$. `SafeMultiShiftTrsm` implements robustness by using one global scaling factor per right-hand side. Each right-hand side is kept consistently scaled throughout the computation. Since a scaling event rescales the entire corresponding right-hand side, frequent scaling hampers the performance.

Paper II contributes a parallel robust tiled backward substitution for $TY = B\Gamma$, which has little overhead from scaling. It demonstrates that the usage of local scaling factors allows sustaining parallel scalability even when a lot of numerical scaling is necessary.

1.4 Robust Solution of the Triangular Sylvester Equation

The triangular Sylvester equation is

$$AX + XB = C,$$

where A and B are quasi-triangular matrices and C is dense. For simplicity, we assume both A and B to be upper quasi-triangular. The triangular Sylvester equation arises as an intermediate problem in the Bartels-Stewart algorithm [3, 34] for the general Sylvester equation. Although the solution of the triangular Sylvester equation is a well-studied problem, Paper III contributes an efficient overflow protection scheme for task-parallel solvers.

Many libraries offer implementations for the triangular Sylvester equation. LAPACK implements the scalar solver `DTRSYL`, the Fortran library `recsy` [15, 25, 26, 27] implements recursive solvers, `libflame` implements

the tiled task-parallel `FLASH_Sylv` [33], and distributed memory implementations are available in `scasy` [15, 16, 17, 18] as well as Elemental [32].

The triangular Sylvester equation can be solved through a variation of backward substitution. The two major differences are as follows. First, since A and B are upper quasi-triangular, four cases occur: A diagonal 1-by-1/2-by-2 block of A encounters a diagonal 1-by-1/2-by-2 block of B. Each case is handled in the form of the solution of a small Sylvester equation. Second, since the solution matrix X appears on both the right and the left side, linear updates are executed as right updates and left updates.

Since the solution of the triangular Sylvester equation is based on backward substitution, overflow can occur. Paper III addresses overflow by dynamically computing a scaling factor $\alpha \in (0, 1]$ such that the solution X to the scaled triangular Sylvester equation $AX + XB = \alpha C$ is computed without exceeding the overflow threshold at any point. However, not all solvers are robust. Paper III contributes a robust task-parallel solver, which can be viewed as an enhancement of the non-robust, task-parallel solver `FLASH_Sylv` available in `libflame` 5.1.0-58.

Our motivation to enhance `FLASH_Sylv` is that it achieves a large fraction of the peak performance in a parallel environment. For this purpose, Paper III designs a robust solver that has the same degree of parallelism. The key idea to not impede parallelism is local scaling factors. The local scaling factors track the scaling level of each tile and decouple tile updates. Paper III demonstrates that our robust solver can achieve a performance similar to the non-robust `FLASH_Sylv`.

Chapter 2
Summary of Papers

2.1 Paper I

Paper I addresses the computation of standard eigenvectors from the real Schur form. The standard eigenvalue problem $AY = Y\Lambda$ aims at computing eigenvalues $\lambda_1, \ldots, \lambda_k$ arranged as $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$ and corresponding non-zero eigenvectors $y_1, \ldots, y_k$ arranged as $Y = [y_1, \ldots, y_k]$. We assume that $A \in \mathbb{R}^{n \times n}$ is dense and non-symmetric and that $k \leq n$. We assume further that the real Schur decomposition of A is available as $A = QTQ^T$. Here, Q is orthogonal and T is upper quasi-triangular.

The eigenvectors Y can be computed through a two-step procedure [13, 10]. The first step computes the eigenvectors of T through a variant of backward substitution. The second step backtransforms this solution to the original basis. The LAPACK 3.7.0 routine `DTREVC3` implements the first step with level-2 BLAS operations, i.e., matrix–vector computations. The second step is implemented with level-3 BLAS if all eigenvectors are sought, i.e., $k = n$. Since level-2 BLAS runs at a much lower fraction of the peak performance than level-3 BLAS, the first step is a bottleneck. Paper I addresses this bottleneck.

Paper I casts the computation in terms of a tiled algorithm. The first step computes an eigenvector $x_\ell \neq 0$ of T corresponding to λ_ℓ by solving $(T - \lambda_\ell I)x_\ell = 0$, $\ell = 1, \ldots, k$. The system $(T - \lambda_\ell I)x_\ell = 0$ involves a different matrix $(T - \lambda_\ell I)$ for each eigenvalue. Since the matrices only differ on the diagonal, a partitioning into tiles such that 2-by-2 blocks are not split and diagonal tiles are square leads to a scheme which closely resembles standard tiled backward substitution. Operations that involve a diagonal tile require a modified backward substitution to handle the shift by λ_ℓ and the 2-by-2 blocks on the diagonal. The newly computed tile is used to update above-lying tiles with an `DGEMM` operation analogously to standard tiled backward substitution. As a consequence, the computation of $X = [x_1, \ldots, x_k]$ is dominated by matrix–matrix operations. Since X attains a generalized upper triangular form, the backtransform $Y = QX$ is implemented as a structure-exploiting tiled matrix–matrix multiplication. An example of the shape of X is given in Figure 2.1. Paper I parallelizes this tiled algorithm for shared memory and distributed memory systems.

Analogously to standard backward substitution, the entries in the solution

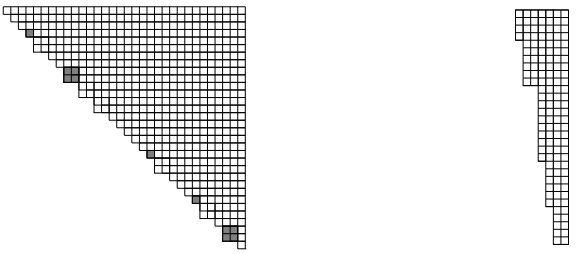

Figure 2.1: The eigenvector matrix X (right) corresponding to the selected blocks of the quasi-triangular matrix T (left) attains a generalized upper triangular form.

X can grow, which can result in an overflow. While the algorithms of Paper I are not yet robust, they provide a blueprint for an efficient eigenvector computation. In this sense, the paper can be viewed as a scalability study under ideal conditions. Adding overflow protection to render the algorithms robust is planned as future work, see Section 3.

2.2 Paper II

Triangular systems can be solved through backward substitution in the case of an upper triangular matrix and forward substitution in the case of a lower triangular matrix. Provided that the triangular system is well-conditioned, these procedures produce accurate solutions. In a practical implementation, the procedures can fail nevertheless. The solution can exceed the representable range of floating-point numbers and, as a consequence, contains infinities representing overflow. Paper II addresses the parallel solution of triangular systems such that overflow does not occur.

Overflow can be avoided by scaling the solution. For this purpose, a scaling factor is associated with the solution vector. Prior to executing an arithmetic operation, overflow protection logic evaluates if this operation could exceed the overflow threshold. If so, the entire solution vector is scaled such that the operation can then be executed without triggering overflow. As a consequence, the computed result always contains valid floating-point numbers.

The strategy of dynamic scaling during the solution process is implemented in the LAPACK 3.7.0 routine `xLATRS`. The routine `xLATRS` supports only one right-hand side. LAPACK 3.7.0 does not contain a routine for computing many right-hand sides simultaneously. Hence, if many right-hand sides are given, the solution vectors have to be computed with `xLATRS` one at a time. This approach misses out on the level-3 BLAS potential.

Paper II contributes a robust backward substitution algorithm for many right-hand sides, which relies heavily on level-3 BLAS. The idea of scaling factors is extended to many right-hand sides by associating one scaling factor

with each right-hand side. Aiming for a tiled algorithm, overflow protection logic was developed to cover tile updates. The implementation `Kiya` meets the expectations of a tiled algorithm and achieves a reasonable fraction of the peak performance. Moreover, the experiments indicate that the overflow protection logic does not hamper the parallel scalability.

2.3 Paper III

Paper III addresses the robust solution of the triangular Sylvester equation. Assuming A and B to be upper quasi-triangular, the scaled triangular Sylvester equation $AX + XB = \alpha C$ can be solved through a variation of backward substitution. Similarly to standard backward substitution, entries in the solution matrix can grow, possibly exceeding the overflow threshold. The main contribution of Paper III is a practical overflow protection scheme for the triangular Sylvester equation.

The need for a robust solver for the triangular Sylvester has been addressed by the scalar LAPACK 3.7.0 implementation `DTRSYL` or the recursion-based library `recsy` [15, 25, 26, 27]. Paper III contributes a tiled, robust algorithm for solving the triangular Sylvester equation. To the best of our knowledge, only non-robust tiled task-parallel solvers existed up to this point. By adding overflow protection, task-parallel solvers can now solve the same class of problems as `DTRSYL` or `recsy`.

Paper III adapts the overflow protection scheme developed in Paper II. Since tile updates are applied from the left and the right, the overflow protection logic is generalized to guard left and right tile updates. Moreover, scaling factors are associated with tiles rather than single right-hand sides. As a consequence, each scaling event affects entire tiles rather than single columns.

The scalability studies evaluate the cost of overflow protection logic and scaling. If scaling is not necessary, the cost of overflow protection is negligible, which makes the robust solver competitive with existing non-robust solvers such as `FLASH_Sylv` [33]. As expected, the performance degrades if overflow protection logic initiates rescaling. The experiments confirm that overflow protection does not impede parallel scalability.

Chapter 3
Future Work

Paper I derived a scalable algorithm for computing eigenvectors from the real Schur form. This algorithm, so far, lacks overflow protection. The next step is to add overflow protection logic, a step which has been worked towards in Paper II and Paper III. Paper II has examined the impact of overflow protection logic for standard tiled backward substitution in a parallel environment. The robust computation of the eigenvector requires a similar approach as Paper II. Each eigenvector is associated with one scaling factor. It remains to solve the small shifted linear systems that arise from 2-by-2 blocks on the diagonal. A similar problem has been encountered in the triangular Sylvester equation solver in Paper III. There, the 2-by-2 blocks on the diagonals required the robust solution of small Sylvester equations. These were solved through the robust solution of the corresponding small linear systems. Hence, Paper II and Paper III provide the foundations that, with some modifications, will render the eigenvector computation from Paper I robust.

Bibliography

[1] B. Adlerborn, C. C. Kjelgaard Mikkelsen, and L. Karlsson. Towards Highly Parallel and Compute-Bound Computation of Eigenvectors for Matrices in Schur Form, May 2017. NLAFET Working Note 10.

[2] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and A. McKenney. *LAPACK Users' Guide*. SIAM, 3rd edition, 1999.

[3] R. H. Bartels and G. W. Stewart. Solution of the Matrix Equation $AX + XB = C$. *Communications of the ACM*, 15(9):820–826, 1972.

[4] K.-J. Bathe and E. L. Wilson. Solution methods for eigenvalue problems in structural mechanics. *International Journal for Numerical Methods in Engineering*, 6(2):213–226, 1973.

[5] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[6] K. Braman, R. Byers, and R. Mathias. The Multishift QR Algorithm. Part I: Maintaining Well-Focused Shifts and Level 3 Performance. *SIAM Journal on Matrix Analysis and Applications*, 23(4):929–047, 2002. Revised edition of the original article from 1991.

[7] K. Braman, R. Byers, and R. Mathias. The Multishift QR Algorithm. Part II: Aggressive Early Deflation. *SIAM Journal on Matrix Analysis and Applications*, 23(4):948–973, 2002.

[8] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38–53, 2009.

[9] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[10] M. R. Fahey. New Complex Parallel Eigenvalue and Eigenvector Routines, August 2001. LAPACK Working Note 153.

[11] J. G. F. Francis. The QR Transformation A Unitary Analogue to the LR Transformation — Part 1. *The Computer Journal*, 4(3):265–271, 1961.

[12] J. G. F. Francis. The QR Transformation — Part 2. *The Computer Journal*, 4(4):332–345, 1962.

[13] M. Gates, A. Haidar, and J. Dongarra. Accelerating computation of eigenvectors in the dense nonsymmetric eigenvalue problem. In *International Conference on High Performance Computing for Computational Science*, pages 182–191. Springer, 2014.

[14] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 3rd edition, 1996.

[15] R. Granat, I. Jonsson, and B. Kågström. RECSY and SCASY library software: Recursive blocked and parallel algorithms for Sylvester-type matrix equations with some applications. In *Parallel Scientific Computing and Optimization*, pages 3–24. Springer, 2009.

[16] R. Granat and B. Kågström. Parallel Solvers for Sylvester-Type Matrix Equations with Applications in Condition Estimation, Part I: Theory and Algorithms. *ACM Transactions on Mathematical Software*, 37(3), 2010. Article No. 32.

[17] R. Granat and B. Kågström. Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part II: The SCASY Software Library. *ACM Transactions on Mathematical Software*, 37(3), 2010. Article No. 33.

[18] R. Granat, B. Kågström, and P. Poromaa. Parallel ScaLAPACK-style algorithms for solving continuous-time Sylvester matrix equations. In *European Conference on Parallel Processing*, pages 800–809. Springer, 2003.

[19] K. K. Gupta. Development of a unified numerical procedure for free vibration analysis of structures. *International Journal for Numerical Methods in Engineering*, 17(2):187–198, 1981.

[20] G. Henry. The Shifted Hessenberg System Solve Computation, 1994. Cornell University, NY, USA.

[21] G. Henry. A parallel unsymmetric inverse iteration solver. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 546–551, 1995.

[22] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.

[23] Intel Math Kernel Library. http://software.intel.com/en-us/mkl [May 2019].

[24] I. C. Ipsen. Computing an Eigenvector with Inverse Iteration. *SIAM Review*, 39(2):254–291, 1997.

[25] I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems – Part I: One-sided and coupled Sylvester-type matrix equations. *ACM Transactions on Mathematical Software*, 28(4):392–415, 2002.

[26] I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems – Part II: Two-sided and Generalized Sylvester and Lyapunov Matrix Equations. *ACM Transactions on Mathematical Software*, 28(4):416–435, 2002.

[27] I. Jonsson and B. Kågström. RECSY – A High Performance Library for Sylvester-Type Matrix Equations. In *European Conference on Parallel Processing*. Springer, 2003.

[28] C. C. Kjelgaard Mikkelsen and L. Karlsson. Blocked Algorithms for Robust Solution of Triangular Linear Systems. In *International Conference on Parallel Processing and Applied Mathematics*, pages 68–78. Springer, 2017.

[29] T. Moon and J. Poulson. Accelerating eigenvector and pseudospectra computation using blocked multi-shift triangular solves, July 2016. arXiv:1607.01477.

[30] OpenBLAS. http://www.openblas.net/ [May, 2019].

[31] A. M. Ostrowski. On the convergence of the Rayleigh quotient iteration for the computation of the characteristic roots and vectors. I. *Archive for Rational Mechanics and Analysis*, 1(1):233–241, 1957.

[32] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software*, 39(2):13, 2013.

[33] E. S. Quintana-Ortí and R. A. van de Geijn. Formal Derivation of Algorithms: The Triangular Sylvester Equation. *ACM Transactions on Mathematical Software*, 29(2):218–243, 2003.

[34] V. Simoncini. Computational Methods for Linear Matrix Equations. *SIAM Review*, 58(3):377–441, 2016.

[35] G. W. Stewart. *Matrix Algorithms. Volume II: Eigensystems*. SIAM, 2001.

[36] F. Van Zee, E. Chan, R. A. van de Geijn, E. S. Quintana-Ortí, and G. Quintana-Ortí. Introducing: The Libflame Library for Dense Matrix Computations. *Computing in Science & Engineering*, 2009.

[37] D. S. Watkins. The QR algorithm revisited. *SIAM Review*, 50(1):133–145, 2008.

[38] J. Wilkinson. The calculation of the eigenvectors of codiagonal matrices. *The Computer Journal*, 1(2):90–96, 1958.