Improved Orthogonality for Dense Hermitian Eigensolvers Based on the MRRR Algorithm

M. Petschow, E. S. Quintana-Ortì, P. Bientinesi

Financial support from the Deutsche Forschungsgemeinschaft (German Research Foundation) through grant GSC 111 is gratefully acknowledged.

Improved Orthogonality for Dense Hermitian Eigensolvers based on the MRRR algorithm

A Mixed Precision Approach

M. Petschow [*], E. S. Quintana-Ortí [†], P. Bientinesi [*]

Abstract

The dense Hermitian eigenproblem is of outstanding importance in numerical computations and a number of excellent algorithms for this problem exist. One of the fastest method is the MRRR algorithm, which is based on a reduction to real tridiagonal form. This approach, although fast, does not deliver the same accuracy (orthogonality) as competing methods like the Divide-and-Conquer or the QR algorithm. In this paper, we demonstrate how the use of mixed precisions in MRRR-based eigensolvers leads to improved orthogonality. At the same time, when compared to the classical use of the MRRR algorithm, our approach comes with no or only limited performance penalty, increases the robustness, and improves scalability.

1 Introduction

In [36], the authors describe how the use of "higher internal precision and mixed input/output types and precisions [in libraries] permits [...] to implement some algorithms that are simpler, more accurate, and sometimes faster." In particular, the internal use of higher precision provides the library developer with extra precision and a wider range of values, which may benefit the accuracy and robustness of numerical routines. As a major difference to software that uses arbitrary precision (e.g., Mathematica [58], Sage [51], and the LAPACK [2] adaptation MPACK [42]) to obtain any desired accuracy – as pointed out in [36] – the use of higher precision should not lower "performance significantly if at all." Our goal is to use mixed precisions to improve the accuracy, robustness and scalability of solvers for the *Hermitian eigenproblem* (HEP) based on the fast method of *Multiple Relatively Robust Representations* (MRRR or MR^3 for short) [18, 20, 21] with little or negligible impact on their execution time.

[*]RWTH Aachen, Aachen Institute for Advanced Study in Computational Engineering Science, 52062 Aachen, Germany. Electronic address: {petschow,pauldj}@aices.rwth-aachen.de

[†]Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12071 Castellón, Spain. Electronic address: quintana@icc.uji.es

The Hermitian eigenproblem consists of finding scalars $\lambda \in \mathbb{R}$ and nonzero vectors $v \in \mathbb{C}^n$ such that the equation

$$Av = \lambda v \tag{1}$$

holds for a given Hermitian matrix $A \in \mathbb{C}^{n \times n}$. In this case, λ is called an *eigenvalue* of A and v is called a corresponding *eigenvector*. Together, (λ, v) form an *eigenpair*. The Spectral Theorem for Hermitian matrices [43] ensures the existence of n eigenpairs (λ_i, v_i) such that the eigenvectors form a complete orthonormal set; that is for all $i, j \in \{1, 2, \ldots, n\}$

$$v_j^* v_i = \begin{cases} 1 & \text{if } j = i\,, \\ 0 & \text{if } j \neq i\,, \end{cases} \tag{2}$$

where we use v^* to denote the complex-conjugate-transpose of v. Frequently, only the eigenvalues or a subset of eigenpairs are desired. In this paper, we consider the case where eigenvectors are computed as well.

For dense matrices, there exist a number of excellent algorithms, which usually proceed in three stages: (1) reduction of A to a real symmetric tridiagonal matrix $T = Q^* A Q$ via a unitary similarity transformation; (2) solution of the symmetric tridiagonal eigenproblem $Tz = \lambda z$; and (3) back-transformation of the eigenvectors $v = Qz$. These solvers commonly differ only in the algorithm that is used for the tridiagonal part. Thus, differences in performance and accuracy are attributed to the *tridiagonal* eigensolver used in the second stage. A study on the performance and accuracy of various tridiagonal eigensolvers demonstrates that among the fastest is the MRRR algorithm [16]. Unfortunately, this approach delivers generally the least accurate results. These observations carry over to *dense* eigensolvers that differ only in their tridiagonal stage.

For a solver based on the MRRR algorithm, we present how the use of mixed precisions leads to more accurate results at very little or even no extra costs in terms of performance and memory usage. As a consequence, *MRRR becomes not only one of the fastest methods, but also as accurate or even more accurate than the competition.* Furthermore, we give compelling experimental evidence that the mixed precision approach *improves both robustness and parallel scalability.* Before we detail the discussion, in Section 1.1 we present that all these goals can be achieved for an eigensolver with single precision input/output.

1.1 Motivating example

High-performance numerical linear algebra libraries such as LAPACK contain routines with input/output data in the single and double precision formats (defined by the IEEE-754 standard [1, 31]) as these formats and their arithmetic is widely supported by both computer languages and hardware. We therefore concentrate on the two cases of single precision and double precision input/output. However, to increase the performance, accuracy, and robustness of routines, numerical libraries can make use of other data types *internally*, that is, invisible to the user. Table 1 shows the relevant floating point formats and their support on our test machine. For example, when we refer to single precision, we mean both the 32-bit data type and its unit round-off ε_s, while the term "precision" alone is a synonym for the unit round-off. Thus, using higher precision means using a data format with smaller unit round-off.

Name	**IEEE-754**	**Precision**	**Underflow**	**Support**
single	binary32	$\varepsilon_s = 2^{-24}$	$\omega_s = 2^{-126}$	Hardware
double	binary64	$\varepsilon_d = 2^{-53}$	$\omega_d = 2^{-1022}$	Hardware
extended	binary80	$\varepsilon_e = 2^{-64}$	$\omega_e = 2^{-16382}$	Hardware
quadruple	binary128	$\varepsilon_q = 2^{-113}$	$\omega_q = 2^{-16382}$	Software

Table 1: The various floating point formats used and their support on our hardware, an *Intel Xeon X7550 "Beckton"*. The ε-terms denote the unit round-off error (for rounding to nearest) and the ω-terms the underflow threshold. We frequently use the letters s, d, e and q synonymously with 32, 64, 80 and 128. Our use of the term *extended* is limited to the specific 80-bit format as specified in the table, opposed to the looser definition given in the original IEEE standard and many programming languages.

In this section, we concentrate on single precision input/output arguments. In this case, the use of higher precision internally – double precision in this case – improves accuracy *and* improves speed, robustness, and scalability of the code. For single precision input/output, the use of higher precision is "easy" as [36] "the computer's native double precision is a way to achieve [the] benefits [of a higher precision format] easily on all commercially significant computers."

We merely show experimental results for accuracy and performance, delaying the discussion of robustness, scalability, and how all the improvements are achieved to later sections. Figures 1 and 2 show accuracy and timing results for three different solvers: `SSYEVD` and `SSYEVR`, LAPACK's Divide-and-Conquer and MRRR implementations respectively, and our routine `mr3smp-mixed`, which uses the mixed-precision approach discussed in detail later. In this experiment, we generated three dense matrices for each matrix size with random entries drawn from uniform distribution in $[-1, 1]$ as input to the three solvers. We report the worst case error and the average time for computing *all* eigenpairs. More information about the experimental setup can be found in later sections.

Figure 1 demonstrates that our approach is – as desired – *more accurate*. At least for these test matrices, both the residual norms the numerical orthogonality, as defined in Section 1.2, are superior to `SSYEVR` and even `SSYEVD`. Figure 2 illustrates that the approach is not only more accurate, but also *faster* than the pure single precision routines `SSYEVR` and `SSYEVD`. The left plot displays the relative execution time compared to `SSYEVR` and suggests that all solvers require about the same execution time. Nonetheless, `mr3smp-mixed` is the fastest in all tests. This is emphasized in the right plot, where the execution time of only the tridiagonal stage is shown. Our tridiagonal solver is up to five times faster than `SSTEMR`, LAPACK's routine for the tridiagonal part within the solver `SSYEVR`. The reason for this (possibly) surprising

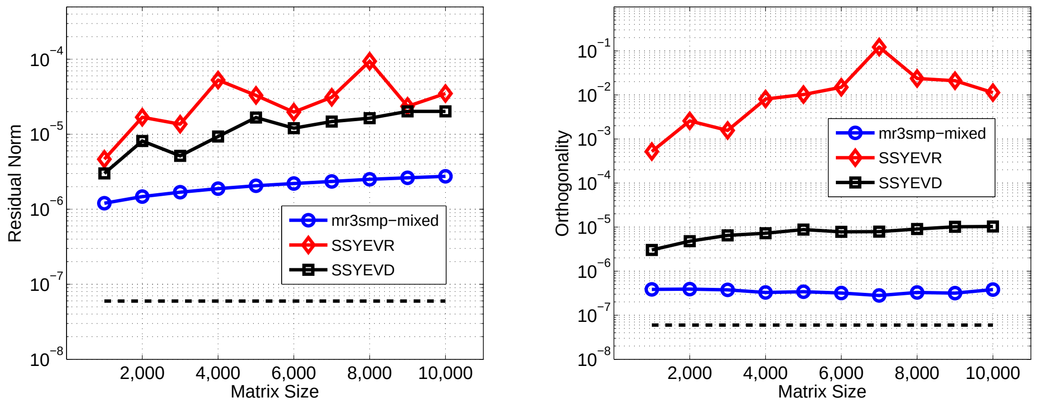


Figure 1: Residual norm $k_r \varepsilon_s$ and orthogonality $k_o \varepsilon_s$ as defined in Section 1.2. The dashed line corresponds to the single precision unit round-off ε_s.

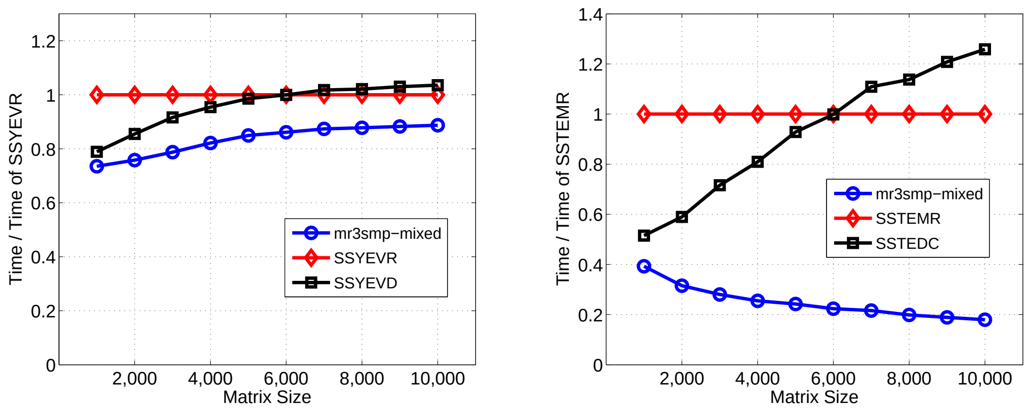


Figure 2: *Left:* Execution time relative to routine SSYEVR on an *Intel Xeon X7550 "Beckton"* processor. All routines are executed single-threaded. *Right:* Execution time of only the tridiagonal stage relative to routine SSTEMR, which is the routine used in SSYEVR for the tridiagonal stage.

result will be discussed in great detail below.

Before we discuss how all this is possible, in Section 1.2 we define exactly what we mean by *accuracy* in the context of Hermitian eigenproblems and in Section 1.3 give a brief overview of existing methods with their advantages and disadvantages.

1.2 Accurate solutions to the Hermitian eigenproblem

Given an Hermitian matrix A and a set of computed eigenpairs $(\hat{\lambda}_i, \hat{v}_i)$, with $\|\hat{v}_i\|_2 = 1$ and $i \in I$, how do we quantify their accuracy?[1] With ε denoting the input/output format's precision, we commonly say that, if the constants

$$k_r = \max_i \frac{\|A\hat{v}_i - \hat{\lambda}_i \hat{v}_i\|}{\varepsilon\|A\|} \quad \text{and} \quad k_o = \max_i \max_{j \neq i} \frac{|\hat{v}_j^* \hat{v}_i|}{\varepsilon}\,, \tag{3}$$

are moderate in magnitude, the eigenpairs are computed *accurately*. In other words, we require a small (norm-wise) relative backward error for each eigenpair and the (numerical) orthogonality of the set of eigenvectors. In the following, we accept this definition of accuracy without further discussion. Also, we generally display the quantities $k_r\varepsilon$ for the maximal residual norm and $k_o\varepsilon$ for the orthogonality of eigenvectors in all accuracy plots. The scaling by ε is mainly for the visual effect as one expects errors to be small.

The numbers k_r, k_o quantify the quality of the result. They depend on a number of factors: the norm $\|\bullet\|$, the algorithm $\mathcal{A}$ used[2], a set of parameters $\mathcal{P}$ of the specific implementation of $\mathcal{A}$, and the input matrix A – in particular its dimension n and its spectrum $\lambda[A]$. Once a specific norm is fixed[3], we thus have $k_r = k_r(\mathcal{A}, \mathcal{P}, n, \lambda[A], A)$ and $k_o = k_o(\mathcal{A}, \mathcal{P}, n, \lambda[A], A)$. From each algorithm $\mathcal{A}$, we isolate the set of parameters $\mathcal{P}$, which includes convergence criteria, thresholds, etc., so that if two algorithms only differ in $\mathcal{P}$, then they are considered the same. For a given algorithm $\mathcal{A}$, we can adjust $\mathcal{P}$ to achieve optimal performance while guaranteeing a desired accuracy.

To compare different algorithms, each with some fixed set of parameters, we try to factor in the dependence of the input matrix by obtaining results for a large set of test matrices. Ideally, this test set would be somewhat standardized and consist of a variety of application and artificial matrices in a wide range of sizes. Taking for each size n the average and worst case accuracy usually represents well the accuracy of an algorithm and gives some *practical* upper bounds $K_r(n)$ and $K_o(n)$ for k_r and k_o, respectively. In general, for a stable algorithm it is possible to provide *theoretical* upper bounds for k_r and k_o, independently of all properties of the input matrix but its size. These theoretical upper bounds are crucial for proving stability and for improving algorithms, but often greatly overestimate the actual error. As James Wilkinson said [28], "a priori bounds are not, in general, quantities that should be used in practice. Practical error bounds should usually be determined in some form of a posteriori error analysis, since this takes full advantage of the statistical distribution of rounding error [...]." Therefore, we evaluate the accuracy of an algorithm by executing it on a set of test matrices.

[1]Here and in the following, we use $\hat{\lambda}_i$ and $\hat{v}_i$ for computed approximations to the exact quantities λ_i and v_i, respectively, and assume that $\|\hat{v}_i\|_2 = 1$ holds exactly.

[2]See Section 1.3 for the discussion of five different algorithms: Jacobi's method, Bisection and Inverse Iteration, QR Iteration, Divide-and-Conquer, and the MRRR algorithm.

[3]Any norm can be used. We use the 1-norm for all our numerical experiments below.

Similar to the accuracy considerations, the average and worst case execution times can be used to assess the performance of different methods. This approach is for instance taken in [16] to evaluate the performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers. All performance numbers depend highly on the input matrices, the architecture, and the implementation of external libraries, such as vendor optimized BLAS [24], making the evaluation a difficult task. In our experiments, the test matrix set is not large enough to draw final conclusions, but the tests suggest general ideas on the behavior of the algorithms. Our experiments are underpinned by the outcomes in [16], where results of a large set of test matrices on different architectures are collected.

1.3 Algorithms for the dense Hermitian eigenproblem

For the solution of dense Hermitian eigenproblems, one is in the luxurious situation of being able to choose among different algorithms: Jacobi's method, Bisection and Inverse Iteration, QR Iteration, the Divide-and-Conquer method, and the already mentioned method of Multiple Relatively Robust Representations. The library user is required to make a selection among the available algorithms, each of them with its strengths and weaknesses and without a clear winner in all situations. At this point, we collect some of the advantages and disadvantages of each method as reported in various studies, e.g. [12, 14, 28, 43]. As already mentioned, we concentrate on the case where the eigenvectors are desired. With the exception of Jacobi's method, all methods are based on the three-stage approach that initially reduces the input matrix to real symmetric tridiagonal form.

- *Jacobi's method* (JM) [32], as stated in [14], "has been very popular, since it is implemented by a very simple program and gives eigenvectors that are orthogonal to working accuracy." It has the advantage of finding eigenvalues to high relative accuracy whenever this is possible [17, 43]. Furthermore, it is naturally suitable for parallelism [28, 49] and can be fast on strongly diagonally dominant matrices. On the downside, in terms of performance, it is usually not competitive to methods that are based on a reduction to real symmetric tridiagonal form [12]. Additionally, JM cannot compute a subset of eigenpairs at reduced cost.

- *Bisection and Inverse Iteration* (BI) [5, 55] has the advantage of being adaptable, that is, the method may be used to compute a subset of eigenpairs at reduced cost. On the other hand, it was shown that the current software can theoretically fail to deliver correct results [19, 12] and, due to the reorthogonalization of eigenvectors, its performance suffers severely on matrices with tightly clustered eigenvalues [12]. While BI has been the method of choice for computing a subset of eigenpairs for many years, the authors of [16] suggest that today MRRR "is preferable to BI for subset computations."

- *QR Iteration* (QR) [27, 35] is the work horse of dense eigencomputations, especially for unsymmetric and generalized eigenproblems [34, 12]. For the Hermitian problem the method faces the competition of the Divide-and-Conquer algorithm, which is usually faster and equally accurate. Recent improvements reduce this performance handicap, making it almost comparable to the fastest methods available for computing all eigenpairs [52]. Like Jacobi's Method, the current implementations of QR are very robust. The method has the distinct feature that in the dense case it requires the least amount of memory since the eigenvectors can overwrite the initial input matrix [52]. If only a moderate subset of eigenpairs has to be computed, other methods are preferred.

- *Divide-and-Conquer* (DC) [10, 29] is among the fastest and most accurate methods available [12, 16]. One advantage of the method lies in the inherent parallelism of the divide-and-conquer approach [25]; a second advantage is the reliance on the highly optimized matrix-matrix kernel. DCs drawbacks are that it requires the largest amount of memory of all methods [16, 48] and that most implementations cannot be used to compute a subset of eigenpairs at reduced cost; for DC and subset computations see [3].

- *Multiple Relatively Robust Representations* (MRRR) [18, 20, 21] cures BI from the explicit reorthogonalization of eigenvectors. As a result, the method is capable of computing k eigenpairs of a tridiagonal matrix in $\mathcal{O}(nk)$ operations and has therefore the lowest worst case complexity of all the tridiagonal eigensolvers. In the dense case, this means that the cost of the tridiagonal stage is asymptotically negligible compared with the $\mathcal{O}(n^3)$ cost of the reduction to tridiagonal form. MRRR is the method of choice for computing small subsets of eigenpairs, but often is the fastest method even when all eigenpairs are desired [16, 47, 48, 7]. In particular, as stated in [23], "the MRRR algorithm is usually faster than all other methods on matrices from industrial applications." Various studies [48, 47, 7, 54] show its excellent scalability in parallel environments. There are mainly two disadvantages of MRRR: (1) The computation relies on finding so called relative robust representations, which are special factorizations of shifted versions of the intermediate tridiagonal matrix that must fulfill certain restrictions, and there exists the danger of failing to find such representations; (2) In general, MRRR delivers the least accurate results of all the listed methods. In the dense case, the residual norms of all methods are comparable, but the orthogonality of the eigenvectors computed by MRRR is inferior to the other methods.

To summarize, all methods will generally deliver accurate results – i.e., with k_r and k_c in Eq. (3) of moderate size – but the MRRR-based solvers are less accurate than the others. In particular, the orthogonality of the eigenvectors can be several orders of magnitudes worse. On the other hand, MRRR is very fast and the method of choice when only a small subset of eigenpairs is desired. In this paper, we address the accuracy disadvantage of MRRR with a pragmatic approach based on mixed precisions. Additionally, this approach decreases the risk posed by the first of MRRR's drawback; experimental evidence for this claim is given in Section 4.

Our goal is to achieve the same accuracy as methods such as DC and QR, while being equally fast or faster. We restrict our experiments to the methods based on a prior reduction to real symmetric tridiagonal form. In this case, extensive tests in [16] on tridiagonal solvers confirm that (1) DC and MRRR are much faster than QR and BI, (2) DC and QR are the most accurate algorithms, and (3) MRRR is preferable to BI for subset computations. All these observations carry over to the dense case, as we assume that the solvers differ only in their tridiagonal part. Because of (1) and (2) and despite the fact that [52] shows that dense solvers based on QR can be made comparable to DC, we compare our results to DC but not QR. As references, we use LAPACK's implementations of DC, BI, and and MRRR.

1.4 Other related work

The term *mixed precision algorithm* is sometimes synonymously used for the following procedure: first solve the problem using a fast low-precision arithmetic, and then refine the result to higher accuracy using a higher precision arithmetic, see for example [4]. This mixed precision iterative refinement approach exploits the fact that there might exist a lower precision arithmetic faster than that of the input/output format. The larger the performance gap between the two arithmetics, the more beneficial is the approach. Iterative refinement (with and without using mixed precisions) has been most extensively studied for the solution of linear systems of equations [4, 15, 30, 8, 41], but other operations such as the solution of Lyapunov equations can benefit from it too [6].

We use of the term *mixed precision* in its more general form; that is, using two or more different precisions for solving a problem. In particular, we use a higher precision in the more sensitive parts of an algorithm to obtain accuracy, which otherwise could not be achieved. This approach is especially effective if the sensitive portion of the algorithm and/or the performance gap between the two arithmetics is small. Similarly to the mixed precision ideas for iterative refinement, the approach is quite general. We will demonstrate it on the specific case of MRRR-based eigensolvers.

The rest of the paper is organized as follows: In Section 2, we detail the discussion of our mixed precision approach in a general setting. In Section 3, we demonstrate the concept for an eigensolver with double precision input/output, and in Section 4 we provide further experimental results.

2 Improved Accuracy Through Mixed Precision

The technique is simple, yet powerful: use a higher than input/output precision inside the tridiagonal eigensolver to improve the overall accuracy. A similar idea was already mentioned in [18], in relation to a preliminary version of the MRRR algorithm, but was never pursued further. With many implementation and algorithmic advances since then (e.g., [37, 22, 56, 7, 54]), it is appropriate to investigate the approach in detail. To this end, we need a tridiagonal eigensolver that differentiates between two precisions:

(1) the input/output precision, say *binary_x*, and (2) the working precision *binary_y* with $y \geq x$. Although any x-bit and y-bit precisions might be chosen, in practice, only the specific values already shown in Table 1 are used for a high-performance library. For example, for the tridiagonal stage of the solver of Section 1.1, the input/output format is *binary32* (single) while the working format is *binary64* (double). In principle, any selection of y, even chosen at run time, leads to correct results; and if $y = x$, we have the original situation of a pure *binary_x* solver.

Provided the precision ε_y is sufficiently smaller than the input/output precision ε_x, say 5–6 orders of magnitude, we obtain improved accuracy to the desired level. What makes the use of (possibly much more costly) higher precision promising for a MRRR-based eigensolver are four properties of the tridiagonal part of the algorithm: (1) it has a lower complexity than the other two stages of the solver; (2) it performs the fewest flops of all tridiagonal solvers; (3) it does not make use of any higher level BLAS operations; (4) it is responsible for the deteriorated accuracy. Because of (1) and (2), the extra cost for large matrices are asymptotically negligible and we are perhaps willing to spend more time in the tridiagonal stage to improve the accuracy of the final result. Furthermore, since MRRR does not rely on higher level BLAS operations, we do not require in our mixed precision approach any optimized BLAS library for higher precisions, which might not be available. Point (4) is important as other methods would generally not benefit from improved accuracy in the tridiagonal stage.

We give four remark on the expected performance: (1) If the y-bit floating point arithmetic is not much slower than the x-bit floating point arithmetic, the approach obtains improved accuracy at no or only marginal extra cost in terms of execution time; (2) The additional cost due to the use of higher precision on large matrices is negligible; (3) The approach works best for complex input matrices as in this case more weight is on the reduction and back-transformation stages; (4) If only a small subset of eigenpairs is computed, the extra cost is reduced, as less y-bit floating point arithmetic is performed. In general, even if the higher arithmetic is significantly slower, mixed precision might work well for *large, complex* input matrices for which only a *fraction of eigenpairs* needs to be computed. Fortunately, this is a situation frequently occurring in practice, see for example [48, 50, 3] and the references therein.

In principle, the mixed precision dense eigensolver could be constructed by the following sequence of routines: (1) reduction to tridiagonal form in *binary_x* arithmetic followed by a conversion of the output to *binary_y*; (2) solution to the tridiagonal eigenproblem using *binary_y* arithmetic and conversion of the result to *binary_x*; and (3) back-transformation of the eigenvectors using *binary_x* arithmetic. For example, for single precision input/output, the sequence of LAPACK routines `SSYTRD`-`DSTEMR`-`SORMTR` with the intermediate data conversions would work. Subsequently, we refer to this as "the naive approach". While it would improve accuracy, it is for two reasons not optimal: (1) the approach potentially increases the memory requirement as we need to explicitly store the eigenvectors of the tridiagonal in the *binary_y* format; and (2) if the y-bit floating point arithmetic is much slower than the

x-bit floating point arithmetic, the performance suffers severely. The first issue is addressed in Section 2.1, where we discuss how the computation can be organized so that the memory requirements do not grow essentially. The second caveat is addressed in Section 3, where we demonstrate that the mixed precision approach becomes feasible even if we need to resort to much slower y-bit floating point arithmetic.

2.1 Memory cost

The memory management in the mixed precision tridiagonal solver is affected by the fact that the eigenvector matrix $Z \in \mathbb{R}^{n \times k}$ in *binary_x* format is used as intermediate work space. Thus if $y > x$, this part of the work space is not sufficient anymore for its customary use, which is the following: for every cluster of two or more close eigenvalues, the method stores up to $2n$ *binary_y* numbers in columns of Z. This is generally not possible if $y > x$. If we restrict $y \leq 2x$, it is still possible store the $2n$ *binary_y* numbers whenever the cluster of eigenvalues is of size four or larger. Thus, the computation must be carefully reorganized so that clusters containing less than four eigenvalues are processed without storing any data in Z temporarily. Additionally, after computing an eigenvector in *binary_y*, it is converted to *binary_x*, written into Z, and discarded. Since also some of the bookkeeping can be done using *binary_x*, we do not require (substantially) more memory.

At the beginning of the tridiagonal stage, we must also make a copy of the input matrix in order to cast it to *binary_y*. Since in the input is tridiagonal, the memory increase due to keeping a copy of the input is rather small. As the result, our mixed precision approach still needs only $\mathcal{O}(n)$ floating point numbers extra work space for the tridiagonal stage, although (most of) the computation is performed in a higher precision.

3 A Solver for Double Precision Input/Output

By Table 1 in Section 1.1, when dealing with *binary64* input/output arguments, we can use either *binary128* (quadruple) or *binary80* (extended) in our mixed precision approach. Both cases are analyzed in Section 3.1 and 3.2, respectively.

3.1 Quadruple precision

We discuss the use of quadruple precision first, as it closely resembles the single/double precision case. The major difference is that this time the *binary_y* arithmetic is not supported by hardware and is therefore much slower than *binary_x*. In general, the situation corresponds to the case where input/output are in the largest IEEE format supported by hardware. Thus, when using quadruple precision for the tridiagonal stage, we must identify which of the computation can still be performed in the much faster hardware supported double precision and relax the accuracy requirements of the tridiagonal eigensolver to improve its performance. We use the double/quadruple case as our case study, but the discussion is general, provided the quantities with index d, such as ε_d, and with index q are respectively replaced with the indices x and y, and the words 'double' and 'quadruple' are respectively replaced by '*binary_x*' and '*binary_y*'.[4] While the mixed precision approach for the single/double case works on virtually any of today's processors, the applicability of the double/quadruple case depends on the performance of the quadruple arithmetic, on the matrix size, and on whether the subset of eigenpairs to be computed is large.

3.1.1 Optimizing the tridiagonal stage for performance

There are two performance optimization approaches for our tridiagonal eigensolver: (1) Given a double precision eigensolver, replace a minimum of the computation to use quadruple and adjust a minimum of the parameters to guarantee a certain accuracy; (2) Given a quadruple precision eigensolver, use as much of double precision computation and loosen as many of the convergence criteria and thresholds as possible while still meeting the accuracy requirements.

We took the second approach as it is incremental: we can only apply some of the changes without breaking the functionality of the mixed precision solver. In the following, we give a list of optimizations that can be incorporated. A reader that is merely interested in the results, can safely skip to Section 3.1.4 at this point. Otherwise, Algorithm 1 presents the MRRR algorithm in sufficient detail for our discussion. Since the algorithm heavily uses the concept of so called *Relatively Robust Representations* of tridiagonals, we give the following definition adapted from [56].

Definition 3.1. *A (partial) Relatively Robust Representation (RRR) of the symmetric tridiagonal $T \in \mathbb{R}^{n \times n}$ is a set $\{x_1, \dots, x_m\}$ of $m \leq 2n-1$ scalars and a mapping $f : \mathbb{R}^m \to \mathbb{R}^{2n-1}$ that define the entries of T in such a way that small relative perturbations $\tilde{x}_i = x_i(1+\xi_i)$, with $|\xi_i| \leq \xi \ll 1$, will only cause relative changes in (some of) the eigenvalues and eigenvectors of $f(\tilde{x}_1, \dots, \tilde{x}_m)$ proportional to ξ. In particular, if M is an RRR for eigenvalue λ, we have $|\tilde{\lambda} - \lambda| = \mathcal{O}(n\xi|\lambda|)$, where $\tilde{\lambda}$ is the perturbed*[5] *eigenvalue of $\widetilde{M}$.*

For example, the non-trivial entries of bidiagonal factorizations $LDL^* = T - \sigma I$ for some shift $\sigma \in \mathbb{R}$ together with the mapping defined by the factorization often form (partial) RRRs and are used in all implementations we consider in this paper. Thus, if preferred by the reader, each occurrence of a representation M might be replaced with LDL^*. For a detailed discussion of the MRRR algorithm and the Relatively Robust Representations of tridiagonals in particular, we refer to [18, 20, 21, 56] and [45], respectively.

Algorithm 1 requires the input matrix to be *unreduced*, i.e., all the off-diagonal entries are non-zero. Because of this requirement and an efficiency benefit, small off-diagonal elements are set to zero in a preprocessing step. We discuss this step first.

[4]Furthermore, in the general discussion, we might assume $\varepsilon_x/\varepsilon_y \geq 10^5$, to improve the orthogonality to levels of other eigensolvers.

[5]Here and in the following, we use the notation $\mathcal{O}(x)$ informally as "of the order of x in magnitude".

Algorithm 1 The Core MRRR Algorithm

Input: An unreduced symmetric tridiagonal $T \in \mathbb{R}^{n \times n}$ and an index set $\Gamma \subseteq \{1, \ldots, n\}$ of desired eigenpairs.
Output: The eigenpairs $(\hat{\lambda}_i, \hat{z}_i)$ with $i \in \Gamma$.

```
1:  if only a small subset of eigenpairs desired or enough parallelism available then
2:      Compute initial eigenvalue approximations $\hat{\lambda}_i$ of $T$ for $i \in \Gamma$ via bisection.
3:  end if
4:  Compute (root) representation $M_0 := T - \tau I$ with $\tau \in \mathbb{R}$ — an RRR for all eigenpairs
    with index $i \in \Gamma$.
5:  Perturb $M_0$ entry-wise by a small random relative amount.
6:  if already computed initial eigenvalue approximations then
7:      Refine initial $\hat{\lambda}_i$ with respect to $M_0$ via bisection.
8:  else
9:      Compute eigenvalues $\hat{\lambda}_i$ with respect to $M_0$ for $i \in \{1, \ldots, n\}$ via the dqds algorithm.
10:     Discard $\hat{\lambda}_i$ if $i \in \{1, \ldots, n\} \setminus \Gamma$.
11: end if
12: Partition $\Gamma = \bigcup_k \Gamma_k$ according to the relative and absolute gaps of the associated eigen-
    values.
13: Form a work queue $Q$, set $depth := 0$ and enqueue each task $(\Gamma_k, M_0, depth)$.
14: while $Q$ not empty do
15:     Dequeue a task $(\widetilde{\Gamma}, M, depth)$.
16:     if $|\widetilde{\Gamma}| > 1$ then
17:         Compute $\widetilde{M} := M - \sigma I$ with $\sigma \in \mathbb{R}$ — a new RRR for all eigenpairs with index
            $i \in \widetilde{\Gamma}$.
18:         Refine $\hat{\lambda}_i$ with index $i \in \widetilde{\Gamma}$ with respect to $\widetilde{M}$ via bisection.
19:         Partition $\widetilde{\Gamma} = \bigcup_k \widetilde{\Gamma}_k$ according to the relative gaps of the associated eigenvalues.
20:         Set $depth := depth + 1$ and enqueue each $(\widetilde{\Gamma}_k, \widetilde{M}, depth)$.
21:     else
22:         Compute $(\hat{\lambda}_i, \hat{z}_i)$ with $i \in \widetilde{\Gamma}$ via inverse iteration with Rayleigh Quotient Correc-
            tion.
23:     end if
24: end while
```

Preprocessing: Splitting the problem into smaller sub-problems. Given the tridiagonal T by its diagonal entries $(c_1, ..., c_n)$ and its off-diagonal entries $(e_1, ..., e_{n-1})$, we set element e_k to zero whenever

$$|e_k| \leq \text{tol}_{split} \|T\| \,, \tag{4}$$

and therefore reduce the problem to smaller (unreduced) sub-problems [13, 43]. The splitting perturbs the eigenvalues by $\mathcal{O}(\text{tol}_{split}\|T\|)$ [13, 43]. After splitting, Algorithm 1 is invoked on each unreduced block.

In LAPACK's DSTEMR, the particular choice of the splitting tolerance is $\text{tol}_{split} = \varepsilon_d$. As our goal for the maximal residual norm does not change, the value of tol_{split} can be maintained and does not need to become the more restrictive ε_q. Since in the context of the dense problem, we cannot in general hope to obtain the eigenvalues to high relative accuracy, we normally employ this absolute splitting criterion. A relative criterion on the other hand, where $\|T\|$ becomes for example $\sqrt{|c_k c_{k+1}|}$ and which might by used for some tridiagonal eigenproblems, requires the tolerance to become ε_q. In both cases, the tolerance for splitting the matrix has no effect on the worst case orthogonality, which depends on the size of the largest sub-problem and therefore is usually improved by splitting. In the rest of this section, we assume that the preprocessing has been done and each sub-problem can be treated independently by invoking Algorithm 1. In particular, whenever we refer to matrix T, it is assumed to be unreduced; whenever we reference the matrix size n in the context of parameter settings, it refers to the size of the processed block.

A note on optimizing for performance. Given an MRRR tridiagonal eigensolver for quadruple precision, we are seeking to relax some of its convergence criteria and thresholds such that we achieve the desired accuracy for the double precision input and output arguments. Often these parameters can take a wide range of values. As the performance not only depends on these parameters in a highly non-trivial fashion, but also on the input matrix, the platform, the performance difference of the double and quadruple arithmetic, we cannot expect to select "optimal" values without some form of (auto-)tuning these parameters on a specific set of matrices and on a specific architecture. Nonetheless, we can choose the parameters in a way that give generally good performance. Additionally, often the tridiagonal eigensolver is not the performance bottleneck of the dense eigenproblem, and tuning these parameters is a secondary problem.

To fully understand the parameters and their effect on the accuracy, the reader should be familiar with the basics of the MRRR algorithm in general and the LAPACK implementation in particular. For a reader that does not have this background, we recommend to read [23] first or skip directly to the results in Section 3.1.4.

3.1.2 Optimizations that do not influence the final accuracy

Some modifications in our tridiagonal eigensolver stem from the fact that we use quadruple precision but require double precision accuracy. The following amendments are *not* perceivable in the output and can therefore equally be used in a tridiagonal eigensolver aiming for quadruple precision accuracy.

Line 2: Convergence criteria for bisection. The bisection procedure relies on the ability to count the number of eigenvalues smaller than given value $\mu \in \mathbb{R}$ [13]. Starting from an interval $[\underline{\lambda}, \overline{\lambda}]$ known to contain eigenvalue λ, for instance by virtue of the Gerschgorin bounds, bisection reduces the width of the interval by a factor two at every iteration. The process is converged when the interval satisfies $|\overline{\lambda} - \underline{\lambda}| \leq \mathrm{rtol}_0 \cdot \max\{|\underline{\lambda}|, |\overline{\lambda}|\}$ or $|\overline{\lambda} - \underline{\lambda}| \leq \mathrm{atol}$.

LAPACK's `DSTEMR` uses $\mathrm{atol} = \mathcal{O}(\omega_d)$ and $\mathrm{rtol}_0 = \sqrt{\varepsilon_d}$, i.e., computing the values to about 8 digits of accuracy (whenever possible). In our solver, we can adjust these parameters. If we desire relative accuracy in this stage, ω_d becomes ω_q, but we do not need to select $\mathrm{rtol}_0 = \sqrt{\varepsilon_q}$. Instead, we could for instance retain $\mathrm{rtol}_0 = \sqrt{\varepsilon_d}$. In fact, there is no obvious functional dependency of rtol_0 with the machine precision at all. We therefore changed the parameter to $\mathrm{rtol}_0 = 10^{-3}$ for all our experiments. Since the result will be refined before the eigenvalues are classified, the actual choice of rtol_0 is not crucial, although it influences performance. In principle, we could omit this step (Line 2) entirely.

Line 7 and 18: Convergence criteria for bisection. The intervals from the previous approximations to the eigenvalues are inflated and used as a starting point for limited bisection to refine the eigenvalue with respect to the RRR. The interval width is halved until it satisfies $|\overline{\lambda} - \underline{\lambda}| \leq \max\{\mathrm{rtol}_1 \cdot \mathrm{gap}(\lambda), \mathrm{rtol}_2 \cdot \max\{|\underline{\lambda}|, |\overline{\lambda}|\}\}$ or $|\overline{\lambda} - \underline{\lambda}| \leq \mathrm{atol}$.

LAPACK's choice in `DSTEMR` is $\mathrm{rtol}_1 = \sqrt{\varepsilon_d}$ and $\mathrm{rtol}_2 = \max\{5\sqrt{\varepsilon_d} \cdot 10^{-3}, 4\varepsilon_d\}$. This reflects the fact that the "eigenvalue needs to be known to high relative accuracy only at the point when the eigenvector is computed. At intermediate stages, while the representation tree is constructed, eigenvalues only need to be accurate enough to distinguish between large and small relative gaps" [53]. In our situation, we do not need to change rtol_1 and rtol_2 if also the minimum relative gap, given by the parameter *gaptol* (see below), remains mainly unchanged. For various reasons, we prefer to use smaller values for *gaptol* and possibly double precision arithmetic for the bisection computation. Thus, we choose $\mathrm{rtol}_1 = \mathrm{rtol}_2 = \max\{5 \cdot \mathrm{gaptol} \cdot 10^{-3}, 4\varepsilon_d\}$.

A side note: "if the proportion of desired eigenpairs is high enough or the full spectrum has to be computed, then all eigenvalues are computed by the more efficient dqds algorithm and unwanted ones are discarded" [23]. However, bisection might be more efficient – even if all the eigenvalues are needed – in a parallel environment. We therefore need to decide whether the dqds algorithm or bisection is used for the initial eigenvalue computation in Lines 1 to 11, see [47]. The changes in the convergence criteria and the precision used (see below) for the computations both influence the minimal amount of parallelism necessary for bisection to become faster than dqds.

Line 9: Precision used in the dqds algorithm. If the dqds algorithm is chosen, let $\mathcal{Z}_q = (q_1, e_1, q_2, e_2, \ldots, q_n, e_n)$ be the quadruple precision input – as defined for example in [46] – to a specific implementation of the algorithm and $\mathcal{Z}_d$ be $\mathcal{Z}_q$ cast to double precision. Then the conversion $\mathcal{Z}_q$ to $\mathcal{Z}_d$ corresponds to an element-wise relative perturbation of at most ε_d. Since $\mathcal{Z}_q$ is an RRR for all eigenvalues, and by Def. 3.1, the exact eigenvalues λ_q and λ_d of respectively $\mathcal{Z}_q$ and $\mathcal{Z}_d$ satisfy $|\lambda_q - \lambda_d| = \mathcal{O}(n\varepsilon_d|\lambda_d|)$. A double precision implementation of the dqds algorithm such as LAPACK's `DLASQ2` produces eigenvalues $\hat{\lambda}_d$ to high relative accuracy, that is $|\hat{\lambda}_d - \lambda_d| = \mathcal{O}(n\varepsilon_d|\lambda_d|)$. Thus, our computed eigenvalues satisfy $|\lambda_q - \hat{\lambda}_d| \leq |\lambda_q - \lambda_d| + |\hat{\lambda}_d - \lambda_d| = \mathcal{O}(n\varepsilon_d|\lambda_d|)$. As a final cast of the result to quad precision introduces no additional error, we obtain the eigenvalues with relative error of $\mathcal{O}(n\varepsilon_d)$. This is more than required, provided we prohibit *gaptol* from being too small, to classify the eigenvalues. In this way, we can perform a (noticeable) portion of the computation in the much faster double precision arithmetic.

Line 7 and 18: Precision used for bisection. By the same argument, we can use the much faster double precision computation when refining the eigenvalues. In Lines 7 and 18, given representation M_q in quadruple precision. A conversion to double, that is to M_d, corresponds to relative perturbations in the entries of M_q. The refinement of the eigenvalue $\hat{\lambda}_d$ with respect to M_d via bisection in double precision with a conversion to the quadruple yields $|\hat{\lambda}_q - \lambda_q| = \mathcal{O}(\mathrm{tol} \cdot |\lambda_q|)$, where the convergence parameter *tol* is chosen as discussed above.[6]

Note that in order to avoid under-/overflow in the double precision execution, it might be necessary that the input matrix T is scaled as in a pure double precision solver. This depends on the specific implementation of the bisection routine, see [13, 38]. Also, the relative convergence criterion cannot be taken smaller than what is achievable by bisection executed double precision.

As the initial eigenvalue approximation and refinement of eigenvalues can make up for a noticeable portion of the overall computation, the use of double precision arithmetic for these parts can speed up the mixed precision approach considerably in those cases where quadruple precision arithmetic is performed rather slow, e.g., when it is simulated in software.

3.1.3 Optimizations that influence the final accuracy

The following optimizations reflect the fact that we are not aiming for quadruple precision accuracy in the tridiagonal stage. The list is not exhaustive, but concentrates on the parameters that influence performance the most. Additionally, the freedom in the parameter selection due to the relaxed accuracy requirement may be used to increase robustness and scalability as well; both aspects are discussed further in Section 4.

Line 5: Random perturbation of the root representation. The elements of the representation M_0, $\{x_1, ..., x_m\}$, are perturbed by small random relative amounts $\tilde{x}_i = x_i(1+\xi_i)$ with $|\xi_i| \leq \xi$ for all $1 \leq i \leq m$ to break up tight clusters. This idea and its importance for robustness is discussed in detail in [22].

`DSTEMR` uses $\xi = 8\varepsilon_d$, which is perturbing each entry by a few units in the last place. In our situation, we can be more aggressive, using $\xi = \varepsilon_d$. Thus, about half of the digits in each entry of the representation M_0 are chosen randomly. This has two major effects: (1) it becomes very unlikely to encounter clusters within clusters under the classification criterion described below and (2) it helps significantly in finding an RRR with a shift close to one end of a cluster.

[6]At this point, we could use M_q to ensure at the cost of two Sturm counts per eigenvalue that the approximation as accurate as required. If not, the corresponding interval could be inflated and refined in quadruple precision. A similar approach could be taken after the double precision dqds algorithm was used.

Line 12 and 19: Minimum relative gap. The unfolding of Algorithm 1 highly depends on how the sets Γ and $\widetilde{\Gamma}$ are partitioned in Line 12 in Line 19, respectively. The criterion is based on the absolute and relative gap of the eigenvalues. To justify the choice made, we invoke the following classical theorem that can be found in similar form for example in [43, 12].

Theorem 3.1. *Given T (by any representation) and an approximation $(\hat{\lambda}, \hat{z})$, with $\|\hat{z}\|_2 = 1$, to the eigenpair (λ, z), let r be the residual $T\hat{z} - \hat{\lambda}\hat{z}$; then*

$$\sin\angle(\hat{z}, z) \leq \frac{\|r\|_2}{gap(\hat{\lambda})}\,, \tag{5}$$

with $gap(\hat{\lambda}) = \min_j\{|\hat{\lambda} - \lambda_j| : \lambda_j \neq \lambda\}$. Proof: See [43, 12, 56, 11].

In order to achieve $\sin\angle(\hat{z}, z) = \mathcal{O}(n\varepsilon)$, we must be able to compute residual norms with $\|r\|_2 = \mathcal{O}(n\varepsilon \text{gap}(\hat{\lambda}))$. Unfortunately, this is not always possible. In particular, if $\text{gap}(\hat{\lambda}) \ll \|T\|_2$, we cannot expect convergence of standard inverse iteration [18, 19]. One of the features of the MRRR algorithm is the computation of eigenpairs with $\|r\|_2 = \mathcal{O}(n\varepsilon|\hat{\lambda}|)$ [44, 21]. Thus, provided $\text{gap}(\hat{\lambda}) \gtrsim |\hat{\lambda}|$, say $\text{gap}(\hat{\lambda})/|\hat{\lambda}| > gaptol$, we can achieve $\sin\angle(\hat{z}, z) \leq \mathcal{O}(n\varepsilon/gaptol)$. The choice of *gaptol* is restricted by the loss of orthogonality that we are willing to accept. In practice, the value is often chosen to be 10^{-3}, reflecting a compromise between achievable orthogonality and practicality [20].

When our working precision becomes quadruple, the requirement can be relaxed. We have

$$\sin\angle(\hat{z}, z) \leq \mathcal{O}\left(\frac{n\varepsilon_q}{\text{gaptol}}\right)\,, \tag{6}$$

and the value of *gaptol* can be chosen (at least) as small as $\varepsilon_q\sqrt{n}/\varepsilon_d \approx 10^{-18}\sqrt{n}$.

The particular choice[7] of *gaptol* affects performance in multiple ways: the convergence criteria for bisection depend explicitly on it, and in order to use double precision computations of the eigenvalues as discussed above, the value cannot be chosen too small. On the other hand, the smaller the value of *gaptol* the less clustering of eigenvalues occurs – resulting in beneficial performance. Taking these considerations into account, we restricted *gaptol* to the interval $[10^{-12}, 10^{-3}]$ and optimized the value for performance. For the experimental results shown in later sections, we fixed *gaptol* to 10^{-10}.

Line 12: Minimum absolute gap. A classification of the eigenvalues based on their relative gaps alone can lead to bulky clusters, especially for large-scale problems. In order for an eigenvalue $\hat{\lambda}$ to be classified as well-separated by the relative criterion, we require

$$gap(\hat{\lambda}) > gaptol \cdot |\hat{\lambda}|\,, \tag{7}$$

[7]In the more general case, $\min\{10^{-3}, \varepsilon_y\sqrt{n}/\varepsilon_x\} \leq gaptol \leq 10^{-3}$.

i.e., the absolute gap for an eigenvalue large in magnitude must be equally large; even though some of the eigenvalues might be well-separated from the others in an absolute sense, they might not be in the relatively sense.

Large clusters can lead to deteriorated performance and impose problems for parallel distributed-memory codes such as [48, 54, 7]. The problem and how to overcome it is discussed in detail in [53, 54]. The solution is to supplement the relative classification criterion in Line 12 with one that takes also the absolute gap of the eigenvalues into account. Using a small adaption of the criterion in [53, 54], we classify an eigenvalue as well-separated if

$$gap(\hat{\lambda}) > \frac{|\lambda_n - \lambda_1|}{n-1} = avgap\,, \tag{8}$$

and additionally split clusters of eigenvalues into smaller clusters whenever the cluster has a separation from the rest of the spectrum greater than the average gap, *avgap*. This criterion can be justified by the analog of Theorem 3.1 for invariant subspaces of any dimension, see [43, 56, 11, 14, 53]. In fact, we could relax this condition to $\alpha \cdot avgap$ with $\alpha \ll 1$, if we also reduce the convergence criterion for inverse iteration accordingly and adjust *gaptol*. For practical matrices, the effect on performance is minor and we used $\alpha = 1$ in all experiments.

Line 22: Stopping criteria for inverse iteration. The MRRR algorithm usually performs Rayleigh Quotient iteration with twisted factorizations (controlled by bisection), a process described in detail in [23, 56, 53]. An eigenpair is accepted if $\|r\|_2 \leq \text{tol}_1 \cdot \text{gap}(\hat{\lambda})$, or if the 'Rayleigh Quotient Correction' $\leq \text{tol}_2 \cdot |\hat{\lambda}|$ and therefore cannot improve the eigenvalue approximation. The convergence criteria are discussed for example in [53].

`DSTEMR` uses $\text{tol}_1 = 4\varepsilon_d \log n$ and $\text{tol}_2 = 2\varepsilon_d$. While tol_2 must become $\mathcal{O}(\varepsilon_q)$, in most cases we can stop the inverse iteration process earlier by relaxing the tol_1 parameter. We chose $\text{tol}_1 = 4\varepsilon_d$ and $\text{tol}_2 = 2\varepsilon_q$ for the experiments.

3.1.4 The effect of the changes to performance and accuracy

To cite Beresford Parlett in [43]: "We might hope that results with low accuracy would cost less than those with high accuracy. In practice, however, numerical methods do not work that way."[8] While in general a trade-off between accuracy and performance is not that straightforward, within the MRRR algorithm it can be realized. In Fig. 3, we illustrate the net effect of all the changes we employed for the tridiagonal solver in isolation.

As the left plot shows, our adapted solver provides a remarkable up to five-fold speedup a on a set of application matrices (Table 2 of Section 4). The reference is `QSTEMR`, a pure quadruple precision tridiagonal eigensolver, which is basically obtained

[8]In fact, the MRRR algorithm is an excellent example of how high accuracy can lead to a fast algorithm.

by automatically translating LAPACK's `DSTEMR` by replacing `DOUBLE PRECISION` data types with `REAL*16` (quadruple) data types.

Such performance gains are not a miracle, but stem from our willingness to give up accuracy up to a certain level. This is depicted by the accuracy of our solutions in Fig. 3 (right).[9] Note that there is nothing special about the line given by ε_d; in fact, we can achieve any accuracy that is not better than the quadruple precision result. In our context however, it makes no sense to further reduce residuals and orthogonality since the results will be converted to double precision. At this place, we also want to point to Fig. 2 again, as now it becomes understandable why our mixed precision approach is faster than `SSYEVR` for single precision input/output.

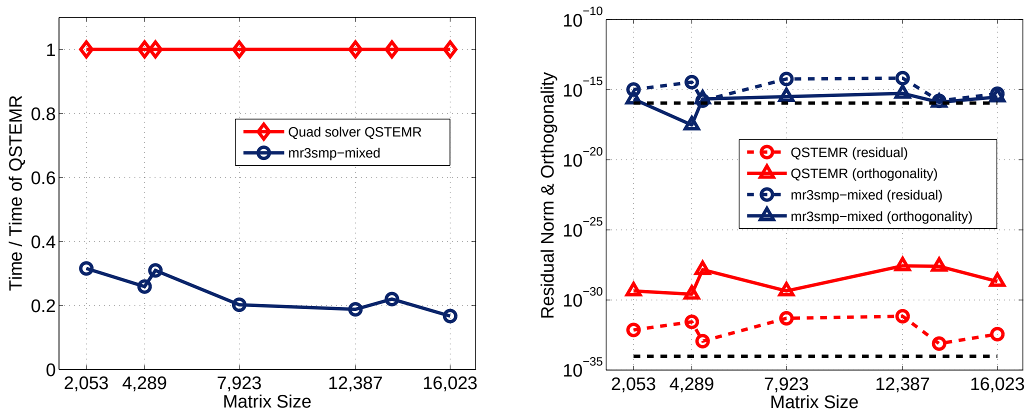


Figure 3: *Left:* Execution time of our adapted tridiagonal solver `mr3smp-mixed` relative to the quad precision routine `QSTEMR` computing all eigenpairs. *Left:* Corresponding accuracy for `mr3smp-mixed` and `QSTEMR`. As a reference, we added ε_d and ε_q as black dashed lines.

3.1.5 Portability of using quadruple precision

Presently, there are two major drawbacks in using quadruple precision: (1) it is rather slow and (2) the support through compilers and languages is limited. These drawbacks make it hard to produce portable code that runs at reasonable performance on different architectures with different compilers. In FORTRAN, it is possible to produce portable code by using the `REAL*16` data type. But, this type might not be supported by all compilers. In a C/C++ environment, the support is entirely compiler dependent. It is for instance supported via the `__float128` and `_Quad` data types with the GNU compilers and the Intel compilers, respectively. Alternatively, an external library implementing the software arithmetic might be used for portability. In any case, if quadruple precision is available, the performance highly depends on the specific implementation. Most likely, the support for quadruple precision will be improved

[9]In this experiments, we kept all the eigenpairs in quadruple precision in order to compute their residuals and the orthogonality in quadruple.

in the future.

3.2 Extended precision

Quadruple precision arithmetic is rather slow, as there exist no widespread hardware support at the time of writing. But, many current architectures have hardware support for a 80-bit extended floating point format (see Table 1). According to William Kahan [33] "this format is intended mainly to help programmers enhance the integrity of their Single and Double software, and to attenuate degradation by round-off in Double matrix computations of larger dimensions, and can easily be used in such a way that substituting Quadruple for Extended need never invalidate its use." Our situation fits into this picture.

As the unit round-off ε_e is only about three orders of magnitude smaller than ε_d, we cannot expect our orthogonality to be improved by more than this amount. The main advantage of the extended format is that compared to double precision its arithmetic comes without any or only a small loss in speed.[10] We therefore expect in all situations improved accuracy without any significant loss in performance. This is confirmed in all test we performed: the tridiagonal stage was never slowed down more than by a factor of two, which results in negligible extra cost for the dense eigensolver. The situation is slightly different between the double/extended and double/quadruple cases; in contrast to the double/quadruple case, we cannot relax the accuracy requirements of the tridiagonal eigensolver when using extended precision – all the extra precision is necessary to improve accuracy. Thus, even if the arithmetic is performed exactly at the same speed, we cannot expect the mixed precision eigensolver using extended to be faster than a solver using pure double precision exclusively.

Experimental results suggest that in many cases the MRRR-based eigensolver using extended precision obtains orthogonality comparable to that of methods such DC or QR. On the other hand, especially for larger matrices, the use of extended precision potentially leads to an orthogonality that is inferior to other methods (see Section 4).

3.2.1 Portability of using extended precision

Not all architectures support the 80-bit extended floating point format, so that its use is not generally portable. A C/C++ code that uses the `long double` data type (introduced in ISO C99) for the higher precision in the tridiagonal solver would achieve the desired result on most architectures. However, some compilers might interpret `long double` as either double or rarely even quadruple. In the first case, we do not gain (or lose) anything. In the later case, we gain accuracy but might lose performance depending on how quadruple is supported. FORTRAN code making use of extended precision is likely not to be portable, as not all compilers support the extended precision format `REAL*10`.

[10]A loss in performance might appear for two reasons: (1) the ability to use vectorized operations is lost, and (2) the algorithm is memory-bandwidth limited.

4 Experimental Results

So far we have not listed the details of the experimental results of the previous sections. At this point, we want to catch up on this. For all LAPACK results, we used version 3.3 in conjunction with version 10.2 of Intel MKL BLAS. For the mixed precision results, we used the same reduction and back-transformation routines as LAPACK in conjunction with a modified mixed precision version of the parallel solver presented in [47]. For the extended precision results, we used version 4.7 of the GNU compilers. In all other experiments, we used version 12.1 of Intel's compiler.[11] In all cases, we enabled optimization level `-O3`. All experiments were performed single-threaded on a Intel Xeon X7550 *"Beckton"* processor with nominal clock speed of 2 GHz.

In the single/double experiment of Section 1.1, we applied all the adjustments discussed in Section 3.1, with only one exception: we do not resort to single precision arithmetic in the dqds algorithm and bisection. In the double/extended experiments, we merely used extended as our higher precision arithmetic; in the double/quadruple case, we applied all the discussed changes.

Since for single precision input/output the mixed precision approach works very well, here we concentrate on double precision input/output. In this case, the mixed precision approach uses either extended or quadruple precision. We refer to these cases as `mr3smp-extended` and `mr3smp-quad`, respectively. The use of quadruple is more critical as it shows that the approach is applicable in many circumstances, even when the higher precision arithmetic used in the tridiagonal stage is much slower than arithmetic in the input/output format.

In this section, we confine ourselves to experiments on a small set of application matricesas listed in Table 2, coming from quantum chemistry and structural mechanics problems.[12] As the performance of the eigensolvers highly depend on the spectra of the input matrices, the platform of the experiment, the used BLAS library, and the implementation of the quadruple arithmetic, we cannot draw final conclusions about performance from these limited test. On the other hand, the orthogonality improvements are quite general and are observed also for a much larger test set originating from [39]. We do not report on residuals as *the worst case residual norms are generally comparable for all solvers.*

To better display the effects of the use of mixed precisions, the performance results are simplified in the following sense: As the routines for the reduction to tridiagonal form and the back-transformation of all solvers are exactly the same, we used for these stages the *minimum* execution time of all runs for each solver. In this way, the cost of the mixed precision approach becomes more visible and we do not have to resort to statistical metrics for the timings. We point out that especially for the subset tests, the run time of the tridiagonal stage for larger matrices is often smaller than the

[11]For all timings, we only compare routines compiled with the same compiler. That is the routines using extended are compared to routines compiled with the GNU compilers.

[12]These matrices are stored in tridiagonal form. In order to create real and complex dense matrices, we generated a random Householder matrix $H = I - \tau vv^*$ and applied the similarity transformation HTH to the tridiagonal matrix T.

Matrix	Size	Application	Reference
A	2,053	Chemistry	ZSM-5 in [26] and Fann3 in [39]
B	4,289	Chemistry	Originating from [9]
C	4,704	Mechanics	T_nasa4704 in [39]
D	7,923	Mechanics	See [7] for information
E	12,387	Mechanics	See [7] for information
F	13,786	Mechanics	See [7] for information
G	16,023	Mechanics	See [7] for information

Table 2: A set of test matrices.

fluctuations in the timings for the reduction to tridiagonal form.

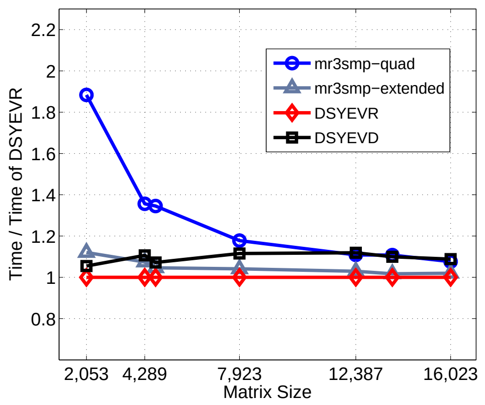
4.1 Real symmetric case

Computing all eigenpairs. Figure 4 refers to the computation of all eigenpairs. We report on the execution time of the mixed precision routines relative to LAPACK's MRRR (`DSYEVR`) and the obtained orthogonality. As a reference, results for LAPACK's DC (`DSYEVD`) are included. The orthogonality is improved using extended and quadruple precision. The left plot shows the performance penalty that we pay for the improvements. In particular, for larger matrices, the additional cost of the mixed precision approach becomes negligible, making it extremely attractive for large-scale problems. For example, for test matrices E, F, and G, our solver `mr3smp-quad` is as fast as `DSYEVD`, although it uses software simulated quadruple precision, while achieving better orthogonality. Since the quadruple arithmetic is currently much slower than the double one, `mr3smp-quad` carries a considerable performance penalty for small matrices. In our case, for matrices with $n < 2,000$, one must expect an increase in the total execution time of a factor larger than two. The situation is similar to the one reported in [4] for the iterative refinement of the solution of linear systems of equations, where the mixed precision approach comes with a performance penalty for small matrices.

In contrast to `mr3smp-quad`, the use of extended precision does not significantly increase the execution time even for smaller matrices, while still improving the orthogonality. As the reason for different performance is solely due to the tridiagonal eigensolver, in the left panel of Fig. 5 we show the execution time of the tridiagonal eigensolvers relative to LAPACK's MRRR (`DSTEMR`).

We want to remark that although the mixed precision approach slows down the tridiagonal stage compared to `DSTEMR` (at least with the current support for quadruple precision arithmetic, see Fig. 5), it has two features that compensate this disadvantage: it increases robustness and parallel scalability of the code. To underpin these statements, in Table 3 we present the recursion depth, the maximal cluster size and the number of times Line 17 in Algorithm 1 is executed when all eigenpairs are computed.

The table shows that in all cases `mr3smp-quad` computes all eigenpairs directly from the representation M_0 obtained in Line 4. Since this representation is definite, no danger of element growth in its computation exist (thus, the RRR can be found).



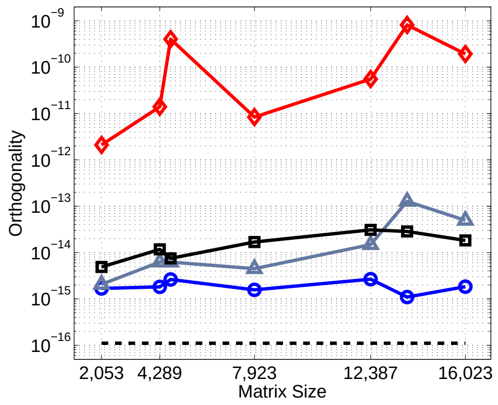


Figure 4: Computation of all eigenpairs. *Left:* Execution time relative to routine `DSYEVR`. *Right:* Orthogonality.

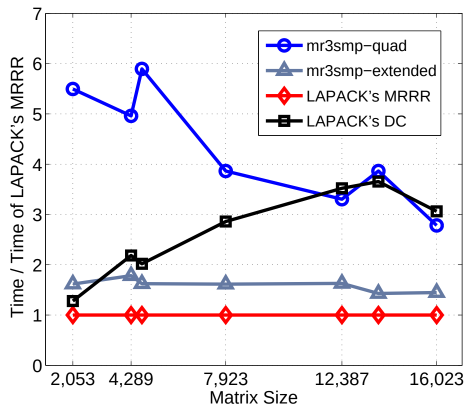
		A	*B*	*C*	*D*	*E*	*F*	*G*
max. depth	`DSTEMR`	2	2	2	4	5	2	5
	`mr3smp-quad`	0	0	0	0	0	0	0
largest cluster	`DSTEMR`	324	1,027	10	5,011	8,871	1,369	14,647
	`mr3smp-quad`	1	1	1	1	1	1	1
new RRR	`DSTEMR`	311	638	1,043	1,089	1,487	1,798	1,825
	`mr3smp-quad`	0	0	0	0	0	0	0

Table 3: Recursion depth, largest encountered cluster, and number of times an RRR for a cluster needs to be computed by executing Line 17 in Algorithm 1 for `DSTEMR` and `mr3smp-quad`.

Such a danger occurs in Line 17, where a new RRR for each cluster needs to be computed. By executing Line 17 only a few times – often no times at all – the danger of not finding a proper RRR is reduced and therefore robustness increased.[13] Since our approach is independent of the actual form of the RRRs, it is possible to additionally use blocked factorizations as proposed in [57] to further improve robustness.

The mixed precision approach is especially appealing in the context of distributed-memory parallelism. The fact that all eigenpairs in our experiment are computed directly from the initial representation implies that the execution is truly *embarrassingly parallel*. That MRRR is embarrassingly parallel was already announced – somewhat prematurely – with its introduction, see [18]. Only later, parallel versions of MRRR [7, 54] found that "the eigenvector computation in MRRR is only embarrassingly parallel if the root representation consists of singletons" [53], i.e., all the eigenvectors are computed from the initial RRR, and that otherwise "load imbalance can occur and hamper the overall performance" [54].

[13]Besides the fact that less RRRs need to be found, additionally, the restriction of what constitutes an RRR might be relaxed.



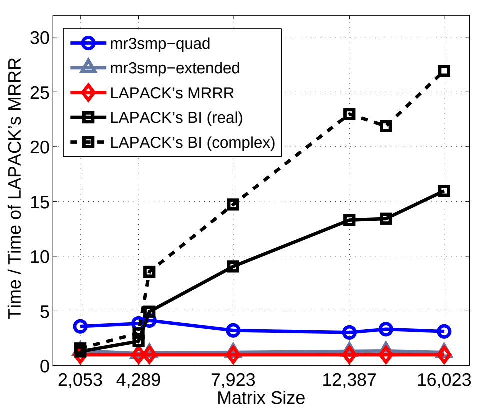


Figure 5: Execution time of the tridiagonal stage relative to LAPACK's MRRR. *Left:* Computation of all eigenpairs. *Right:* Computation of 20% of the eigenpairs corresponding to the smallest eigenvalues.

While one can expect only very limited clustering of eigenvalues for application matrices arising from dense inputs, it is not always the case that the recursion depth is zero. Experiments on all the tridiagonal matrices provided explicitly by the STETESTER [39] – a total of 176 matrices ranging in size from 3 to 24,873 – showed that the worst case residual norm and worst case orthogonality were given by respectively $1.5 \cdot 10^{-14}$ and $1.2 \cdot 10^{-15}$ and the recursion depth was limited to two for all matrices. In fact, only four artificially constructed matrices, glued Wilkinson matrices [22], which are challenging for the MRRR algorithm, had clusters within clusters. In most cases, with the settings of our experiments, the clustering was very benign or even no clustering was observed. For example, the largest matrix in the test set, *Bennighof_24873*, had only a single cluster of size 37. Furthermore, it is also possible to significantly lower the *gaptol* parameter, say to 10^{-16}, and therefore reduce clustering even more. For such small values, in the approximation and refinement of eigenvalues we need to resort to quadruple precision, which so far we avoided for performance reasons, see Section 3.

Our results suggest that all experimental results hold similarly for the multi-threaded and the distributed-memory case. The MRRR algorithm was already highly scalable, see [47, 48, 7, 54], and the mixed precision approach additionally improves scalability – often making the computation truly embarrassingly parallel.

Computing a subset of eigenpairs. The situation is more favorable when only a subset of eigenpairs needs to be computed. As `DSYEVD` does not allow subset computations at reduced cost, a user can resort to either BI or MRRR. The capabilities of BI are accessible via LAPACK's routine `DSYEVX`. Recently, the routine `DSYEVR` was edited, so that it uses BI instead of MRRR in the subset case. We therefore refer to `'DSYEVR (BI)'` when we use BI and `'DSYEVR (MRRR)'` when we force the use of

MRRR instead.[14] In Fig. 6, we report the execution time relative to LAPACK's MRRR for computing 20% of the eigenpairs associated with the smallest eigenvalues and the corresponding orthogonality.

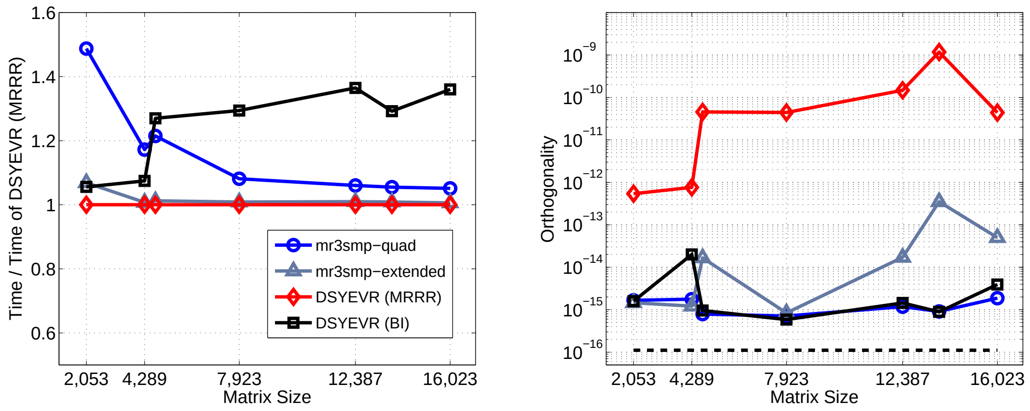


Figure 6: Computation of 20% of the eigenpairs corresponding to the smallest eigenvalues. *Left:* Execution time relative to routine `DSYEVR` that is forced to use MRRR. *Right:* Orthogonality.

BI and `mr3smp-quad` achieved the best orthogonality. At the moment, for small matrices ($n \ll 2{,}000$) the use quadruple might be be too expensive, but in this case the mixed precision routine can easily be run in double precision only or BI can be used for accuracy and performance. As support for quadruple precision improves, the overhead will further decrease or completely vanish. The use of extended precision comes almost without any performance penalty. On the other hand, for larger matrices, the orthogonality might still be inferior to other methods. To illustrate the source of the differing run times, the right panel of Fig. 5 presents the execution time of the tridiagonal eigensolver relative to LAPACK's MRRR. As expected, due to explicit orthogonalization via the Gram-Schmidt procedure, BI potentially becomes considerably slower than MRRR.

4.2 Complex Hermitian case

Computing all eigenpairs. In Fig. 7, we show results for computing all eigenpairs of complex valued Hermitian matrices. The left and right panel display the execution time of all solvers relative to LAPACK's MRRR (`ZHEEVR`) and the orthogonality, respectively. As predicted, the extra cost due to the higher precision becomes relatively smaller for complex valued input compared to real valued input – compare Figs. 4 and 7. Similarly, if the mixed precision solver is used for the generalized eigenproblem based on Cholesky-Wilkinson algorithm [40, 43], the approach will increase the execution only marginally even for relatively small problems.

[14]In all experiments, we used BI with default parameters.

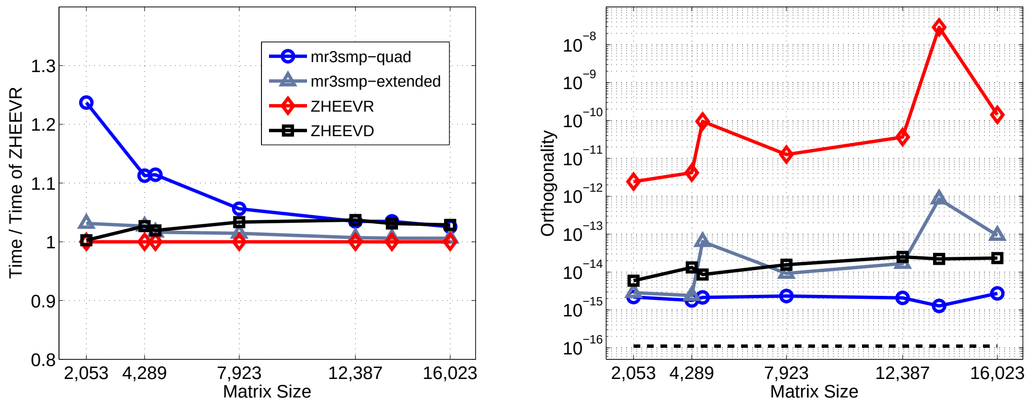


Figure 7: Computation of all eigenpairs. *Left:* Execution time relative to routine ZHEEVR. *Right:* Orthogonality.

We want to remark that the timing plots might be misleading and suggest that the cost of the mixed precision approach is high. Indeed, for test matrix A, using quadruple precision increased the run time by about 23% relative to ZHEEVR. This means in absolute time that the mixed precision approach required about 27 seconds and ZHEEVR only 22 seconds. For larger matrices the absolute execution time increases as n^3 and the performance gap between mixed precision approach and pure double precision solver vanishes. Such a scenario is observed with test matrix F, for which we obtain an orthogonality of $1.3 \cdot 10^{-15}$ with mr3smp-quad compared to $2.9 \cdot 10^{-8}$ with ZHEEVR, while spending only about 4% more in the total execution time.

Computing a subset of eigenpairs. As in the real valued case, we compute 20% of the eigenpairs associated with the smallest eigenvalues. The execution time relative to LAPACK's MRRR and the corresponding orthogonality are displayed in Fig. 8.

For our test case, the extra cost due to the use of higher precision in mr3smp-extended or mr3smp-quad is quite small. Even if quadruple precision was used, for the smallest matrix the run time only increased by 13%. In a similar experiment – computing 10% of the eigenpairs corresponding to the smallest eigenvalues – the extra cost for mr3smp-quad was less than 6% compared to LAPACK's MRRR. We believe that this additional cost could be reduced further, for instance by omitting the initial approximation of the eigenvalues of T using quadruple arithmetic – see Line 2 of Algorithm 1. On the other hand, such a penalty in the execution time is already below the fluctuations observed in repeated experiments. Currently, mr3smp-extended is faster than mr3smp-quad for smaller problems, but using extended precision cannot quite deliver the same orthogonality as obtained with quadruple.

The relative timings of the tridiagonal eigensolvers are depicted in the right panel of Fig. 5. Interestingly, BI is almost by a factor two slower than in the real valued case, although exactly the same computation is performed. The reason is that the Gram-

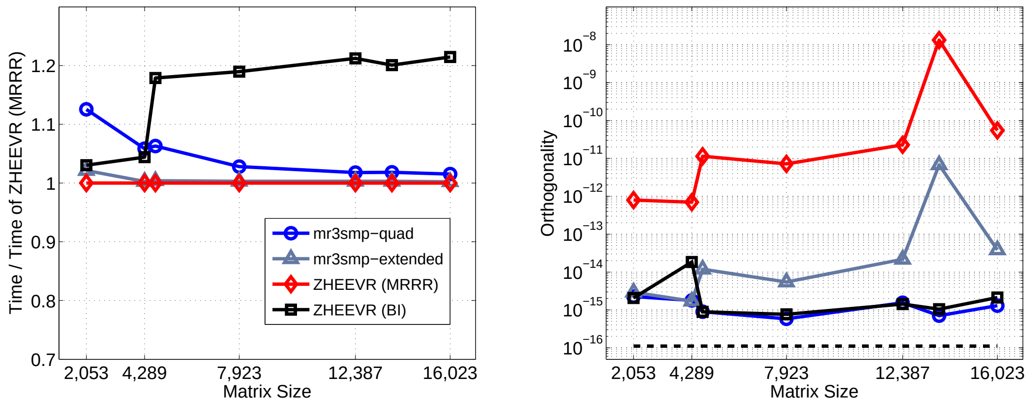


Figure 8: Computation of 20% of the eigenpairs corresponding to the smallest eigenvalues are computed. *Left:* Execution time relative to routine `ZHEEVR` (when forced to use MRRR). *Right:* Orthogonality.

Schmidt orthogonalization, a memory bandwidth-limited operation, is performed on complex data (although all imaginary parts of the involved vectors are zero).

4.3 Summarizing the experimental results

Using our mixed precision approach improves the orthogonality considerably – possibly at the cost of some performance. The additional cost depend on the difference in speed between the precision of input/output and the higher precision used in the tridiagonal stage. As demonstrated in the single/double precision case, the mixed precision approach might lead to faster executions. In any case, for larger matrices the additional cost becomes negligible as the tridiagonal stage has a lower complexity than the other two stages of the Hermitian eigenproblem. For double precision input/output and small matrices, the use of quadruple precision comes with high extra cost, as currently the support for quadruple precision is very limited and the arithmetic is rather slow as it is performed entirely in software. In the future, with improved support for quadruple – through (partial) hardware support or advances in the algorithms for software simulation – the additional cost of the mixed precision approach vanishes. An alternative on many architectures today is the use of a hardware supported 80-bit extended floating point format. In this case, the execution time is hardly affected, but it cannot guarantee the same orthogonality as quadruple. In addition to improving the orthogonality, our approach increases both robustness and scalability of the solver. For this reasons, the mixed precision approach is ideal for large-scale distributed-memory solvers such as [47, 48, 7, 54].

5 Conclusions

In order to achieve improvements in accuracy, robustness, and scalability of MRRR-based eigensolvers, we take on a different perspective of the tridiagonal MRRR algorithm. In our perspective, given a format *binary_x* for the input/output arguments, a *binary_y* floating point arithmetic is used to obtain any desired (achievable) accuracy. In particular, provided the precision of the y-bit floating point format is sufficiently smaller than the precision of the x-bit format, we obtain eigenvectors that are truly orthogonal to input/output precision. We showed through cases studies that the performance, robustness, and scalability of a tridiagonal eigensolver that incorporates our mixed precision approach can be improved by relaxing its accuracy requirements. The applicability of the approach depends mainly on the difference in performance between the x-bit and y-bit floating point arithmetics. This is illustrated by two cases with respectively a small and a large gap in performance: (1) *binary32* (single) input/output with *binary64* (double) used internally and (2) *binary64* input/output with *binary128* (quadruple) used internally.

For single precision input/output arguments, we obtain routines for dense Hermitian eigenproblems that are more accurate *and* faster, more robust, and more scalable than the corresponding single precision eigensolver. Additionally, our mixed precision approach is portable and has memory requirements similar to the original. In the single precision case, the approach has *no major drawback* and works well for all matrix sizes, whether all eigenpairs are computed or just a subset of them.

For double precision input/output arguments, we can resort to either extended or quadruple precision. The first option offers a somewhat improved accuracy without major performance cost. The latter option provides all the benefits mentioned in the single precision case, but might slow down the computation due to today's limited support for quadruple precision. Nonetheless, for large matrices, the extra cost is small even with today's software simulated arithmetic. This is even more when only a small subset of eigenpairs is computed and when the routines are executed in parallel. Furthermore, if the support for quadruple precision improves in the future, the mixed precision approach will – like for single precision input/output – provide higher accuracy without any major performance penalty.

As a result, we obtain MRRR-based eigensolvers for the Hermitian eigenproblem that are at least as accurate as other methods like the Divide-and-Conquer or the QR algorithm while largely maintaining or even improving the strengths of MRRR: speed and scalability.

Acknowledgments

The authors would like to thank Diego Fabregat for discussion on an earlier draft of this report. Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 is gratefully acknowledged. Enrique S. Quintana-Ortí was supported by project TIN2011-23283 and FEDER.

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers. *ANSI/IEEE 754-1985*. 1985.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. W. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[3] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P. Willems. Parallel Solution of Partial Symmetric Eigenvalue Problems from Electronic Structure Calculations. *Parallel Comput.*, 37:783–794, Dec. 2011.

[4] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov. Accelerating Scientific Computations with Mixed Precision Algorithms. *Computer Physics Communications*, 180(12):2526 – 2533, 2009.

[5] W. Barth, R. S. Martin, and J. H. Wilkinson. Calculation of the Eigenvalues of a Symmetric Tridiagonal Matrix by the Method of Bisection. *Numer. Math.*, V9(5):386–393, 1967.

[6] P. Benner, P. Ezzatti, D. Kressner, E. S. Quintana-Ortí, and A. Remón. A Mixed-Precision Algorithm for the Solution of Lyapunov Equations on Hybrid CPU-GPU Platforms. *Parallel Comput.*, 37(8):439–450, Aug. 2011.

[7] P. Bientinesi, I. Dhillon, and R. van de Geijn. A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations. *SIAM J. Sci. Comput.*, 27:43–66, July 2005.

[8] Å. Björck. Iterative Refinement and Reliable Computing. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, pages 249–266, Oxford, 1990. Clarendon Press.

[9] S. Blügel, G. Bihlmeyer, D. Wortmann, C. Friedrich, M. Heide, M. Lezaic, F. Freimuth, and M. Betzinger. *The Jülich FLEUR Project*. Jülich Research Center, 1987. `http://www.flapw.de`.

[10] J. Cuppen. A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem. *Numer. Math.*, 36:177–195, 1981.

[11] C. Davis and W. Kahan. The Rotation of Eigenvectors by a Perturbation. III. *SIAM J. Numer. Anal.*, 7(1):pp. 1–46, 1970.

[12] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, PA, USA, 1997.

[13] J. W. Demmel, I. Dhillon, and H. Ren. On the Correctness of some Bisection-like Parallel Eigenvalue Algorithms in Floating Point Arithmetic. *Electronic Trans. Num. Anal.*, 3:116–149, 1995.

[14] J. W. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide*. SIAM, Philadelphia, PA, USA, 2000.

[15] J. W. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy. Error Bounds from Extra-Precise Iterative Refinement. *ACM Trans. Math. Softw.*, 32(2):325–351, June 2006.

[16] J. W. Demmel, O. Marques, B. Parlett, and C. Vömel. Performance and Accuracy of LAPACK's Symmetric Tridiagonal Eigensolvers. *SIAM J. Sci. Comp.*, 30:1508–1526, 2008.

[17] J. W. Demmel and K. Veselic. Jacobi's Method is more accurate than QR. *SIAM J. Matrix Anal. Appl*, 13:1204–1245, 1992.

[18] I. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, EECS Department, University of California, Berkeley, 1997.

[19] I. Dhillon. Current Inverse Iteration Software Can Fail. *BIT*, 38:685–704, 1998.

[20] I. Dhillon and B. Parlett. Multiple Representations to Compute Orthogonal Eigenvectors of Symmetric Tridiagonal Matrices. *Linear Algebra Appl.*, 387:1–28, 2004.

[21] I. Dhillon and B. Parlett. Orthogonal Eigenvectors and Relative Gaps. *SIAM J. Matrix Anal. Appl.*, 25:858–899, 2004.

[22] I. Dhillon, B. Parlett, and C. Vömel. Glued Matrices and the MRRR Algorithm. *SIAM J. Sci. Comput*, 27:496–510, 2005.

[23] I. Dhillon, B. Parlett, and C. Vömel. The Design and Implementation of the MRRR Algorithm. *ACM Trans. Math. Software*, 32:533–560, December 2006.

[24] J. Dongarra, J. Du Cruz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 16:1–17, 1990.

[25] J. Dongarra and D. C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 8(2):139–154, Mar. 1987.

[26] G. Fann, R. Littlefield, and D. Elwood. Performance of a Fully Parallel Dense Real Symmetric Eigensolver in Quantum Chemistry Applications. In *Proceedings of High Performance Computing '95*, Simulation MultiConference. The Society for Computer Simulation, 1995.

[27] J. Francis. The QR Transform - A Unitary Analogue to the LR Transformation, Part I and II. *The Comp. J.*, 4, 1961/1962.

[28] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[29] M. Gu and S. C. Eisenstat. A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem. *SIAM J. Matrix Anal. Appl.*, 16(1):172–191, 1995.

[30] N. Higham. Iterative Refinement for Linear Systems and LAPACK. *IMA Journal of Numerical Analysis*, 17(4):495–509, 1997.

[31] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, 2008.

[32] C. G. J. Jacobi. Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen. 30:51–94, 1846.

[33] W. Kahan. Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic. Manuscript, 1997.

[34] D. Kressner. *Numerical Methods for General And Structured Eigenvalue Problems*. Springer, Berlin Heidelberg New York, 2005.

[35] V. Kublanovskaya. On some Algorithms for the Solution of the Complete Eigenvalue Problem. *Zh. Vych. Mat.*, 1:555–572, 1961.

[36] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, A. Kapur, M. C. Martin, T. Tung, and D. J. Yoo. Design, Implementation and Testing of Extended and Mixed Precision BLAS. *ACM Trans. Math. Software*, 28:2002, 2002.

[37] O. Marques, B. Parlett, and C. Vömel. Computations of Eigenpair Subsets with the MRRR Algorithm. *Numerical Linear Algebra with Applications*, 13(8):643–653, 2006.

[38] O. Marques, E. Riedy, and C. Vömel. Benefits of IEEE-754 Features in Modern Symmetric Tridiagonal Eigensolvers. *SIAM J. Sci. Comput.*, 28:1613–1633, September 2006.

[39] O. A. Marques, C. Vömel, J. W. Demmel, and B. N. Parlett. Algorithm 880: A Testing Infrastructure for Symmetric Tridiagonal Eigensolvers. *ACM Trans. Math. Softw.*, 35(1):8:1–8:13, July 2008.

[40] R. Martin and J. Wilkinson. Reduction of the Symmetric Eigenproblem $Ax = \lambda Bx$; and Related Problems to Standard Form. *Numerische Mathematik*, 11:99–110, 1968. 10.1007/BF02165306.

[41] C. B. Moler. Iterative Refinement in Floating Point. *J. ACM*, 14(2):316–321, Apr. 1967.

[42] M. Nakata. *The MPACK (MBLAS / MLAPACK): A Multiple Precision Arithmetic Version of BLAS and LAPACK*, 2012. `http://mplapack.sourceforge.net`.

[43] B. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[44] B. Parlett and I. Dhillon. Fernando's Solution to Wilkinson's Problem: an Application of Double Factorization. *Linear Algebra Appl.*, 267:247–279, 1996.

[45] B. Parlett and I. Dhillon. Relatively Robust Representations of Symmetric Tridiagonals. *Linear Algebra Appl.*, 309(1-3):121 – 151, 2000.

[46] B. Parlett and O. Marques. An Implementation of the DQDS Algorithm (Positive Case). *Linear Algebra Appl.*, 309:217–259, 1999.

[47] M. Petschow and P. Bientinesi. MR^3-SMP: A Symmetric Tridiagonal Eigensolver for Multi-Core Architectures. *Parallel Computing*, 37(12):795 – 805, 2011.

[48] M. Petschow, E. Peise, and P. Bientinesi. High-Performance Solvers For Dense Hermitian Eigenproblems. *Submitted*, 2011.

[49] A. Sameh. On Jacobi and Jacobi-like Algorithms for a Parallel Computer. *Math. Comp.*, 25(115):579–590, 1971.

[50] M. Sears, K. Stanley, and G. Henry. Application of a High Performance Parallel Eigensolver to Electronic Structure Calculations. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '98, pages 1–1, Washington, DC, USA, 1998. IEEE Computer Society.

[51] W. Stein et al. *Sage Mathematics Software.* The Sage Development Team, 2012. `http://www.sagemath.org`.

[52] F. G. Van Zee, R. van de Geijn, and G. Quintana-Orti. Restructuring the QR Algorithm for High-Performance Application of Givens Rotations. Technical Report TR-11-36, The University of Texas at Austin, Department of Computer Sciences, October 2011.

[53] C. Vömel. A Refined Representation Tree for MRRR. LAPACK Working Note 194, Department of Computer Science, University of Tennessee, Knoxville, Nov. 2007.

[54] C. Vömel. ScaLAPACK's MRRR Algorithm. *ACM Trans. Math. Software*, 37:1:1–1:35, January 2010.

[55] J. H. Wilkinson. The Calculation of the Eigenvectors of Codiagonal Matrices. *Comp. J.*, 1(2):90–96, 1958.

[56] P. Willems. *On MR^3-type Algorithms for the Tridiagonal Symmetric Eigenproblem and Bidiagonal SVD.* PhD thesis, University of Wuppertal, 2010.

[57] P. Willems and B. Lang. Block Factorizations and qd-type Transformations for the MR^3 Algorithm. *Electron. Trans. Numer. Anal.*, 38:363–400, 2011.

[58] S. Wolfram. *Mathematica - a system for doing mathematics by computer (2. ed.).* Addison-Wesley, 1992.