Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Comparison of eigensolvers for symmetric band matrices

Michael Moldaschl, Wilfried N. Gansterer [*]

University of Vienna, Faculty of Computer Science, Währingerstrasse 29, Vienna, Austria

HIGHLIGHTS

- Single core performance of symmetric band eigensolvers is compared experimentally.
- Tridiagonalizing a band matrix produces subnormal numbers and degrades performance.
- Methods without tridiagonalizing the full band matrix have performance advantages.
- For clustered eigenvalues the block divide-and-conquer (BD&C) method is fastest.
- For separated eigenvalues and narrow bands the BTF method is fastest.

ARTICLE INFO

Article history:
Received 31 October 2012
Received in revised form 7 January 2014
Accepted 9 January 2014
Available online 31 January 2014

Keywords:
Symmetric banded eigenvalue problem
Tridiagonalization
Subnormal numbers
Denormalized numbers

ABSTRACT

We compare different algorithms for computing eigenvalues and eigenvectors of a symmetric band matrix across a wide range of synthetic test problems. Of particular interest is a comparison of state-of-the-art tridiagonalization-based methods as implemented in LAPACK or PLASMA on the one hand, and the block divide-and-conquer (BD&C) algorithm as well as the block twisted factorization (BTF) method on the other hand. The BD&C algorithm does not require tridiagonalization of the original band matrix at all, and the current version of the BTF method tridiagonalizes the original band matrix only for computing the eigenvalues.

Avoiding the tridiagonalization process sidesteps the cost of backtransformation of the eigenvectors. Beyond that, we discovered another disadvantage of the backtransformation process for band matrices: In several scenarios, a lot of gradual underflow is observed in the (optional) accumulation of the transformation matrix and in the (obligatory) backtransformation step. According to the IEEE 754 standard for floating-point arithmetic, this implies many operations with subnormal (denormalized) numbers, which causes severe slowdowns compared to the other algorithms without backtransformation of the eigenvectors. We illustrate that in these cases the performance of existing methods from LAPACK and PLASMA reaches a competitive level only if subnormal numbers are disabled (and thus the IEEE standard is violated).

Overall, our performance studies illustrate that if the problem size is large enough relative to the bandwidth, BD&C tends to achieve the highest performance of all methods if the spectrum to be computed is clustered. For test problems with well separated eigenvalues, the BTF method tends to become the fastest algorithm with growing problem size.

[*] Corresponding author.
E-mail addresses: michael.moldaschl@univie.ac.at (M. Moldaschl), wilfried.gansterer@univie.ac.at (W.N. Gansterer).

http://dx.doi.org/10.1016/j.scico.2014.01.005

1. Introduction

State-of-the-art eigensolvers for dense symmetric matrices transform the original matrix to symmetric tridiagonal form in a preprocessing step. The tridiagonal problem can then be solved by any standard tridiagonal eigensolver. This tridiagonalization process and the corresponding backtransformation process of the eigenvectors of the tridiagonal representation are dominating the computational cost of the entire solver. On multi- and many-core architectures it turns out to be advantageous to perform the tridiagonalization as a *two-step band reduction* process with a symmetric band matrix as intermediate step [1,2]. Although the tridiagonalization of a band matrix cannot be completely based on BLAS 3 operations, it can be parallelized efficiently for a certain range of bandwidths. In the current version 2.6.0 of PLASMA, this reduction from the intermediate band matrix to tridiagonal form is done using such a "bulge-chasing" technique [1,2]. The tridiagonalization process has several known disadvantages. Most importantly, every band reduction step causes a costly backtransformation for eigenvectors.

These known disadvantages of the tridiagonalization process have motivated the development of algorithms for computing eigenpairs of a symmetric band matrix *without* tridiagonalizing it as a whole, such as the block divide-and-conquer (BD&C) method [3,4].

In this paper, we perform a comprehensive comparison of five different basic approaches for computing eigenpairs of a given symmetric band matrix. From now on, we will denote methods or routines from LAPACK [5] or composed of building blocks from LAPACK with the prefix "L/", and methods or routines composed of building blocks from PLASMA [6] with the prefix "P/". More specifically, we compare

- L/DSBEVD from LAPACK, which first tridiagonalizes the given symmetric band matrix, then applies the tridiagonal divide-and-conquer method [7,8], and finally transforms the eigenvectors back;
- L/DSBEVR, which also first tridiagonalizes the given symmetric band matrix, then applies the MRRR algorithm [9], and finally also transforms the eigenvectors back;
- P/DSBEVD, which consists of the same basic algorithmic steps as L/DSBEVD, but uses PLASMA routines as building blocks;
- the block divide-and-conquer (BD&C) method [3,4], which directly computes eigenvalues and eigenvectors of a symmetric band matrix without tridiagonalizing it as a whole; and
- the BTF method [10], which tridiagonalizes the given symmetric band matrix only for computing the eigenvalues, but computes the eigenvectors directly from the band matrix and thus avoids constructing or applying the corresponding similarity transformation.

1.1. Existing work

LAPACK contains the following methods for solving the symmetric banded eigenvalue problem: L/xSBEV, L/xSBEVX and L/xSBEVD. In all three methods the computation is split into three steps. First, the symmetric band matrix is tridiagonalized using L/xSBTRD. Then, the eigenpairs of the symmetric tridiagonal eigenvalue problem are computed – either using L/xSTEDC (tridiagonal divide-and-conquer algorithm) in L/xSBEVD or using L/xSTEQR (QR algorithm) in L/xSBEV and L/xSBEVX. Finally, in all three routines the eigenvectors of the tridiagonal matrix are multiplied with the orthogonal transformation from L/xSBTRD. L/xSBTRD is based on Givens rotations (plane rotations) which are also used to eliminate any fill-in outside the band during the computation.

Another routine in LAPACK for symmetric *tridiagonal* eigenproblems, L/xSTEMR (or the usually used wrapper routine L/xSTEGR), is based on *relatively robust representations* [9] for computing selected eigenpairs. This method is used in the driver routine L/xSYEVR for symmetric full matrices, but no driver routine exists in LAPACK for symmetric band matrices. In [11], all symmetric tridiagonal eigensolvers available in LAPACK are compared in terms of numerical accuracy and runtime, and it is shown that xSTEDC and xSTEMR are the fastest methods. Thus, we focus on these two methods in this paper.

It turns out that single step tridiagonalization of a symmetric full matrix cannot be implemented efficiently on multicore systems. It is much better to perform tridiagonalization in *two* band reduction steps – first transforming the full matrix into a symmetric band matrix and then tridiagonalizing the band matrix [2]. This approach is implemented in the function P/xSYTRD of the PLASMA library. However, currently there is no eigensolver routine for symmetric band matrices in PLASMA.

Besides the methods already available in widespread numerical linear algebra libraries, there are some recent algorithmic developments which mostly operate on the symmetric band matrix directly and do not or only partly require a tridiagonalization process. In particular, this comprises the *block divide-and-conquer* (*BD&C*) algorithm [3,4] and the *block twisted factorization* (*BTF*) method. The BD&C algorithm is a generalization of the tridiagonal divide-and-conquer method for symmetric block tridiagonal matrices. Every band matrix can be represented as a block tridiagonal matrix with upper and lower triangular off-diagonal blocks. The BTF method [10] tridiagonalizes the band matrix only for computing its eigenvalues, and it factorizes the shifted band (or block tridiagonal) matrix in order to compute its eigenvectors by an inverse iteration process. In [10] is has been shown how a very good starting vector can be determined from the block twisted factorizations of the shifted band matrix, and in general only very few iterations are needed for an accurate eigenvector approximation.

1.2. Synopsis

In Section 2, we provide a detailed review of the methods compared in this paper. Section 3 summarizes the experimental setup used in our runtime comparisons, including the types of test matrices generated. Section 4 presents the results of a first set of experiments in terms of runtimes as well as numerical accuracy achieved. These experimental results are analyzed and explained in Section 5, where we also illustrate the influence of subnormal (denormalized) numbers on runtime and numerical accuracy of the backtransformation of the eigenvectors. We then repeat the experimental comparisons of the different methods when subnormal numbers are *disabled*. Finally, Section 6 concludes the paper.

2. Methods

Five different methods and their implementations are compared in this paper: (i) the LAPACK routine L/DSBEVD, (ii) the routine L/DSBEVR which is based on the MRRR algorithm [9] and composed of building blocks from LAPACK, (iii) the routine P/DSBEVD which is an adaptation of the PLASMA routine P/DSYEVD for band matrices and composed of building blocks from PLASMA, (iv) the BD&C algorithm [3,4], and (v) the BTF method [10]. In the following sections, these five methods and their theoretical properties are briefly reviewed.

2.1. L/DSBEVD and L/DSBEVR

While L/DSBEVD is an existing LAPACK routine for computing the eigenvalues and eigenvectors of a symmetric band matrix, we put together the routine L/DSBEVR using building blocks from LAPACK and BLAS. As a first step, both routines transform the symmetric band matrix into a symmetric tridiagonal matrix using the LAPACK routine L/DSBTRD. This transformation is based on Givens rotations and ensures that the computation remains within the band. Then, L/DSBEVD uses the LAPACK routine L/DSTEDC (tridiagonal divide-and-conquer algorithm) for solving the tridiagonal eigenproblem, whereas L/DSBEVR uses the LAPACK routine L/DSTEMR (multiple relatively robust representations). Finally, in both methods the eigenvectors are transformed by multiplying them with the transformation matrix.

2.2. P/DSBEVD

This new routine is based on building blocks from PLASMA. In particular, the existing routine P/DSYEVD is modified in two aspects: First, the new routine P/DSBEVD skips the full-to-band reduction used in P/DSYEVD, but only tridiagonalizes the symmetric banded input matrix using the bulge chasing technique [1,2] available in PLASMA. Second, only a single backtransformation has to be applied to the computed eigenvectors of the tridiagonal eigenproblem (because of the skipped full-to-band reduction). Like P/DSYEVD and L/DSBEVD, P/DSBEVD uses the routine L/DSTEDC for solving the tridiagonal eigenproblem.

2.3. Block divide-and-conquer (BD&C) algorithm

The BD&C algorithm [3,4] computes all n eigenvalues λ_i and eigenvectors v_i of a symmetric block tridiagonal (banded) matrix M_p, defined as

$$M_p = \begin{pmatrix} B_1 & C_1^\top & & & \\ C_1 & B_2 & C_2^\top & & \\ & C_2 & B_3 & \ddots & \\ & & \ddots & \ddots & C_{p-1}^\top \\ & & & C_{p-1} & B_p \end{pmatrix} \in \mathbb{R}^{n\times n}. \tag{1}$$

Here, $B_i \in \mathbb{R}^{b_i\times b_i}$, $C_i \in \mathbb{R}^{b_{i+1}\times b_i}$, $B_i = B_i^\top$, $n = \sum_{i=1}^{p} b_i$, $rank(C_i) = l_i \leqslant \min(b_i, b_{i+1})$, and $l = \sum_{i=1}^{p-1} l_i$. In the following, we consider the special case $b_i = b$ and C_i being an upper triangular matrix $\forall i$, i.e., a symmetric band matrix with semibandwidth b.

Three algorithmic steps can be distinguished in BD&C: *subdivision*, *solution of the independent subproblems* and *synthesis of the solutions of the subproblems*. In the *subdivision* step the singular value decompositions (SVD) of all off-diagonal blocks $C_i = \sum_{j=1}^{l_i} \sigma_j^i u_j^i (v_j^i)^\top = U_i \Sigma_i V_i^\top$, $1 \leqslant i < p$, are computed. Then, based on these SVDs, new diagonal blocks $\tilde{B}_i$ are computed:

$$\tilde{B}_i := \begin{cases} B_1 - V_1\Sigma_1 V_1^\top & i = 1, \\ B_i - U_{i-1}\Sigma_{i-1}U_{i-1}^\top - V_i\Sigma_i V_i^\top & 2 \leqslant i \leqslant p-1, \\ B_p - U_{p-1}\Sigma_{p-1}U_{p-1}^\top & i = p, \end{cases} \tag{2}$$

and the corresponding rank-one updates w_j can be constructed as rows of block updates W_i:

$$W_i^\top = \begin{cases} \left((V_1\Sigma_1^{1/2})^\top \quad (U_1\Sigma_1^{1/2})^\top \quad 0 \quad 0 \right) & i = 1, \\ \left(0 \quad (V_i\Sigma_i^{1/2})^\top \quad (U_i\Sigma_i^{1/2})^\top \quad 0 \right) & 2 \leqslant i \leqslant p-2, \\ \left(0 \quad 0 \quad (V_{p-1}\Sigma_{p-1}^{1/2})^\top \quad (U_{p-1}\Sigma_{p-1}^{1/2})^\top \right) & i = p-1. \end{cases} \tag{3}$$

In summary, these updates can be used to represent the original block tridiagonal matrix M_p as a sequence of l rank-one modifications of the block diagonal matrix $\tilde{M}_p = \operatorname{diag}(\tilde{B}_1, \ldots, \tilde{B}_p)$:

$$M_p = \tilde{M}_p + \sum_{j=1}^{l} w_j w_j^\top. \tag{4}$$

Each of the w_j “connects” two neighboring blocks and has non-zero entries only in the corresponding region, which is exploited in the implementation.

In the next step (*solution of subproblems*), the complete eigenvalue decomposition $\tilde{M}_p = QDQ^\top$ is computed, which corresponds simply to the independent eigenvalue decompositions of all $\tilde{B}_i$. The *synthesis problem* is then represented by

$$M_p = Q\left(D + \sum_{j=1}^{l} y_j y_j^\top\right)Q^\top \quad \text{with } y_j = Q^\top w_j. \tag{5}$$

Thus, the original symmetric block tridiagonal (banded) eigenproblem has been transformed to a rank-l modification problem of a diagonal matrix. This rank-l modification problem can be solved as a sequence of l rank-one modification problems like in the divide-and-conquer algorithm implemented in the LAPACK routine L/DSTEDC. This also includes the process of *deflation*, which is very important in terms of efficiency because it tends to significantly reduce the cost of the performance dominating part of the computation [4]. Note that no tridiagonalization of the entire band matrix is required.

2.4. The block twisted factorization (BTF) method

The BTF method is based on inverse iteration for independently computing eigenvectors using previously computed accurate approximations of the corresponding eigenvalues. “Accurate” in this context means that the approximation is closer to the wanted eigenvalue than to any other. This can be difficult for clustered eigenvalues. In [10] it is illustrated that clustered eigenvalues may result in a loss of orthogonality of the computed eigenvectors. In the implementation of the BTF method considered in this paper, the eigenvalues are computed by the routine L/DSBEVD in the configuration “eigenvalues only” (parameter JOBZ = “N”).

In the next step of the BTF method, the matrix is shifted by the approximation of the eigenvalue λ_j and then inverse iteration is used to compute an approximation of the corresponding eigenvector v_j:

$$(M_p - \lambda_j I)x_{i+1} = W x_{i+1} = x_i. \tag{6}$$

Two ingredients are necessary for this iteration: a starting vector x_0 has to be constructed which enables fast convergence of the process to the eigenvector, and the matrix $W := M_p - \lambda_j I$ has to be factorized for solving the linear system in each iteration. In the BTF method a good starting vector x_0 is constructed based on information from *block twisted factorizations* of W [10], and only very few iterations are needed for obtaining a very accurate result. The factorization

$$W = P \begin{pmatrix} L_1^+ & & & & & & \\ M_2^+ & \ddots & & & & & \\ & \ddots & L_{i-1}^+ & & & & \\ & & M_i^+ & L_i & M_{i+1}^- & & \\ & & & & L_{i+1}^- & \ddots & \\ & & & & & \ddots & M_p^- \\ & & & & & & L_p^- \end{pmatrix} \begin{pmatrix} U_1^+ & N_1^+ & & & & & \\ & \ddots & \ddots & & & & \\ & & U_{i-1}^+ & N_{i-1}^+ & & & \\ & & & U_i & & & \\ & & & N_i^- & U_{i+1}^- & & \\ & & & & \ddots & \ddots & \\ & & & & & N_{p-1}^- & U_p^- \end{pmatrix} \tag{7}$$

is called *block twisted factorization*, because it is a combination of a block forward and a block backward factorization. In total, p different block twisted factorizations exist, depending on in which block the forward and backward factorization meet.[1] For solving Eq. (6) only one factorization would be necessary, but the information of all block twisted factorizations

[1] The + or − superscript in Eq. (7) defines whether it is a part of the forward or of the backward factorization.

is used to define a good starting vector x_0. Basically, the complete forward and the complete backward factorization need to be computed to then get all twisted factorizations by updating the diagonal blocks and factorizing them:

$$B_i - P_i^+ M_i^+ N_{i-1}^+ - P_{i+1}^- M_{i+1}^- N_i^- = P_i L_i U_i \quad \forall i = 1, 2, \dots, p. \tag{8}$$

As motivated by theoretical considerations in [10], the position of the minimal diagonal entry over all U_i can be used to define the position of the only non-zero entry in the starting vector x_0. This strategy is called *minsca* in [10] and experiments showed that very high accuracy can be reached in many cases within a single iteration. As illustrated in Sections 4 and 5.2, two iterations yield a satisfactory residual in all cases.

2.5. Theoretical comparison of the methods

The five methods described above apply different concepts for solving the symmetric banded eigenvalue problem. The first three methods, L/DSBEVD, L/DSBEVR and P/DSBEVD, tridiagonalize the symmetric $n \times n$ band matrix M_p as a whole. Then, one of the available numerically reliable and very fast tridiagonal eigensolvers can be applied. However, as a consequence of this transformation, the eigenvectors of the tridiagonal matrix need to be transformed back via (explicit or implicit) multiplication with the $n \times n$ transformation matrix.

We would normally expect only a negligible influence of the numerical properties of the input data (distances between eigenvalues, condition, etc.), on the overall runtime performance of these three routines. Although the runtime of the tridiagonal eigensolver is influenced by these numerical properties, this step does not dominate the overall performance. The computational cost is dominated by the tridiagonalization and backtransformation of the eigenvectors, and in these steps the number of operations only depends on n and b. More specifically, the operation count for reducing a banded matrix with semibandwidth b to tridiagonal form is $O(n^2 b)$ [12]. Thus, one would expect that the runtime of the LAPACK/PLASMA routines only depends on n and b, but hardly varies with spectral properties of the problem matrices. Surprisingly, we discovered that this is *not* the case, as illustrated in Section 4.

The other two methods, the BD&C algorithm and the BTF method, do not require the accumulation of the transformation for backtransforming the eigenvectors. The BD&C algorithm is based on the same technology for computing eigenpairs of rank-one modifications of a diagonal matrix as the tridiagonal eigensolver used in L/DSBEVD and is basically as accurate as this LAPACK routine (see [4] and Table 1). However, the computational cost of the BD&C algorithm strongly depends on the bandwidth b [4]. Moreover, the numerical properties of the input data influence the runtime of the BD&C algorithm in a similar way as the tridiagonal divide-and-conquer eigensolver. However, in the case of BD&C the complexity of the influenced part is much higher (absolutely as well as relatively in the context of the method), and thus a stronger influence on the overall runtime performance has to be expected.

In the BTF method, the number of iterations required in the eigenvector computation is influenced by numerical properties of the problem matrix. In situations where inverse iteration does not achieve orthogonality (e.g., strongly clustered eigenvalues), it also inherits these accuracy drawbacks. Although BTF cannot always guarantee orthogonality of the eigenvectors, the residuals are as good as in the other methods in all cases (cf. [10]). In the BTF method, all blocks of M_p need to be factorized, which costs $O(n^2 b^2)$ operations. This indicates that there is also a strong influence of the bandwidth b on its computational cost and we can expect it to be more competitive the smaller b is relative to n.

3. Experimental setup

Our runtime comparisons are based on randomly generated symmetric band matrices with the structure (1), but varying numerical properties in terms of eigenvalue distributions. We distinguish the following seven representative matrix types for our experiments (ε denotes machine precision):

Type 1: Random symmetric band matrices with elements uniformly distributed in $[0, 1)$
Type 2: Random eigenvalues drawn from a uniform distribution in $[-1, 1]$
Type 3: Eigenvalues geometrically distributed in $[-1, -\varepsilon] \cup [\varepsilon, 1]$
Type 4: Eigenvalues arithmetically distributed in $[-1, -\varepsilon] \cup [\varepsilon, 1]$
Type 5: Eigenvalues whose logarithms are uniformly distributed in $[-1, -\varepsilon] \cup [\varepsilon, 1]$
Type 6: Eigenvalues clustered around ± 1
Type 7: Eigenvalues clustered around $\pm\varepsilon$

The first two types represent symmetric band matrices whose eigenvalues tend not to be clustered, while the numerically more difficult matrix types 3, 4, 5, 6 are used for analyzing the behavior of the methods for various types of clustered eigenvalues. The last matrix type 7 is a “pathological” case where all eigenvalues are clustered around $\pm\varepsilon$.

The accuracy of the computed results is evaluated by the residuals $\mathfrak{R}_i$, defined as

$$\mathfrak{R}_i := \frac{\|(M_p - \lambda_i I) v_i\|_1}{\|M_p\|_1 \|v_i\|_1} \tag{9}$$

Table 1
Summary of numerical accuracy for $n = 3072$ and $b = 16$. For each matrix type, the percentages shown are averages over ten randomly generated test matrices.

Matrix type	$\mathfrak{R}_i \leqslant n\epsilon$ [%]			$\mathfrak{O}_i \leqslant n\epsilon$ [%]		
	BTF	BD&C, L/P/DSBEVD	L/DSBEVR	BTF	BD&C, L/P/DSBEVD	L/DSBEVR
1	100.0	100.0	100.0	85.4	100.0	97.7
2	100.0	100.0	100.0	99.9	100.0	100.0
3	100.0	100.0	30.0	99.9	100.0	30.0
4	100.0	100.0	100.0	98.4	100.0	99.5
5	100.0	100.0	100.0	92.6	100.0	99.4
6	100.0	100.0	100.0	0.0	100.0	99.6
7	100.0	100.0	100.0	0.0	100.0	100.0
avg	100.0	100.0	90.0	68.0	100.0	89.5

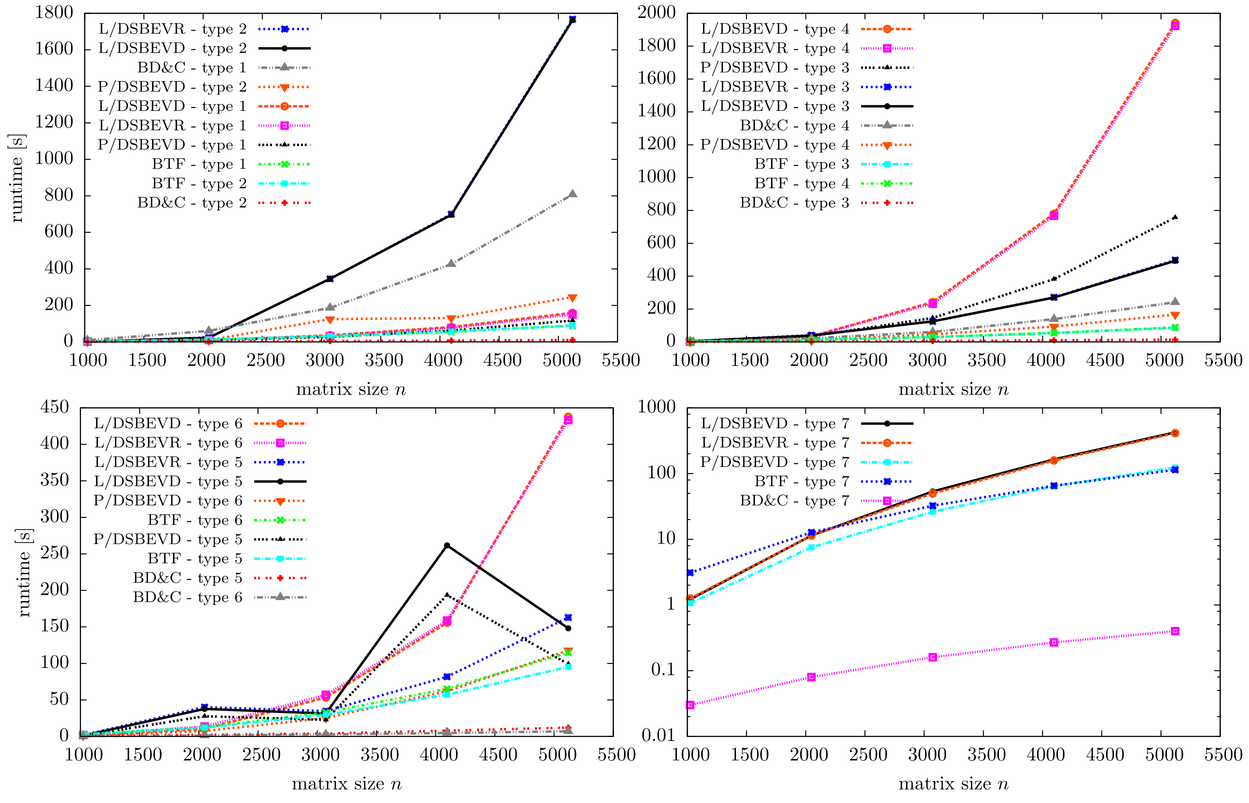

Fig. 1. Runtimes for all five methods for matrices with bandwidth $b = 16$. Different matrix types are shown in different subfigures, and the y-axis for matrix type 7 is scaled logarithmically. The order in the legend reflects the order of the corresponding runtimes.

and by the orthogonality error $\mathfrak{O}_i$ of the ith eigenvector approximation, defined as

$$\mathfrak{O}_i := \left\| \left(V^\top V - I\right)(:, i) \right\|_\infty. \tag{10}$$

In the BTF method, we terminated the inverse iteration process after the second iteration. The resulting residual satisfied $\mathfrak{R}_i \leqslant n\varepsilon$ in all test cases (cf. Table 1).

All experiments were performed on a single core of an Intel i7-860 CPU (2.8 GHz, 8 GB main memory) using the GNU Fortran 4.4.3 compiler. The CPU features *turbo boost* and *hyper-threading* were deactivated. At the software layer, version 3.2.2 of LAPACK, version 2.5.0b1 of PLASMA and version 3.6.0 ATLAS BLAS (only single-threaded) were used.

4. Runtime comparisons

The runtimes of the five different methods for varying dimensions, but fixed bandwidth $b = 16$ are summarized in Fig. 1 across the seven test matrix types. A first observation is that the runtimes of L/DSBEVD and L/DSBEVR are almost the same for all tested matrices. The second observation is that the runtimes of these two methods and also of P/DSBEVD strongly

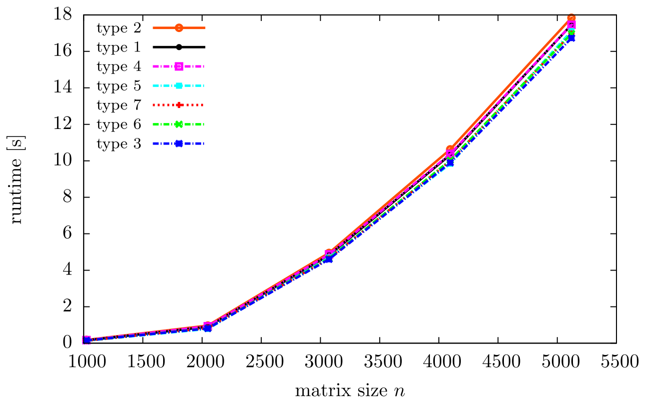

Fig. 2. Runtimes of the LAPACK routine L/DSBEVD for computing *eigenvalues only* of symmetric band matrices with different types and sizes with block size $b = 16$. The test matrices used were exactly the same as in Fig. 1.

vary with the matrix type. This is really surprising because, as mentioned in Section 2.5, one would expect the runtimes of the dominating parts (tridiagonalization and backtransformation) to be *independent* of the eigenvalue distribution. The reasons for this unexpected behavior will be explained in Section 5. The third observation is that the BTF method turns out to be fastest for matrix types 1 and 4, whereas BD&C is fastest for matrix types 2, 3, 5, 6 and 7. Overall, for a bandwidth $b = 16$ and large n one of the methods BD&C or BTF is always better than routines from LAPACK or PLASMA which need to perform a backtransformation of the eigenvectors. Only for matrix types 1 and 4 the PLASMA-based routine is faster than BD&C.

We mention that for matrix type 5 and $n = 4096$ and for nearly all dimensions of matrix type 3, L/DSBEVR was not able to compute all eigenpairs for the original test matrix used in the experiments. More specifically, L/DSTEMR did not converge and returned an error code. Therefore, for these cases we randomly generated other test matrices for which all eigenpairs could be computed and show the runtimes for these test matrices in Fig. 1.

In addition to the runtimes, we compared the numerical accuracy of all five methods in terms of the residuals $\mathfrak{R}_i$ and the orthogonality errors $\mathfrak{O}_i$ as defined in Eqs. (9) and (10), respectively. Table 1 illustrates for how many computed eigenpairs the methods compared were able to achieve a residual smaller than $n\varepsilon$. For the purpose of representativeness, we ran the experiments for ten randomly generated matrices of each matrix type. The percentages shown in Table 1 are averages over these ten instances per matrix type. Analogous information is given for the orthogonality error. We can see that in almost all cases (except L/DSBEVR on matrix type 3), the residual of all computed eigenpairs is below the threshold. Moreover, we can see that BD&C, L/DSBEVD and P/DSBEVD achieve a low orthogonality error in all cases. In contrast, the threshold $n\varepsilon$ for the orthogonality error is harder to achieve for BTF and L/DSBEVR. As already mentioned in [10], the orthogonality error of eigenvectors computed by BTF can be large for clustered eigenvalues (in some cases, it may even become $O(1)$ if the same eigenvector is approximated for shifts which are very close to each other). However, Table 1 illustrates that except for matrix types 6 and 7 with strongly clustered eigenvalues, the orthogonality error of BTF is quite comparable to the one of L/DSBEVR.

5. The influence of subnormal numbers

In this section, we discuss the connection between the runtimes reported in Section 4 and the occurrence of subnormal (denormalized) floating-point numbers.

5.1. Interpretation of experimental results

In Fig. 1 we saw that L/DSBEVD and P/DSBEVD are for some matrix types significantly slower than for others. This observation is very surprising, since the different spectral properties of the various matrix types should at most influence the runtime of the eigensolver of the tridiagonal matrix, whose computational cost tends to be negligible compared to the one of the tridiagonalization and backtransformation phases.

In Fig. 2 we show the runtime of L/DSBEVD for computing *eigenvalues only* (i.e., without computation and backtransformation of eigenvectors) for all matrix types. In contrast to Fig. 1, the runtime difference between different matrix types is very small. In fact, a more detailed investigation reveals that the runtime differences in Fig. 2 are only caused by the eigensolver of the tridiagonal matrix (amount of deflation) and that the runtime for the tridiagonalization phase is independent of the matrix type.

As a consequence of the differences between Figs. 1 and 2, we can conclude that the influence on the runtimes for different matrix types comes from the construction or from the application of the transformation matrix, which reduces the band matrix M_p to tridiagonal form. In fact, we will illustrate in the following, that the occurrence of *subnormal numbers*

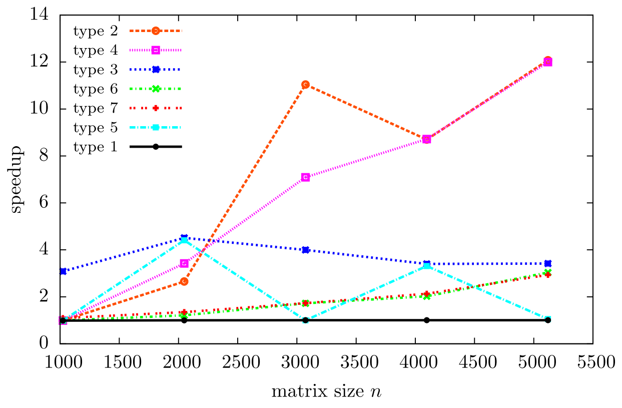

Fig. 3. Speedup of L/DSBEVD without subnormal numbers compared to L/DSBEVD with subnormal numbers. All matrices have a bandwidth $b = 16$.

causes these runtime differences. In order to make this clearer, let us first review the relevant aspects of floating-point arithmetic.

According to the IEEE 754 standard for floating-point arithmetic [13], a 64 bit double precision number contains 52 bits for the mantissa, 11 bits for the exponent and one bit for the sign. The minimal exponent for normalized floating-point numbers is -1022. Thus, the minimal gap between a regular double precision number and zero is 2^{-1022}. However, the IEEE 754 standard allows for *denormalizing* a non-zero floating-point number if a smaller exponent is needed to represent it correctly. More specifically, it defines the additional use of numbers in the range $[2^{-1074}, 2^{-1022})$. This process is called *gradual underflow* and the resulting floating-point numbers are called *subnormal* numbers (*denormalized* numbers in an earlier version of the standard). By using subnormal numbers it can be ensured that for two floating-point numbers x and y the relation $x = y \Leftrightarrow x - y = 0$ is always true.

There are various ways of implementing subnormal numbers. Even the most efficient implementation directly in hardware causes some overhead compared to regular floating-point arithmetic. Nevertheless, as shown in [14], most CPUs do not contain a hardware implementation of subnormal numbers. In this case, a kernel trap is raised to allow the operating system kernel to handle a floating-point operation with a subnormal number. This software emulation can take up to several hundred CPU cycles. Consequently, heavy use of subnormal numbers tends to lead to significant performance penalties. In [15] the handling of subnormal numbers in software has been called “one of the most troublesome aspects of the standard”. Still today, Intel recommends avoiding the use of subnormal numbers because of the large performance overhead [16].

5.2. Comparison of the methods without subnormal numbers

In order to illustrate the influence of subnormal numbers on the runtimes observed in Section 4, in this section we compare the runtimes of the BTF method, the BD&C algorithm, L/DSBEVD and P/DSBEVD *without* subnormal numbers. It turns out that the runtimes results of the BTF method and the BD&C algorithm are not influenced by gradual underflow and thus equal to Section 4, whereas L/DSBEVD and P/DSBEVD are significantly faster when gradual underflow is disabled. Since L/DSBEVR is influenced by disabling subnormal numbers in the same way as L/DSBEVD, we focus on the latter routine in this section.

Compilers usually have a flag for rounding subnormal numbers to zero (and thus “disabling” them). For the used GNU Fortran 4.4.3 compiler this flag is called *-funsafe-math-optimizations*. To ensure that the observed differences in the runtime are only caused by operations with subnormal numbers, we verified the results with an Intel compiler[2] which automatically disables subnormal numbers for all optimization levels higher than O0. However, for the Intel compiler, subnormal numbers can be explicitly activated or deactivated using the flag *-no-ftz* or *-ftz*,[3] respectively. Note that this strategy for avoiding the performance decrease associated with subnormal numbers violates the IEEE 754 standard [17].

5.2.1. Comparison of runtimes

The strong influence of subnormal numbers on the runtimes of Lapack and Plasma routines is illustrated in Fig. 3 for L/DSBEVD and in Figs. 4 and 5 for P/DSBEVD. In all figures we depict the “speedup” as the quotient of the runtimes with subnormal numbers divided by the runtimes when subnormal numbers are disabled.

We can see that in all cases the runtime for matrix type 1 is not influenced by disabling subnormal numbers, whereas significant speedups are possible for all other matrix types. We also notice that the influence of subnormal numbers is

[2] Intel(R) C++ Compiler XE 13.1 Update 1 for Linux.

[3] *-ftz* stands for flush-to-zero and causes that subnormal outputs of floating-point operations are rounded to zero and that subnormal inputs of floating-point operations are treated as zero.

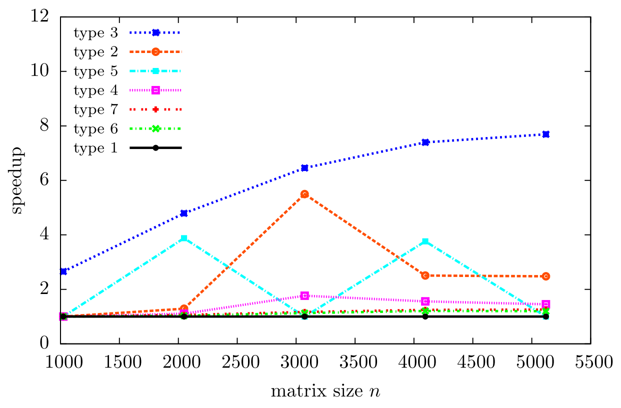

Fig. 4. Speedup of P/DSBEVD without subnormal numbers compared to P/DSBEVD with subnormal numbers. All matrices have bandwidth $b = 16$.

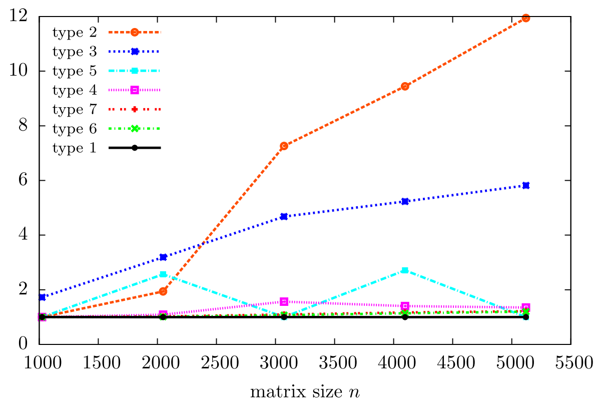

Fig. 5. Speedup of P/DSBEVD without subnormal numbers compared to P/DSBEVD with subnormal numbers. The tile size is in all cases reduced to force an *in-band* tridiagonalization. All matrices have bandwidth $b = 16$.

Table 2
Total runtimes t_z of P/DSBEVD without subnormal numbers relative to total runtimes t_{sn} with subnormal numbers as well as differences in the time t^{bt} required for backtransformation of the eigenvectors, both relative to t_{sn} for $n = 3072$ and $b = 16$. If a column sums up to 100%, all speedup occurs in the backtransformation process.

Matrix type	1	2	3	4	5	6	7
t_z/t_{sn} [%]	100.0	18.2	15.5	56.6	100.0	88.7	85.4
$(t_{sn}^{bt} - t_z^{bt})/t_{sn}$ [%]	0.0	81.8	83.5	43.4	0.0	11.3	14.6

different for the two routines. While the speedups of P/DSBEVD and L/DSBEVD are very similar for matrix type 5, for matrix type 3 the speedup of P/DSBEVD is higher. For all other matrix types, the speedup and thus the influence of subnormal numbers is higher for L/DSBEVD than for P/DSBEVD. In the worst case, the speedup for L/DSBEVD is up to twelve, while for P/DSBEVD it is below eight. We explain this different behavior later.

Before that, we illustrate that basically all the operations with subnormal numbers arise in the backtransformation of the eigenvectors. In Table 2, we show two sets of values across the different test matrix types for $n = 3072$ and $b = 16$: (i) the ratios of the total runtimes of P/DSBEVD without subnormal numbers to the runtimes with subnormal numbers (t_z/t_{sn}), and (ii) the differences in the times t^{bt} required for backtransforming the eigenvectors with and without subnormal numbers, relative to the total runtime of P/DSBEVD with subnormal numbers. Note that if the two values in one column sum up to 100%, then *all* speedup resulting from disabling subnormal numbers occurs in the backtransformation of the eigenvectors. This is the case for all matrix types except for matrix type 3, where a tiny portion of the effect occurs in the tridiagonalization process, as closer inspection reveals.

There are two important algorithmic differences between L/DSBEVD and P/DSBEVD. First, L/DSBEVD tridiagonalizes the banded matrix *in-band*, whereas P/DSBEVD performs an *out-of-band* reduction to tridiagonal form, because the original bandwidth used in our experiments ($b = 16$) is smaller than the default tile size (120) for the tridiagonalization. Thus, during the reduction process in P/DSBEVD, the bandwidth is increased up to the tile size. Second, L/DSBEVD accumulates the reduction transformation during the tridiagonalization process, which corresponds to an accumulation onto the identity

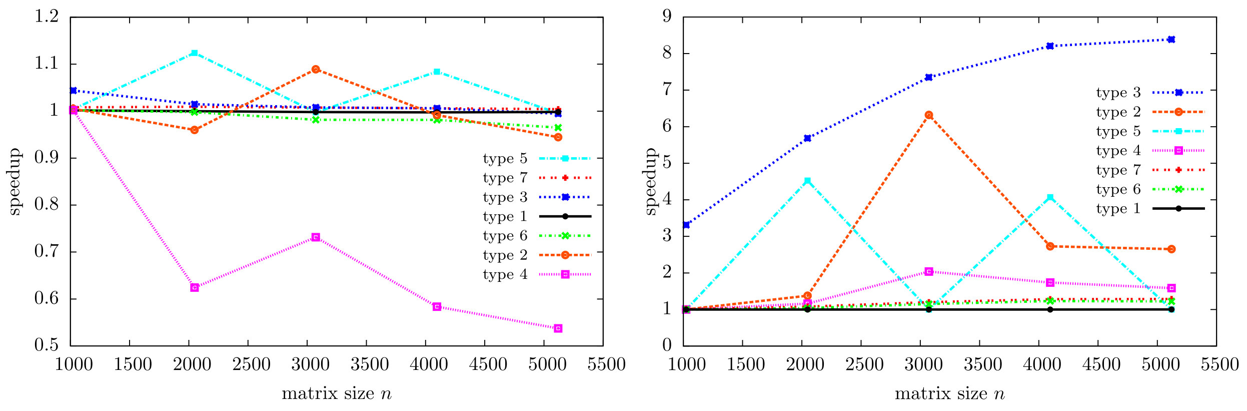

Fig. 6. Left subfigure: Speedup of the accumulation of the transformation on the identity matrix compared to direct application on the eigenvector matrix in Plasma (subnormal numbers enabled). Right subfigure: Speedup of direct application of the transformation in Plasma (out-of-band with default tile size) on the eigenvector matrix for subnormal numbers disabled over subnormal numbers enabled. All matrices have bandwidth $b = 16$.

matrix. In contrast, P/DSBEVD first only stores the relevant information for each transformation step during the reduction process and then applies the individual transformation steps in reverse order to the computed eigenvectors.

Obviously, P/DSBEVD can be "forced" to perform an in-band tridiagonalization by reducing the tile size used to the bandwidth. Interestingly, this also influences the effect of disabling subnormal numbers. Comparing Figs. 4 and 5, we see that the maximum speedup is higher for in-band tridiagonalization (matrix type 2). However, for all matrix types except for matrix type 2 in-band tridiagonalization reduces the speedup achieved by disabling subnormal numbers. However, in terms of absolute runtimes in-band tridiagonalization is much slower because of the non-optimal tile size.

In order to investigate whether the concrete strategy for accumulating and applying the transformation matrix influences the number of operations with subnormal numbers and thus the runtime, we modified P/DSBEVD to accumulate the transformation onto the identity matrix instead of applying it on the eigenvector matrix of the tridiagonal eigenproblem directly. In Fig. 6 we compare the runtimes of the two accumulation strategies. In the left subfigure the speedup of the accumulation of the transformations on the identity matrix is illustrated with activated subnormal numbers. We can see that the runtimes are roughly equal for both accumulation strategies, except for matrix type 4 where the accumulation onto the eigenvectors is faster. In the right figure the speedup of the accumulation on the eigenvectors without subnormal numbers over the accumulation with subnormal numbers is illustrated. We can see that the performance of the accumulation on the eigenvectors as well as on the identity matrix is highly influenced by the subnormal numbers.

5.2.2. Comparison of numerical accuracy

After having illustrated that disabling subnormal numbers can lead to significant performance gains for the methods which need to backtransform the eigenvectors, we also need to investigate which influence this has on the numerical error of the results. For this purpose, we compared the residuals $\mathfrak{R}_i$ of the results computed by L/DSBEVD, L/DSBEVR and P/DSBEVD with subnormal numbers enabled to the residuals $\mathfrak{R}_{z,i}$ of the results computed by L/DSBEVD, L/DSBEVR and P/DSBEVD when subnormal numbers are disabled based on the quantity $\mathfrak{S} := |\max_{i=1,n}(\mathfrak{R}_i) - \max_{i=1,n}(\mathfrak{R}_{z,i})| / \max_{i=1,n}(\mathfrak{R}_i)$.

It turns out that residuals and orthogonality error basically do not change. For example, for $b = 16$ over all tested matrices $\mathfrak{S}$ did not exceed $2 \cdot \varepsilon$. This shows that for the random test matrices all these routines achieve the same accuracy within significantly shorter time when subnormal numbers are disabled.

5.2.3. Methods with eigenvector backtransformation vs. BD&C and BTF

Since we could confirm that in our test cases the numerical accuracy of the tridiagonalization-based method from Lapack is not compromised by disabling the use of subnormal numbers, we now take a look at how this change influences the runtime comparison between L/DSBEVD and BD&C as well as BTF, which do not require backtransformation of the eigenvectors. As mentioned in Section 2.5, there is a significant influence of the bandwidth b on the computational cost of BD&C and BTF. Therefore, we included four different bandwidths $b = 8, 16, 32, 64$ in our experiments. The routines L/DSBEVR and P/DSBEVD tend to be slower than L/DSBEVD (since we did not invest time into optimizing the building blocks from Plasma for execution on a single core and for the range of bandwidths considered), and therefore we only include the comparison with L/DSBEVD in this section.

In Fig. 7 the speedup of BD&C and BTF over L/DSBEVD without subnormal numbers is illustrated for matrix type 1. As we already saw in Fig. 1, this matrix type is the most difficult type for BD&C, because very little or no deflation happens during the computation. Therefore, there is no speedup for small matrix sizes and it only grows slowly with n. The results for matrix type 4 are very similar and therefore not shown.

In Fig. 8 the speedup of BD&C and BTF over L/DSBEVD without subnormal numbers is illustrated for matrix type 2. Here, especially BD&C achieves much higher speedups than for matrix type 1 and in particular for small bandwidths the speedup grows significantly with n for both BD&C and BTF. The speedup values for matrix types 3 and 5 are similar, and even higher

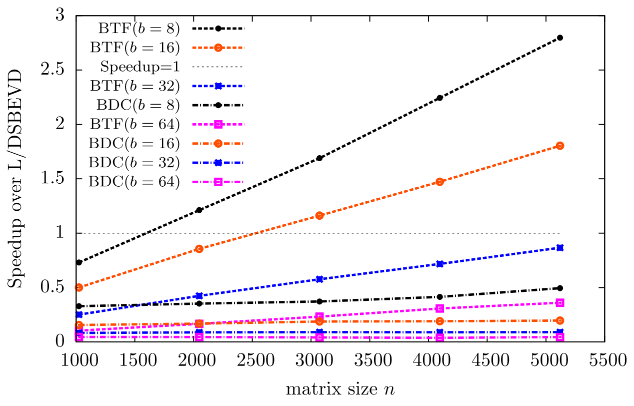

Fig. 7. Speedups of BTF and BD&C over L/DSBEVD for matrix type 1. The speedup values for matrix type 4 are very similar.

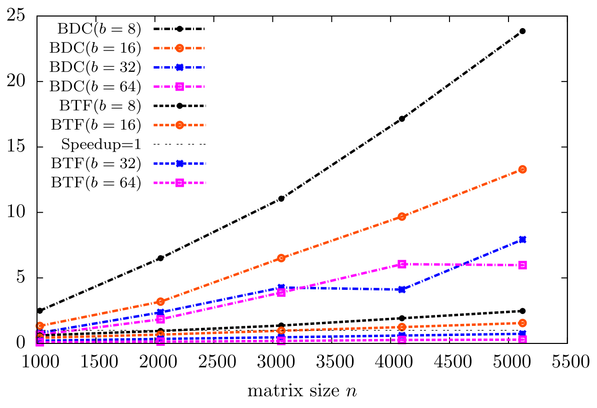

Fig. 8. Speedup of BTF and BD&C over L/DSBEVD for matrix type 2. The speedup values for matrix types 3 and 5 are very similar, and the speedup for matrix type 6 is even higher.

for matrix type 6. For matrix type 7, the speedup values of BD&C when subnormal numbers are disabled are also orders of magnitude higher due to the high amount of deflation (cf. Fig. 1). While the speedup of the BTF method is very similar for the different matrix types, the behavior of BD&C strongly depends on the clustering of the eigenvalues. For most matrix types BD&C is much faster than L/DSBEVD and in general the speedup increases with n. The decreasing speedup in Fig. 8 for the matrices $n = 4096$, $b = 32$ and $n = 5120$, $b = 64$ can be explained with variations in deflation.

Although for small n and large b, BD&C and in particular BTF actually exhibit a “slow-down” compared to L/DSBEVD, the speedup grows basically linearly with n for all tested bandwidths. Based on this observation, we can make a linear extrapolation for estimating for which matrix sizes n a speedup higher than 1 can be expected.

6. Conclusions

We performed a detailed experimental runtime comparison of five different methods for computing eigenpairs of randomly generated symmetric band matrices with different eigenvalue distributions. Our comparison included the routine L/DSBEVD from LAPACK, a routine L/DSBEVR based on building blocks from LAPACK, a routine P/DSBEVD constructed from building blocks available in PLASMA, our own implementation of the BD&C algorithm and our own implementation of the more recently proposed BTF method. Methodologically, the main distinction between the methods compared is that the first three initially reduce the given symmetric band matrix to tridiagonal form in a preprocessing step, whereas the BD&C algorithm never tridiagonalizes the entire band matrix. The BTF method as considered in this paper is in some sense a “hybrid” form between these two approaches, since it also tridiagonalizes the given band matrix, but it does not need to accumulate or apply the corresponding transformation matrix.

The experiments in this paper clearly illustrate a potential disadvantage of the backtransformation processes in solvers for symmetric banded eigenvalue problems. In particular (cf. Table 2 and Fig. 6), the backtransformation of the eigenvectors and the corresponding accumulation of the transformation from banded to tridiagonal form as implemented in state-of-the-art solvers for symmetric banded eigenvalue problems from LAPACK or PLASMA leads to many operations with subnormal numbers for several types of test matrices. It is well known that operations with subnormal numbers in many cases lead to a severe performance degradation. A precise analysis of the reasons for this strong occurrence of subnormal numbers in the LAPACK and PLASMA-based routines is work in progress.

We found that *disabling* the use of subnormal numbers significantly improves the performance of the Lapack and the Plasma-based routines (up to a factor of 12 in some test cases, cf. Figs. 3, 4 and 5) and makes them basically independent of the test matrix type, whereas it did not influence the runtimes of BD&C or BTF. Although disabling subnormal numbers formally violates the IEEE 754 standard for binary floating-point arithmetic, we did not observe any loss of numerical accuracy in our test cases.

However, even when disabling subnormal numbers, the two methods which do not require tridiagonalization of the given matrix and/or backtransformation of the eigenvectors turned out to be very competitive and in many cases faster than the Lapack- and Plasma-based routines, in particular for large problem sizes and/or small bandwidths. For example, the BD&C algorithm is the clear winner in all test cases with clustered eigenvalues, since the amount of deflation tends to be high in these cases. Compared to L/DSBEVD, for a bandwidth $b = 8$ BD&C achieved speedup values over 20 even for matrices with eigenvalues drawn from a uniform distribution, and speedup values over 100 for the pathological case of matrices with eigenvalues clustered around $\pm$ machine epsilon. In the test cases where very little deflation occurs, BD&C does not perform so well. In these cases (e.g., random matrices with uniformly distributed elements), the BTF method tends to have performance advantages over the other methods for increasing problem sizes and/or small bandwidths (linearly growing speedup values, almost reaching three in the range of problem sizes considered). Only for strongly clustered eigenvalues, the BTF method sometimes has shortcomings in terms of the orthogonality of the computed eigenvectors. In some cases, it approximates the same eigenvector for different shifts, and then the orthogonality error can be as large as $O(1)$, as already discussed in [10]. Future work will thus concentrate on investigating potential approaches for improving the eigenvector orthogonality in the BTF method.

Acknowledgements

This work was partially supported by the Austrian Science Fund FWF: project number S10608 (NFN SISE). We would like to thank the anonymous referees for their valuable comments which helped us improving the manuscript.

References

[1] P. Luszczek, H. Ltaief, J. Dongarra, Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures, in: Parallel Distributed Processing Symposium (IPDPS), IEEE International, 2011, pp. 944–955.
[2] A. Haidar, H. Ltaief, J. Dongarra, Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, ACM, New York, NY, USA, 2011, pp. 8:1–8:11.
[3] W.N. Gansterer, R.C. Ward, R.P. Muller, An extension of the divide-and-conquer method for a class of symmetric block-tridiagonal eigenproblems, ACM Trans. Math. Softw. 28 (2002) 45–58.
[4] W. Gansterer, R. Ward, R. Muller, W. Goddard, Computing approximate eigenpairs of symmetric block tridiagonal matrices, SIAM J. Sci. Comput. 25 (2003) 65–85.
[5] E. Anderson, Z. Bai, C.H. Bischof, S. Blackford, J.W. Demmel, J.J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D.C. Sorensen, Lapack Users' Guide, 3rd edition, SIAM Press, Philadelphia, PA, 1999.
[6] University of Tennessee Knoxville, Plasma users's guide, parallel linear algebra software for multicore architectures, Version 2.3, 2010, http://icl.cs.utk.edu/projectsfiles/plasma/pdf/users_guide.pdf.
[7] J. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, Numer. Math. 36 (1980) 177–195.
[8] M. Gu, S.C. Eisenstat, A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem, SIAM J. Matrix Anal. Appl. 16 (1995) 172–191.
[9] I.S. Dhillon, B.N. Parlett, Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices, Linear Algebra Appl. 387 (2004) 1–28.
[10] G. König, M. Moldaschl, W.N. Gansterer, Computing eigenvectors of block tridiagonal matrices based on twisted block factorizations, J. Comput. Appl. Math. 236 (2012) 3696–3703.
[11] J. Demmel, O. Marques, B. Parlett, C. Vömel, Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers, SIAM J. Sci. Comput. 30 (2008) 1508–1526.
[12] B.N. Parlett, The Symmetric Eigenvalue Problem, SIAM Press, Philadelphia, PA, 1997.
[13] IEEE standard for floating-point arithmetic, IEEE Std 754-2008, 2008, pp. 1–58.
[14] I. Dooley, L. Kale, Quantifying the interference caused by subnormal floating-point values, in: Proceedings of the Workshop on Operating System Interference in High Performance Applications, 2006.
[15] D. Goldberg, What every computer scientist should know about floating point arithmetic, ACM Comput. Surv. 23 (1991) 5–48.
[16] Intel Corporation, Intel®64 and IA-32 Architectures Optimization Reference Manual, vol. A, April 2012, pp. 3–95, http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html.
[17] W. Kahan, Lecture notes on the status of IEEE standard 754 for binary floating-point arithmetic, http://www.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps, 1996.