International Conference on Computational Science, ICCS 2010

Toward a parallel solver for generalized complex symmetric eigenvalue problems

Hannes Schabauer, Christoph Pacher, Andrew G. Sunderland, Wilfried N. Gansterer

Abstract

Methods for numerically solving generalized complex symmetric (non-Hermitian) eigenvalue problems (EVPs) $Ax = ABx$ serially and in parallel are investigated. This research is motivated by two observations: Firstly, the conventional approach for solving such problems serially, as implemented, e.g., in zggev (LAPACK), is to treat complex symmetric problems as general complex and therefore does not exploit the structural properties. Secondly, there is currently no parallel solver for dense (generalized or standard) non-Hermitian EVPs in ScaLAPACK. The approach presented in this paper especially aims at exploiting the structural properties present in complex symmetric EVPs and at investigating the potential trade-offs between performance improvements and loss of numerical accuracy due to instabilities. For the serial case, a complete reduction based solver for computing eigenvalues of the generalized complex symmetric EVP has been designed, implemented, and is evaluated in terms of numerical accuracy as well as in terms of runtime performance. It is shown that the serial solver achieves a speedup of up to 43 compared to zggev (LAPACK), although at the cost of a reduced accuracy. Furthermore, the major parts of this reduction based solver have been parallelized based on ScaLAPACK and MPI. Their scaling behavior is evaluated on a cluster utilizing up to 1024 cores. Moreover, the parallel codes developed achieve encouraging parallel speedups comparable to the ones of ScaLAPACK routines for the complex Hermitian EVP.

Keywords: generalized complex symmetric eigenproblem, complex symmetric reflector, indefinite factorization, eigensolver

1. Introduction

Generalized complex symmetric EVPs are a special variant of generalized complex non-Hermitian EVPs. Although they do not occur as frequently in practice as real symmetric or complex Hermitian problems, there are many important applications where they arise [1]. An important example is the numerical solution of Maxwell’s equations...
with complex material coefficients (accounting for losses) and/or certain absorbing boundary conditions used in the simulation of optoelectronic devices [2, 3, 4, 5].

Original routines from LAPACK or ScaLAPACK mentioned in this paper are followed by “(LAPACK)” or “(ScaLAPACK)”, respectively. The conventional approach for solving generalized complex symmetric EVPs \( Ax = \lambda Bx \), as implemented e.g. in \texttt{zggev} (LAPACK) [6], is to treat them as general complex and therefore abstain from utilizing the structural symmetry. For the serial case, we have implemented a complete solver for computing eigenvalues that is being evaluated in this paper. The approach is considerably faster than \texttt{zggev} (LAPACK). However, we investigate numerical accuracy, being aware about potential losses due to an indefinite factorization and due to the use of non unitary transformations in the tridiagonalization process.

A parallel solver for non-Hermitian dense EVPs is currently not available in ScaLAPACK [7]. Currently, there also seem to be no future plans for explicitly handling complex symmetric EVPs in these standard libraries [8]. SLEPc [9] features the routine EPSSolve which is able to solve non-Hermitian complex generalized EVPs in parallel, but its focus is on sparse problems rather than on dense ones. In this paper, we discuss and evaluate a complete serial generalized complex symmetric solver to compute eigenvalues and some central components of a corresponding parallel eigensolver which we designed and implemented.

**Related Work**

Some papers address *standard* \((Ax = \lambda x)\) dense complex symmetric EVPs, including [10, 11, 12] by applying complex orthogonal transformations.

Papers about generalized dense complex symmetric eigensolvers are very rare. In [13], a procedure based on the generalized Jacobi method is proposed, where two very small examples are given, but neither accuracy nor runtimes are evaluated. Projection methods have been applied to generalized sparse complex symmetric EVPs, e.g., subspace iteration [14] or variants of the Jacobi-Davidson method [2, 5]. For dense problems, or if large parts of the spectrum have to be computed, reduction methods may be preferable in terms of performance. The most common strategy so far for reduction-based complex symmetric eigensolvers is to ignore the algebraic properties and to apply the technology available for general non-Hermitian problems, as demonstrated in \texttt{zggev} (LAPACK). More specifically, this means that firstly \( B \) is reduced to triangular form by applying a \( QR \) decomposition, then the problem is reduced to generalized Hessenberg form using unitary transformations. From the generalized Hessenberg form, eigenvalues and eigenvectors are computed with the \( QZ \) algorithm.

In our paper, we focus on the complete process of a dense generalized complex symmetric eigensolver for computing eigenvalues, and in contrast to most of the existing work, we both implement and evaluate the solver. Most of the discussed papers focus on aspects of the sub-methods, rather than evaluating large problems on state-of-the-art computer infrastructures. Moreover, we observe that methods and both serial and parallel codes especially tailored for generalized complex symmetric EVPs are very rare, so are evaluations of them. The approach presented in this paper is the continuation of our efforts that started with the investigation of the sequential splitting [3] and non-splitting [15] tridiagonalization processes. In this paper, we investigate a solver for generalized complex symmetric EVPs in serial and parallel.

**Synopsis**

The remainder of this paper is organized as follows. Section 2 discusses algorithmic methodology and the mathematical background, while Section 3 discusses the corresponding implementations. Section 4 summarizes experimental results for the serial solver and parallel codes in terms of accuracy and runtime behavior, and Section 5 concludes and points out future work.

### 2. Methodology

This section discusses algorithmic strategies of the methods developed and implemented. These foundations apply to both the serial and parallel codes.

The traditional approach for solving a generalized complex symmetric EVP is to treat it as a generalized complex non-Hermitian problem and to use the LAPACK routine \texttt{zggev}, since currently no solver is available that exploits the complex symmetry. \texttt{zggev} firstly decomposes \( B \mapsto QR \) in routine \texttt{zgeqrf}, then forms \( Q^H A \) in routine \texttt{zunmqr}. This
transforms the original EVP $Ax = \lambda Bx$ into the problem $Q^H Ax = \lambda Rx$. Subsequently, using Givens rotations, $Q^H A$ and $R$ are simultaneously reduced to Hessenberg and triangular form, respectively, in routine zgghrd, resulting in the problem $Hy = \lambda y$. Finally, the QZ algorithm is used in zggeqz for computing eigenvalues, and ztgevc can be used for computing eigenvectors.

2.1. Basic Approach

The procedure investigated in this paper for solving $Ax = \lambda Bx$ follows the cardinal steps (i) factorization $B \mapsto LL^\top$, (ii) reduction to standard form $My = \lambda y$, (iii) tridiagonalization to $Tz = \lambda z$, (iv) computing eigenvalues $\lambda_k$ and eigenvectors $z_k$ (if desired) of the tridiagonal problem $T$, and (v) backtransformation of eigenvectors from $z$ to $x$ if desired.

The algorithms for the real symmetric and for the complex Hermitian case use orthogonal and unitary transformation matrices, respectively. In our method for the complex symmetric case, we need to use complex orthogonal transformation matrices with norms potentially larger than one [15, 3]. This leads one to expect a loss of numerical accuracy. However, in practice this does not always have to be prohibitive, as our experiments in Section 4 illustrate.

Overall, the dominating step in terms of computational effort is the tridiagonalization step. For tridiagonalizing a complex symmetric matrix $M$, two basic approaches can be distinguished: the splitting and the non-splitting method. The splitting approach [12, 3] is based on separating the tridiagonalization of the real part of $M$ from the tridiagonalization of the imaginary part of $M$ (which are both real symmetric matrices) as much as possible. The non-splitting approach is an alternative which operates on the complex symmetric matrix as a whole, based on generalizations of complex Householder reflectors [15]. Earlier results indicate that splitting methods may achieve better numerical accuracy, but seem somewhat limited performance-wise and in terms of parallelization potential [3]. Consequently, the approach described in this paper is based on the non-splitting method.

2.2. Reduction to Standard Problem

In the presence of efficient solvers for standard EVPs, it appears obvious to start by reducing the generalized EVP to a standard EVP. The first step of the reduction is a symmetry preserving factorization of $B \mapsto LL^\top$. Subsequently we compute $M = L^{-1}AL^{-\top}$ to arrive at a standard EVP $My = \lambda y$, where $y = L^\top x$.

The usual way for factorizing positive definite matrices is the Cholesky factorization [16], while a Bunch-Kaufman (BK) factorization [16] should be applied for indefinite matrices. Although it has been argued that Cholesky factorization without pivoting can be generalized to some complex symmetric matrices [17, 18], in general this can produce ill-conditioned factors $L$. Nevertheless, we decided to investigate the generalization of standard Cholesky factorization since higher parallel performance than BK factorization is anticipated.

2.3. Tridiagonalization

The basic building block for the non-splitting complex symmetric tridiagonalization method is a generalization of Householder reflectors [16]. As illustrated in [15], a complex symmetric reflector $M$ for complex symmetric matrices can be defined formally analogously to the well known Householder reflectors for real symmetric matrices, i.e.,

$$M = I_n - \frac{2}{x^\top x}xx^\top,$$

where the vector $x \in \mathbb{C}^n$ that eliminates all but the first entry of a given vector $z \in \mathbb{C}^n$ (i.e., $Mz = \hat{z}e_1$, $\hat{z} = \pm \sqrt{\bar{z}^\top z} \in \mathbb{C}$) is given (scaled to $x_1 = 1$) as

$$x_1 = 1, \quad x_k = z_k/(z_1 - \hat{z}) \quad \text{for} \quad k \in \{2, \ldots, n\}, \quad \text{and} \quad \frac{2}{x^\top x} = \frac{\hat{z} - z_1}{\hat{z}} = 1 - \frac{z_1}{\hat{z}}.$$  

To avoid numerical errors due to cancellation and to keep $x_1$ and $x_2, \ldots, n$ of the same order of magnitude, the sign for $\hat{z}$ is chosen such that $|z_1 - \hat{z}|$ is maximized. Note that $\sqrt{x^\top x}$ for $x \in \mathbb{C}^n$ is not a norm (consider, e.g., $(1 i)(1 i)^\top = 0$). Symmetry-preserving complex symmetric reflectors are not unitary. This causes numerical instabilities.

2.4. Solution of the Tridiagonal Problem

After tridiagonalizing, the spectrum of the complex symmetric EVP $Tz = \lambda z$ can be computed by applying a QL procedure [19].
3. Implementation

At this point, we discuss our serial and parallel implementations of the methodology summarized in Section 2. As in LAPACK and ScaLAPACK, all our implementations feature both blocked and unblocked versions. Our new routines are named according to the LAPACK / ScaLAPACK naming scheme.

3.1. Sequential Approach

The basis for our implementation is our solver zsyevn for the standard complex symmetric EVP developed earlier [15]. Starting from the LAPACK solver for the generalized real symmetric EVP with positive definite $B$, dsygv (LAPACK), we adapted the missing two steps for solving a generalized complex symmetric EVP, i.e., factorization of $B$ and reduction to the standard problem. Consequently, we call our solver for generalized complex symmetric problems zsygvn (the $n$ indicates that in the tridiagonalization phase the non-splitting method is used and not the splitting method [3]). In the following, we will describe the basic building blocks of zsygvn.

The procedure starts with factorizing $B \mapsto LL^T$ in the routine zpotrfi, a modification of the real symmetric factorization routine dpotrf (LAPACK) which computes a symmetric factorization of the complex symmetric matrix $B$. In complex numbers, the Cholesky factorization procedure does not break down and thus the routine neither tests for definiteness nor performs pivoting.

The following reduction of the generalized problem EVP $AX = \lambda BX$ to standard form $MY = \lambda Y$ is performed in the routine zsygst by computing $M = L^{-1}AL^{-T}$. Our implementation is algorithmically slightly different from the one in the LAPACK routine daga2st, which applies a transformation to the input matrix from both sides at the same time. Our solution implemented in zsygst is a simplified preliminary implementation of this operation for the purpose of rapid prototyping. zya2st consecutively solves two linear systems of the type $LX = B$ with $L$ from the factorization step in order to construct $M = L^{-1}AL^{-T}$. In more detail, first $zttrs$ (LAPACK) is used for solving $LX = A$ for $X$, i.e., $X = L^{-1}A$. Then, $X$ is transposed, and $zttrs$ is used again for solving $LM = X$ for $M$, i.e., $M = L^{-1}X = L^{-1}AL^{-T}$.

Solving two linear systems of equations with the ill-conditioned factor $L$ introduces a certain amount of error into $M$. In future work, we will explore alternative indefinite factorizations, e.g. a BK factorization, to eliminate this fundamental source of numerical instability.

Next, the routine zsyevn [15], which has a functionality corresponding to dsyev from LAPACK, computes the eigenvalues of $M$. In zsyevn, first the routine zsytr2 (zsytr1 is our splitting routine, not discussed in this paper) is called for transforming the matrix $M$ to tridiagonal form using the non-splitting method as described in [15], resulting in the problem $Tz = \lambda z$.

The eigenvalues of the resulting complex symmetric tridiagonal problem are computed with a slightly modified version of the routine compev implementing a QL procedure [20]. Modifications involved changes to allow proper compilation on modern compilers, removals of unused codelines (e.g., print-statements), and substitution of the hardcoded machine precision $\epsilon$ by a call to the corresponding LAPACK routine dlamch. At the moment, our implementation does not yet support the computation of eigenvectors of the complex-symmetric EVP.

3.2. Parallel Approach

Our parallel solver for generalized complex symmetric EVPs is a ScaLAPACK-based MPI-style parallelization of the sequential codes summarized in Section 3.1 for both shared and distributed memory architectures. The algorithmic methodology for the serial and parallel case is the same. Not all functionality available in LAPACK is also implemented in ScaLAPACK. In particular, there are no parallel solvers for complex non-Hermitian EVPs in ScaLAPACK. Our starting point is based on the parallel real symmetric variant pdsysv (ScaLAPACK). pdsysyv first calls pdpotrf to factorize $B$, then applies pdsyngst to reduce the generalized to a standard EVP, followed by pdsyevx to solve it.

In terms of computational steps, our parallel solver matches its sequential counterpart, adding parallel functionality to the respective parts. In common with other ScaLAPACK driver routines, the method is parallelized by a data parallel mechanism utilizing a block-cyclic distribution. The hierarchy of calling subroutines, distribution- and communication schemes of our implementation are the same as in the parallel solvers for real symmetric problems implemented in ScaLAPACK.

The developed parallel driver routine for solving a generalized complex symmetric EVP is called pzsygvn. Besides auxiliary routines, it calls (i) pzpotrf for a parallel complex symmetric indefinite factorization, (ii) pzsygst for a parallel reduction of the generalized complex symmetric EVP to a standard complex symmetric EVP, and (iii)
pzsyevn for solving the corresponding standard EVP. pzsyevn first applies a parallel non-splitting tridiagonalization and transforms the matrix of the standard problem to tridiagonal form. The rest of this routine is work in progress. In particular, we do not yet have a parallel solver for computing eigenpairs of the resulting complex symmetric tridiagonal problem. Our implementations of Steps (i)-(iii) are new developments which have not yet been discussed elsewhere.

4. Performance Study

This section analyzes the performance of the newly developed serial solver routine zsygvn for generalized complex symmetric EVPs and of parts of its parallel pendant pzsygvn. We started evaluating accuracy and sequential runtime performance on our SMP system called Goedel and subsequently evaluated parallel runtimes on a cluster called HPCx.

Our test data consists of complex symmetric matrices from two different types, type RND (standard EVP) and type SP (generalized EVP). For type RND, a matrix $M$ is constructed according to $M := S + S^T$, where $S$ has real and imaginary parts uniformly distributed in the interval $[0, 1]$. For type SP, matrix $B$ is created randomly as for type RND, but matrix $A$ is constructed such that the generalized problem $(A, B)$ has a given spectrum: We start with two random complex symmetric matrices, $B$ and $Z$. Using zggev (LAPACK) we compute the matrix $X = (x_1, x_2, \ldots, x_\nu)$ of right eigenvectors of $Z$. We double-check that $Z$ has no multiple eigenvalues as well as that $x_\nu^T x_\nu \neq 0$ and $x_\nu^T x_\lambda$ is sufficiently small for $\nu \neq \lambda$ for all (pairs of) eigenvectors. After scaling each eigenvector, $x_\nu \mapsto x_\nu (x_\nu^T x_\nu)^{-0.5}$. $X$ is a complex orthogonal matrix, i.e. $XX^\top = X^\top X = I$. Then we construct a diagonal matrix $\Lambda$ with prescribed eigenvalues $\lambda_k = k + (k-1)^{2+1}i$. We deliberately choose the eigenvalues with a big distance between two neighbors. Then, $B$ is factorized $B \mapsto LL^\top$. Finally, the complex symmetric matrix $A := LXAX^\top L^\top$ is constructed. The generalized problem $(A, B)$ then has the prescribed $\lambda_k$ as eigenvalues.

4.1. Evaluation of Sequential Solver

We evaluated numerical accuracy and runtime performance of our sequential solver zsygvn for generalized complex symmetric EVPs. For this purpose, the sequential codes were run on a Sun Fire v40z with 4 dual-core Opteron 875 CPUs (2.2 GHz) and 24 GB main memory, where only a single core was used and remaining cores were (to the extent possible) idle. Suse Linux Enterprise Server 10, the GNU Fortran 95 (GCC) 4.1.2 compiler optimizing 03, LAPACK 3.1.1, and Goto BLAS 1.20 were used.

4.1.1. Numerical Accuracy

We discuss maximum relative eigenvalue errors $\mathcal{E} := \max_k \frac{|\lambda_k - \hat{\lambda}_k|}{|\lambda_k|}$, where $\lambda_k$ and $\hat{\lambda}_k$ denote the exact value and the result of the evaluated routine, respectively.

Having constructed the test matrix pairs $(A, B)$, we compute the eigenvalues $\lambda_k$ of $Ax = \lambda Bx$ with zggev (LAPACK). We observe that the test problems are indeed numerically very challenging, as we get a relevant deviation of $\lambda_k$ from the prescribed eigenvalues $\hat{\lambda}_k$. We attribute most of this deviation to the construction of the test matrix pairs (which involves non-unitary matrices with partly high condition numbers), and not to zggev (LAPACK).

Figure 1 shows this deviation (take $\lambda_k$ for $\hat{\lambda}_k$) together with eigenvalue errors for our solvers zsygvn and zsyevn (compared to corresponding LAPACK routines) for several problem sizes and different test data. Our standard solver achieves smaller errors than our generalized solver. From comparing the eigenvalue error of our standard non-splitting solver zsyevn to the eigenvalue error of our generalized solver zsygvn we can estimate that the additional indefinite factorization of $B$ and the reduction step to the standard EVP lead to the loss of one to two significant digits in the resulting eigenvalues.

---

1 See website of HPCx, http://www.hpcx.ac.uk/.
2 In exact arithmetic, the eigenvectors of different eigenvalues of any (complex) symmetric matrix satisfy $x^\top y = 0$.
3 Randomly generated complex triangular matrices tend to have very large condition numbers [21]. This is not the case for randomly generated full complex symmetric matrices. Therefore, we start with generating a full random complex symmetric matrix $B$. 
Ezsyevn vs. zgeev, RND
E ˆλk vs. zggev, SP
E zayevn vs. zggev, SP
E zayevn vs. zgeev, LAPACK

Figure 1: Maximum relative errors of eigenvalues, ε, of (i) new generalized complex symmetric solver zsygvn compared to the dense generalized general solver zggev (LAPACK) for test matrices of type SP, (ii) prescribed eigenvalues, ˆλ, from calculated eigenvalues with zggev (LAPACK), and (iii) complex symmetric solver zsyevn compared to zgeev (LAPACK).

4.1.2. Runtimes

For evaluating runtime performance of zsygvn, we compare it with its competitor routine for general complex problems, zggev (LAPACK). A speedup S is computed as $S = \frac{T_1}{T_2}$, where $T_1$ denotes the time consumed for the solution of the problem when using routine zggev (LAPACK), and $T_2$ denotes the time consumed for the solution of the same problem when using routine zsygvn. Figure 2 shows the speedup curve. We observe an increasing speedup with increasing problem sizes, reaching a value of 43 for larger problem sizes. This increase of the speedup S with the matrix order is due to the fact that our routine zsygvn has a lower asymptotic complexity than zggev (operating with a tridiagonal matrix in the final phase instead of a Hessenberg matrix since symmetry is preserved).

4.2. Evaluation of Parallel Solver

Parallel codes were run on HPCx, which is a cluster featuring 168 IBM eServer 575 LPARs running IBM AIX and offering 1280 dual core IBM Power5 CPUs (2560 cores). This computer provides a theoretical peak performance of 15.3 TFlop/s. All codes were compiled with the IBM xlf 10.1.0.10, LAPACK 3.0, ScaLAPACK 1.7, BLAS codes linked against IBM ESSL 4.3.0.0.

Parallel evaluations involve problem sizes of order 4096, 6144, and 8192 on 1, 2, 4, 8, ..., 512, and 1024 nodes on HPCx. Experimental analyses revealed that (as anticipated) parallel routines feature practically the same accuracy as corresponding sequential routines, therefore we only show runtime results for the parallel case.

Before evaluating parallel scalability, it is important to measure the shares of runtimes of individual routines, in order to determine dominating parts. Therefore, we measured runtimes of the parallel code on a single core for different orders $n$ on the same machine that we use for parallel evaluations. We observe that the factorization of $B$ (pzpotrfi) consumes relatively little time, reduction from generalized to standard problem (pzsygst) consumes a considerably bigger share, and tridiagonalization (pzsytr2) consumes most of the time. For larger orders, we find the same sequence of shares with factorization taking a slightly increasing share. For order 8192, we measure a share of 5% for factorization, 35% for reduction, and 60% for tridiagonalization. Figure 3 depicts absolute runtimes (logscale) of the analyzed routines. Accordingly, the tridiagonalization step is the dominating part for all analyzed orders on a single core of HPCx.

Figure 4 depicts the parallel scaling behavior of the factorization, reduction, and tridiagonalization parts of pzygyvn, as well as the one of the Hermitian tridiagonalization pzhetrd (ScaLAPACK), for comparison. The relative speedups of a problem with order $n = 8192$ on 2 to 1024 cores in parallel are shown. pzpotrfi scales well until
Schabauer et al. / Procedia Computer Science 1 (2012) 437–445

Speedups of zsygvn

Figure 2: Speedups of zsygvn over zggev (LAPACK).

about 512 cores, for 1024 cores the consumed overall runtime remains roughly stable. pzsygst scales very well for 2 to 1024 cores. For parallel tridiagonalization in pzsytr2, we observe a somewhat surprising superlinear speedup on 4 and 8 cores within a single node that we believe is probably caused by cache effects. Superlinear speedups on HPCx have also been reported, e.g., in a thesis by E. Davidson on studying the performance of a lattice Boltzmann code [22, p.33]. pzsytr2 scales well for 2 to 64 cores, then parallel scalability reduces, the maximum feasible number of cores for this routine on HPCx turns out to be 512; for 1024 cores, pzsytr2 is slower than on 512 cores. For the purpose of proving that the dominating routine pzsytr2 does not contain any "odd" runtime behavior, we compared it with the Hermitian routine pzhetrd which is algorithmically very similar; the main difference being the different computation of the complex elementary reflector. Unsurprisingly, evaluations for tridiagonalization evidence a very similar runtime behavior for all number of cores. In exact numbers, we measure speedups of 139/317/115 for routines pzpotrfi/pzsygst/pzsytr2 on 512 cores, and 150/453/80 on 1024 cores, respectively. Scalability of the new routines is comparable to that of state-of-the-art ScaLAPACK routines, as the comparison with pzhetrd shows.

5. Conclusions

A sequential generalized complex symmetric eigensolver zsygvn and principal components (including factorization, reduction to standard EVP, and tridiagonalization) of its parallel counterpart pzsygvn have been investigated. The achieved numerical accuracy is not yet satisfactory and reflects the numerical difficulties in the current version of the symmetry-preserving solver. In addition to the non-splitting tridiagonalization, the critical parts in terms of loss of accuracy are the indefinite factorization and the reduction operation to a standard EVP. Improvements in the latter two are expected from the integration of appropriate pivoting strategies. At the same time, we achieved speedups up to 43 compared to zggev (LAPACK). Scalability studies for the components of the parallel solver pzsygvn on 2 to 1024 cores of a high performance cluster evidence very encouraging results. In the future, we will focus on numerical aspects and on the completion of the parallel generalized complex symmetric eigensolver.

References

Figure 3: Absolute runtimes in seconds for factorization in \texttt{pzpotrfi}, reduction in \texttt{pzsygst}, and tridiagonalization in \texttt{pzsytr2} of the parallel code on a single CPU core on HPCx.

Figure 4: Parallel speedups on HPCx, featuring 2 to 1024 cores ($n = 8192$). Evaluated codes include factorization in \texttt{pzpotrfi}, reduction to standard EVP in \texttt{pzsygst}, and tridiagonalization in \texttt{pzsytr2}. \texttt{pzsytr2} is compared with Hermitian pendant \texttt{pzhetrd} (ScALAPACK).