A parallel structured banded DC algorithm for symmetric eigenvalue problems

Shengguo Li[†], Xia Liao[†], Yutong Lu[⋆], Jose E. Roman[‡], and Xiaoqiang Yue[§]

[†] *College of Computer Science, National University of Defense Technology, Changsha, China, 410073*
[⋆] *National Supercomputer Center in Guangzhou, and the School of Data and Computer Science, Sun Yatsen University, Guangzhou, China, 510006*
[‡] *D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Camí de Vera s/n, València, Spain, 46022*
[§] *Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Key Laboratory of Intelligent Computing & Information Processing of Ministry of Education, Xiangtan University, Xiangtan, China, 411105*

July 1, 2023

Abstract

In this paper, a novel parallel structured divide-and-conquer (DC) algorithm is proposed for symmetric banded eigenvalue problems, denoted by PBSDC, which modifies the classical parallel banded DC (PBDC) algorithm by reducing its computational cost. The main tool that PBSDC uses is a parallel structured matrix multiplication algorithm (PSMMA), which can be much faster than the general dense matrix multiplication ScaLAPACK routine `PDGEMM`. Numerous experiments have been performed on Tianhe-2 supercomputer to compare PBSDC with PBDC and ELPA. For matrices with few deflations, PBSDC can be much faster than PBDC since computations are saved. For matrices with many deflations and/or small bandwidths, PBSDC can be faster than the tridiagonalization-based DC implemented in LAPACK and ELPA. However, PBSDC would become slower than ELPA for matrices with relatively large bandwidths.

1 Introduction

The banded symmetric eigenvalue problem is very important in various computational science applications. Matrices generated from electronic structure calculations usually have banded structures [39, 4]. Banded matrices also appear in the block Lanczos algorithm [40] and appear in the intermediate step of solvers for dense eigenvalue problems, where a dense matrix is first reduced to a banded form and finally to the tridiagonal form [5, 6]. The symmetric banded reduction from band to tridiagonal consists of memory bounded operations, and does not scale well. The objective of this paper is to develop an efficient parallel eigensolver for symmetric banded matrices *without using tridiagonalization*. The main tools that we used are the classical banded DC (BDC) algorithm [2, 19] and rank-structured matrix techniques.

The main advantage of BDC is that the *tridiagonalization* step is avoided, which consists of memory-bound BLAS-2 operations. The main problem of BDC is that it requires more floating-point operations than the classic tridiagonal DC algorithm such as implemented in LAPACK and ELPA [38]. The worst complexity of BDC increases from $O(n^3)$ to $O(n^3 r)$ [2, 23], where n is the dimension of matrix and r is the semibandwidth. The main computation part lies in computing the eigenvectors via matrix-matrix multiplications (MMM) in $O(n^3)$ flops. Thus, BDC requires r times more MMM than tridiagonal DC implemented in ELPA. That explains why BDC is not widely used in practice. In this work, we use a parallel structured matrix multiplication algorithm (PSMMA) [35] to reduce the computational cost of BDC.

We first review some important works related with BDC. The BDC algorithm for symmetric eigenvalue problems is well-studied, and it boils down to computing the eigendecomposition of a diagonal matrix plus a rank-r modification, where r is the semibandwidth of matrix. Two methods were proposed by Arbenz in [2]. The first method turns the rank-r modification into a small $r \times r$ eigenproblem, whose eigenvalues are computed by a bisection-type algorithm, and the eigenvectors are suggested to be computed via inverse iteration. Unfortunately, this approach was found to suffer from numerical stability issues for eigenvector computations. The second method considers the rank-r modification as a sequence of rank-one modifications, which is numerically stable but requires more flops than the first one. Most works are based on the second approach since it's numerically stable. For simplicity, the second method will be denoted by BDC in the following sections. Later, Gansterer *et al.* [19, 20] proposed a similar block-tridiagonal DC algorithm for block-tridiagonal matrices. Bai and Ward [4] presented a distributed-memory parallel implementation of BDC, and their implementation was named PBDnC. Haidar, Ltaief and Dongarra [28] implemented BDC by using tile algorithms via a dynamic runtime system for multicore architectures. The package EigenExa [29] also implements BDC, and it fixes the semibandwidth to be 2. ELPA [38] has a routine for solving the banded symmetric eigenvalue problem, which is based on tridiagonalization, the banded matrix is firstly reduced to its tridiagonal form. ELPA requires the block size n_b for data redistribution to be equal to the semibandwidth r. Our implementation is similar to PBDnC and EigenExa, but we further use rank-structured matrix techniques to reduce computational cost.

Rank-structured matrices include semiseparable and quasiseparable [16, 46], $\mathcal{H}$ [26], $\mathcal{H}^2$ [27], SSS [10] and HSS [9] matrices, which are recently playing a very important role in the design of fast algorithms [50, 25, 46]. In a previous work, BDC was accelerated by using HSS matrices on shared memory platforms [34]. Compared with the classical tridiagonal DC in LAPACK, the accelerated BDC algorithm [34] can be much faster for matrices with narrow bandwidths or many deflations. In this paper, we similarly accelerate BDC on *distributed-memory* parallel platforms. Instead of using HSS matrices, we in this paper use PSMMA, which has been introduced recently in [35]. The process of PSMMA is similar to Cannon [8] and Fox [17] algorithms. PSMMA can be seen as a structured version of PUMMA [11] but has *three* advantages over PUMMA. One advantage is that PSMMA constructs local submatrices by using generators without any communication and thus the *communication* cost is reduced. Another is that PSMMA combines with low-rank approximations and therefore the *computation* cost is also reduced. The third is that PSMMA requires less workspace and the size of local matrix multiplications is also larger than PUMMA. The details are included in section 3.

By incorporating PSMMA into the distributed parallel BDC algorithm [19], a parallel banded structured DC (PBSDC) algorithm is proposed in section 3.2. For matrices with few deflations, PBSDC can be much faster than that without using PSMMA since the computation cost can be reduced a lot. For matrices with small bandwidths and many deflations, PBSDC can be 4–163 times faster than the tridagonlization based DC algorithm in ELPA. The performance of ELPA highly depends on the block size parameter n_b which is used for data distribution. The banded eigensolver in ELPA requires r and n_b to be equal. When r is large, PBSDC can be slower than ELPA since PBSDC will requires more flops. Therefore, PBSDC is unfortunately only suitable for matrices with narrow semibandwidths, and it can not be used for accelerating ELPA or ScaLAPACK for the dense eigenvalue problems since n_b is usually chosen to be around 64 in ScaLAPACK and ELPA.

In summary, our main contributions of this paper are as follows.

1. A parallel banded structured DC (PBSDC) algorithm is proposed for the eigenvalue problems on distributed-memory parallel machines, which exploits the low-rank structure and can be much faster than BDC.

2. Many numerical tests have been done on a supercomputer to compare PBSDC with BDC and ELPA. It is interesting to see that PBSDC can be 4–163 times faster than ELPA for matrices with narrow semibandwidths. PBSDC becomes slower than ELPA when semibandwidths are large.

Thus, PSMMA can makes BDC faster than ELPA for narrow banded symmetric matrices, but does not helps for matrices with relatively large semibandwidths.

The remaining sections of this paper are organized as follows. Section 2 introduces the classical banded DC algorithm and shows how it can be accelerated by using rank-structured matrix techniques. Section 3 introduces the parallel structured matrix multiplication algorithm (PSMMA) and the implementation details of PBSDC are shown in section 3.2. Section 4 includes all the numerical results, and conclusions and future works are summarized in section 5.

2 Preliminaries

In this section, we first review the process of classical direct methods for the symmetric banded eigenvalue problems and the process of classical banded DC algorithm [2] in the first two subsections. Some related works are introduced in section 2.3, which all exploit the off-diagonally low-rank structure of symmetric banded matrices.

2.1 Classical direct method for symmetric banded matrices

In this subsection, we briefly introduce the classical direct method for solving a standard eigenvalue problem,

$$AX = X\Lambda,$$

where A is a $n \times n$ symmetric banded matrix, X is the $n \times n$ eigenvector matrix and $\Lambda = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$ is a diagonal matrix and its diagonal entries are eigenvalues.

The classical direct method is based on symmetric bandwidth reduction, and it consists of *three* steps. The input banded matrix is first reduced to tridiagonal form through sequences of Householder transformations, and then compute the eigendecomposition of that tridiagonal matrix, and finally the eigenvectors are computed via *backtranform*. Figure 1 shows the main process of direct eigensolvers for symmetric banded matrices. It consists of the following three steps.

(I) Reduce the banded matrix D to tridiagonal form,

$$T = VDV^T, \tag{1}$$

where V is an orthonormal matrix. This is done via the *bulge chasing* procedure [40, 6], and the operations are memory-bounded.

(II) Solve the tridiagonal eigenvalue problem:

$$T\widehat{Y} = \widehat{Y}\Lambda, \tag{2}$$

which can be solved by QR [18], DC [12, 24, 45], Multiple Relatively Robust Representation (MRRR) [15, 49, 41]. ELPA implements an efficient DC algorithm [3].

(III) Backtransform to obtain the eigenvectors of the banded matrix,

$$\widehat{X} = V^T\widehat{Y}. \tag{3}$$

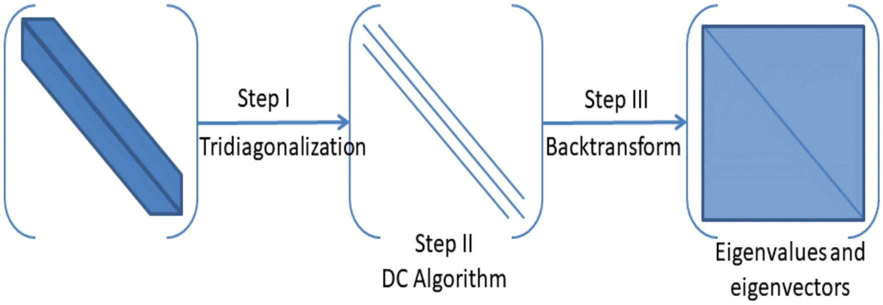

Figure 1: The main process of classical DC algorithm for symmetric banded matrices.

The main drawback of this approach is that it is based on tridiagonalization which is hard is to scale to many processes. For banded matrices with size of around $10,000$, the total time of classical direct method usually stops decreasing when using more than 1024 processes [33].

2.2 BDC for the banded symmetric eigenproblem

In this subsection, we briefly introduce the banded DC algorithm for the symmetric banded eigenvalue problem. For additional details, we refer to works [2, 28, 20]. The divide-and-conquer algorithm consists of three stages: (a) divide a big problem into small subproblems; (b) solve small problems; (c) synthesize the solution of small problems.

Assume that $A \in \mathbb{R}^{n\times n}$ is a symmetric banded matrix, with semibandwidth (the number of nonzero diagonals above or below the main diagonal) equal to r. Partition A as

$$A = \begin{bmatrix} B_1 & C_1^T \\ C_1 & B_2 \end{bmatrix}, \tag{4}$$

where C_1 is a matrix of all zeros except for the $r \times r$ top-right block which is upper triangular. Furthermore, assume the compact SVD of C_1 is $X\Sigma Y^T$, with $\Sigma = \text{diag}(\sigma_i)$ of size $r \times r$. Then, A can be split into the following form,

$$A = \begin{bmatrix} \hat{B}_1 & 0 \\ 0 & \hat{B}_2 \end{bmatrix} + Z\Sigma Z^T, \tag{5}$$

where $\hat{B}_1 = B_1 - Y\Sigma Y^T$, $\hat{B}_2 = B_2 - X\Sigma X^T$, and $Z = \begin{bmatrix} Y \\ X \end{bmatrix}$.

Solve the subproblems independently, $\hat{B}_i = Q_i D_i Q_i^T, i = 1, 2$, and substitute them into (5),

$$\begin{aligned} A &= \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix} + \sum_{i=1}^{r} \sigma_i z_i z_i^T \\ &= Q^{(0)} D Q^{(0)T} + \sum_{i=1}^{r} \sigma_i z_i z_i^T \\ &= Q^{(0)} \{ D + \sigma_1 u_1 u_1^T + \sum_{i=2}^{r} \sigma_i u_i u_i^T \} Q^{(0)T}, \end{aligned} \tag{6}$$

where $u_i = Q^{(0)T} z_i$, $i = 1, \ldots, r$. Thus, the computation is reduced to a sequence of r rank-one updates.

As is well-known, the eigenvalues of a diagonal plus rank-one perturbation can be computed by solving a secular equation, and the corresponding eigenvectors can be expressed explicitly, see Theorem 2.1 in Reference [12] Lemma 5.2 in Reference [14] and also References [7, 48]. For completeness, we restate this result in the following theorem.

Theorem 1 (Bunch, Nielsen, and Sorensen [7]) *Assume that $D = diag(d_1, \ldots, d_n)$ such that $d_1 < d_2 < \cdots < d_n$, and $u \in \mathbb{R}^n$ is a vector and a scalar $\rho > 0$. Then, the eigenvector corresponding to λ_i, an eigenvalue of $M = D + \rho uu^T$, is*

$$q_i = \left(\tfrac{u_1}{d_1 - \lambda_i}, \ldots, \tfrac{u_n}{d_n - \lambda_i} \right)^T \Big/ \sqrt{\sum_{j=1}^{n} \frac{u_j^2}{(d_j - \lambda_i)^2}}, \tag{7}$$

and the eigenvalues of M satisfy

$$d_1 < \lambda_1 < d_2 < \lambda_2 < \cdots < d_n < \lambda_n. \tag{8}$$

The eigenvectors of $D + \sigma_1 u_1 u_1^T = Q^{(1)} D^{(1)} Q^{(1)T}$ have off-diagonally low-rank structure and $Q^{(1)}$ is a Cauchy-like matrix. Therefore, $Q^{(1)}$ can be approximated by a rank-structured matrix accurately. In a previous work [34], $Q^{(1)}$ is approximated by an HSS matrix $Q_H (\approx Q^{(1)})$, and the products of Q_H and $Q^{(0)}$ can be done by using the fast HSS matrix multiplication algorithms. Similarly, repeating the process r times, we obtain the eigendecomposition of $D + \sum_{i=1}^{r} \sigma_i u_i u_i^T$ in (6) approximately.

In this paper we extend this technique to distributed-memory parallel platforms. As in the case of tridiagonal DC [35], we can use a structured parallel matrix multiplication algorithm (PSMMA) instead of using HSS matrix algorithms. We find that PSMMA has better scalability than HSS for rank-structured Cauchy-like matrices [32]. By incorporating PSMMA into BDC, we propose a parallel structured banded DC (PBSDC) algorithm. In the next section, we briefly introduce PSMMA and describe the implementation details of PBSDC, which is based on ScaLAPACK, and similar to PBDnC [4] and EigenExa [29].

2.3 Related works

There have been many works that exploit the off-diagonally low-rank property of some matrices arising in the symmetric (banded) eigenvalue problems. Vogel *et al.* [47] propose a superfast divide-and-conquer method for a class of symmetric matrices with off-diagonal low-rank structure, which can compute all the eigenvalues and eigenvectors but in a factorized form. In some applications, the eigenvectors are required explicitly. In this paper, we focus on the case of computing eigenvectors explicitly. Kressner and Susnjara [30] present a fast algorithm for computing spectral projectors of banded matrices by combining QDWH with HODLR [1], and a structured spectral DC algorithm is further proposed for computing all the eigenpairs of a symmetric banded matrix [44]. Although these two works exploit the low-rank structure, they are not built on BDC. As opposed to previous works, we are interested in the distributed-memory parallel case in this work. For distributed-memory machines, a parallel hybrid tridiagonal DC algorithm was proposed in [32] by using STRUMPACK (STRUctured Matrices PACKage) [43], which provides some distributed-memory parallel HSS algorithms. Recently, a parallel tridiagonal DC algorithm is proposed in [35] by using PSMMA, which scales up to 4096 processes and has better scalability than that [32] using STRUMPACK. We extend the techniques used in [35] to the symmetric banded matrices, and propose a parallel BDC algorithm. For BDC with rank-structured accelerations, there are no results for distributed-memory parallel machines in previous works.

3 The PBSDC Algorithm

In this section, we first introduce some parallel matrix multiplication algorithms briefly, which compute $C = A \times B$, and PSMMA is included in section 3.1. Then, the implementation details of PBSDC based on PSMMA are introduced in section 3.2.

3.1 PSMMA for structured matrices

Cannon's algorithm [8] was the first efficient algorithm for parallel matrix multiplication providing theoretically optimal communication cost. However, it requires the process grid to be square, which limits its practical usage, and Fox's algorithm [17] has the same problem. Both Cannon and Fox algorithms were initially based on the block data distribution (BDD) format. The PUMMA algorithm [11] is a generalized Fox algorithm, and it works for a general $p \times q$ processor grid. PUMMA was designed for ScaLAPACK and used the block cyclic data distribution (BCDD) form. PSMMA is designed for structured matrix multiplications, and it works for general process grids and uses the BCDD form.

We introduce PSMMA for computing $C = A \times B$, where $A \in \mathbb{R}^{m\times k}$ is a general matrix, $B \in \mathbb{R}^{k\times n}$ is a structured matrix. By 'structured matrix' we mean a matrix whose entries can be expressed by using $O(n)$ parameters, where n is the dimension of matrix, and these parameters are called *generators*. The case that A is a structured matrix and B is a general matrix is similar. When both A and B are structured matrices, all operations can be performed locally without any communication. In the following sections, we assume A is stored in BCDD and B is represented by its generators.

Assume that matrix A has M block rows and K block columns, and matrix B has K block rows and N block columns. Block (I, J) of C is then computed by

$$C(I,J) = \sum_{\ell=0}^{K-1} A(I,\ell)\cdot B(\ell,J), \tag{9}$$

where $I = 0, 1, \ldots, M-1$, $J = 0, 1, \ldots, N-1$. Fig. 2 shows a 6×6 block matrix stored in a 2×3 process grid. It is easy to see that the matrix in BCDD form can be obtained from the original matrix by row and column block permutations.

Since the matrix B can be represented by its generators, any submatrices of B can be formed locally. This fact enables us to treat the submatrices of A in each process as a whole continuous block, and we only need to construct the proper submatrices of B correctly. After discovering this fact, the implementation of PSMMA becomes very natural and simple, just following the equation (9). The whole procedure of PSMMA is shown in Algorithm 1.

To illustrate the algorithm from the process point of view, we show how the submatrices of C stored at process P_0 (located at position $(0, 0)$ of the process grid) are computed for the matrix shown in the left subfigure of Figure 2. This consists of three steps, and the process is depicted in Figure 3. The column indices of B are fixed, and its row indices are determined by the column indices of A. After each step, matrix A would be shifted leftward, and the local matrix A on process P_0 is updated by the matrix on process P_1. As shown in the right subfigure of Figure 2, the local matrix A of process P_0 at step 1 is updated by that of process P_1, which is located at the right of process P_0. After process P_0 has received all the A submatrices from all other processes, the algorithm finishes.

Algorithm 1 works for matrices both in BCDD and BDD form. It only depends on the distribution of A to determine the row indices of local matrix B. In step 3(b) of Algorithm 1, we construct a low rank approximation only when the local matrix B is probably low rank. For the banded DC algorithm in section 2, matrix B is a Cauchy-like matrix and we can check whether the intersection of *CIndex* and *RIndex* is empty or not. If the intersection is empty, the B submatrix is probably numerically low-rank. Otherwise, the B submatrix is probably

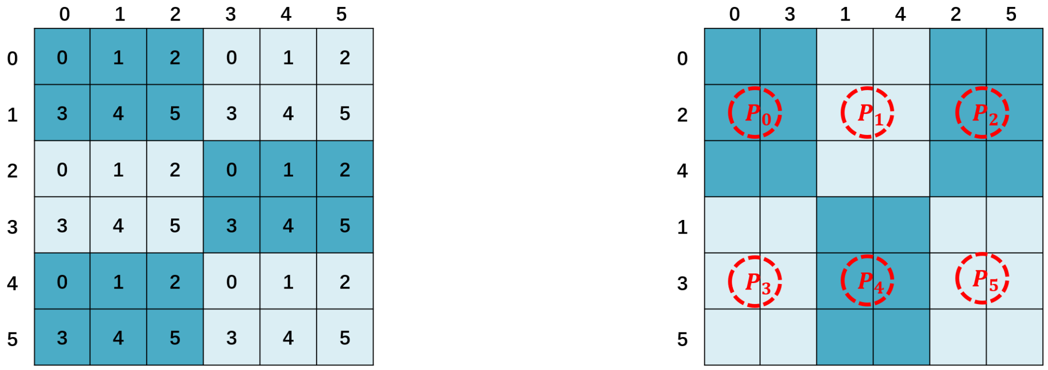

Figure 2: A matrix with 6×6 blocks is distributed over a 2×3 process grid. (left) Matrix point-of-view; (right) Process point-of-view.

Algorithm 1 PSMMA for a structured matrix B

```
Input: $A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{k \times n}$, where $A$ is distributed over a $p \times q$ process grid, $B$ is a structured matrix and all processes have a copy of its generators;
Output: $C = A \times B$;
Each process constructs the column indices ($CIndex$) of matrix $B$ based on its process column in the process grid;
Set $C = 0$.
do $\ell = 0, q - 1$
    (a) For each process $(i, j)$, construct the row indices ($RIndex$) of matrix $A$ based on the process column mod $(j + \ell, q)$;
    (b) Calculate the required $B$ subblock $B(RIndex, CIndex)$, and construct its low-rank approximation (if needed) by using its generators, $B \approx U_B V_B$;
    (c) Multiply the copied $A$ subblock with the currently residing $B$ subblock: $C = C + (A \cdot U_B) \cdot V_B$;
    (d) Shift matrix $A$ leftward cyclically along each process row;
end do
```

full rank. For Cauchy-like matrices in our problem, we use SRRSC [31] to construct a low-rank approximation to matrix B.

3.2 The PBSDC algorithm

In this subsection, we show the details of our implementation based on ScaLAPACK. Our routines follow the scheme of ScaLAPACK routine PDSTEDC, and we modify it and its dependent routines. EigenExa [29] is also written based on the same guideline. The main point is to change the tridiagonal matrix to a banded matrix. The banded DC algorithm previously proposed for shared memory platforms [34] is also implemented similarly, which is based on LAPACK routines.

The parallel strategy used in ScaLAPACK is data parallelism. Data parallelism distributes data evenly to all processors and invokes all relevant processors to work on the same task as the algorithm proceeds [4]. So does EigenExa [29] and our PBSDC. In contrast, PBDnC [4] is based on a mixed data/task parallel strategy. Note that task parallelism assigns each processor to a different task working simultaneously whenever possible. However, this approach requires to redistribute matrices from one subgrid to the supergrid (via communication), and may have workload imbalance problem.

We assume that $A \in \mathbb{R}^{n \times n}$ is a symmetric banded matrix with semibandwidth r, and its diagonal entries are stored in a vector D of length n, and its off-diagonal entries are stored in a tall matrix $E \in \mathbb{R}^{n \times r}$. In

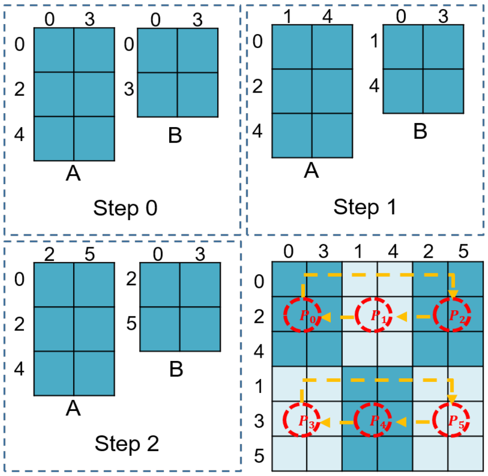

Figure 3: The process for computing the submatrices of C located at process $(0, 0)$.

our implementation, the routine `mpdsbedc` is the starting routine, which is modified from ScaLAPACK routine `PDSTEDC`. Their input parameters are also similar and the difference is that E is now a tall-skinny matrix instead of a vector. The routine `mpdsbedc` first checks the size of n and if n is smaller than n_b (the block size for cyclic distribution), or the number of processes $n_p = 1$, it calls the LAPACK routine `DSYEVD` after some preparations. Otherwise, `mpdsbedc` calls the main computational routine `mpdlaed0`.

The routine `mpdlaed0` is modified from ScaLAPACK routine `PDLAED0`. It consists of the three stages of DC algorithm: (a) It divides the whole problem into some subproblems and constructs the divide-and-conquer tree, shown in Figure 4. It computes the SVD of off-diagonal blocks $C_i = X_i \Sigma_i Y_i^T$, and constructs the subproblems. Assume that there are q subproblems,

$$\begin{aligned} \hat{B}_1 &= B_1 - Y_1 \Sigma_1 Y_1^T, \\ \hat{B}_i &= B_i - X_{i-1} \Sigma_{i-1} X_{i-1}^T - Y_i \Sigma_i Y_i^T, \text{ for } 2 \le i \le q-1, \\ \hat{B}_q &= B_q - X_{q-1} \Sigma_{q-1} X_{q-1}^T. \end{aligned}$$

(b) Each subproblem is solved by calling LAPACK routine `DSYEVD`, which is designed for symmetric dense matrices. There is another routine called `DSBEVD` that is designed for symmetric banded matrices. The reason why we do not use `DSBEVD` is that it is usually slower than `DSYEVD`. (c) It updates the Z vectors, see equation (6), by calling routine `mpdlaedz`, which is modified from ScaLAPACK routine `PDLAEDZ`. The difference from `PDLAEDZ` is that `mpdlaedz` updates several vectors instead of just one vector. Finally, the merged eigenvalue problem is solved by `mpdlaed1`, which recursively computes the eigendecomposition of matrix $D + \sum_{i=1}^{r} \sigma_i u_i u_i^T$. It is done by using r rank-one updates.

The routine `mpdlaed1` is modified from ScaLAPACK routine `PDLAED1`, and it calls `mpdlaed2` to perform deflations and `mpdlaed3` to solve secular equations and compute the eigenvectors of a diagonal matrix with rank-one perturbation. The routines `mpdlaed2` and `mpdlaed3` are modified from ScaLAPACK routines `PDLAED2` and `PDLAED3`, respectively. `PDLAED3` originally uses general matrix multiplication routine `PDGEMM` to update eigenvectors. We use PSMMA to update eigenvectors in our PBSDC algorithm. To combine with PSMMA, we need to further modify `mpdlaed2` and `mpdlaed3`. Note that PBSDC becomes the classic parallel BDC algorithm without using PSMMA.

The excellent performance of DC algorithm is partially due to *deflation* [7, 12], which happens in two cases. If the entry z_i of z are negligible or zero, the corresponding $(\lambda_i, \hat{q}_i)$ is already an eigenpair of $D + \sigma_k z z^T$. Similarly, if two eigenvalues in D are identical then one entry of z can be transformed to zero by applying a sequence of plane rotations. About the deflation process, we refer the interested readers to [12, 22]. The functionality of `mpdlaed2` is to perform deflations [12]. In `mpdlaed2`, all the deflated eigenvalues are moved to the end of D by a permutation matrix, and so are the corresponding eigenvectors. After some deflations, (6) reduces to

$$A = Q^{(0)} \left\{ (GP) \begin{pmatrix} \bar{D} + b_k \bar{z} \bar{z}^T & \\ & \bar{D}_d \end{pmatrix} (GP)^T + \sum_{i=2}^{r} \sigma_i u_i u_i^T \right\} Q^{(0)T}, \tag{10}$$

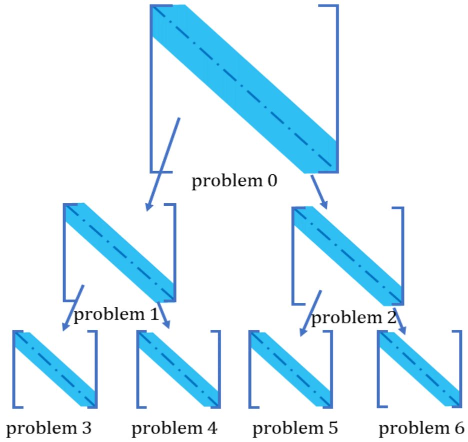

Figure 4: The divide-and-conquer tree for the DC algorithm.

where G is the product of all rotations, P is a permutation matrix, and $\bar{D}_d$ are the deflated eigenvalues. Assume that $\bar{D} + b_k \bar{z}\bar{z}^T = \widehat{Q}\Lambda\widehat{Q}^T$, the eigenvectors of A are updated as

$$Q^{(1)} = Q^{(0)}(GP)\begin{pmatrix} \widehat{Q} & \\ & I_d \end{pmatrix} = \left[\begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} GP\right]\begin{pmatrix} \widehat{Q} & \\ & I_d \end{pmatrix}. \tag{11}$$

To improve efficiency, Gu [21] suggested a permutation strategy, which has been used in ScaLAPACK. The matrix in square brackets is permuted as $\begin{pmatrix} Q_{11} & Q_{12} & 0 & Q_{14} \\ 0 & Q_{22} & Q_{23} & Q_{24} \end{pmatrix}$, and complexity can be reduced by exploiting its sparsity structure. By Theorem 1, we know that $\widehat{Q}$ is a rank-structured Cauchy-like matrix. When the size of matrix $\widehat{Q}$ is large, we can use PSMMA to update the eigenvectors instead of using Gu's strategy, since it will be faster. PSMMA and Gu's strategy cannot be used together since the permutations would destroy the rank-structure of $\widehat{Q}$. Therefore, we detect the size of matrix $\widehat{Q}$ and when it is large enough, we use PSMMA instead of using Gu's permutation strategy.

When the PSMMA algorithm is chosen, `mpdlaed1` calls a new routine `mpdlaed3a` which is similar to `mpdlaed3` and computes the generators of $\widehat{Q}$, and passes them to PSMMA to update the eigenvectors via structured matrix-matrix multiplications.

Remark 1 *To keep the orthogonality of $\widehat{Q}$ (see equation (7)), $d_i - \lambda_j$ must be computed by*

$$d_i - \lambda_j = \begin{cases} (d_i - d_j) - \gamma_j & \text{if } i \le j \\ (d_i - d_{j+1}) + \mu_j & \text{if } i > j \end{cases}, \tag{12}$$

where $\gamma_i = \lambda_i - d_i$ (the distance between λ_i and d_i), and $\mu_i = d_{i+1} - \lambda_i$ (the distance between λ_i and d_{i+1}), which can be returned by calling the LAPACK routine `DLAED4`. In our implementation, $\widehat{Q}$ is represented by using five *generators, $\{d_i\}$, $\{\gamma_i\}$, $\{\mu_i\}$, $\{u_i\}$ and $\{v_i\}$.*

Remark 2 *The main difference of PBSDC with the classical parallel BDC (PBDC) is that PBSDC further uses PSMMA to update eigenvectors instead of using `PDGEMM`.*

4 Numerical results

Numerical results are obtained on the Tianhe-2 supercomputer [37, 36], located in Guangzhou, China. Each compute node is equipped with two Intel Xeon E5-2692 CPUs and our experiments only use CPU cores. For all these numerical experiments, we only used plain MPI, run 24 MPI processes per node in principle, and one process per core. The details of compute nodes are shown in Table 1.

Table 1: The test platform and environment of one node.

Items	Values
2*CPU	Intel Xeon CPU E5-2692 v2@2.2GHz
Memory size	64GB (DDR3)
Operating System	Linux 3.10.0
Complier	Intel ifort 2013_sp1.2.144
Optimization	-O3 -mavx

Example 1 In this example we use some matrices from LAPACK testing, whose eigenvalues are illustrated in Table 2. These matrices have been used in some previous works [28, 34, 42]. In our experiments, the parameter k is set to 10^6, and the dimension of matrix is $n = 20,000$. We use the LAPACK routine `DLATMS` to generate some symmetric banded matrices, and then store them in sparse form. The diagonal elements are stored in a vector D of length n, and the upper nonzero entries are stored in another matrix E of size $n \times r$, where r is the semibandwidth of the banded matrix.

Since matrices of Types 1 and 2 only have two distinct eigenvalues and many eigenvalues are multiple, they would produce many deflations, nearly 100%. This is very good for PBSDC since there are no large matrix-matrix multiplications for PBSDC in this case and it does not require extra flops. The percentage of deflations for Type 3 is about 50% and for Type 4 about 20%. Since Type 3 and Type 4 represent the general case, we use these two types of matrices to present the results of PBSDC. For a random symmetric banded matrix with Gaussian nonzero entries, the percentage of eigenvalues computed by deflation is also about $20\% - 50\%$.

Table 2: Some matrices from LAPACK testing

Type	Description
Type 1	$\lambda_1 = 1, \lambda_i = \frac{1}{k}, i = 2, 3, \dots, n$
Type 2	$\lambda_i = 1, i = 2, 3, \dots, n-1, \lambda_n = \frac{1}{k}$
Type 3	$\lambda_i = k^{-(\frac{i-1}{n-1})}, i = 2, 3, \dots, n$
Type 4	$\lambda_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{k}), i = 2, 3, \dots, n$

We let r be 5 and 15, and the results are shown in left and right subfigures of Figure 5, respectively. To the best of authors' knowledge, ELPA is the only open source parallel distributed-memory package that can compute the full eigendecomposition of a general banded symmetric matrix. Thus, we compare PBSDC with ELPA (version 2018.11.001). ELPA provides an interface to compute the eigendecomposition of symmetric banded matrices, but requires n_b to be equal to r. For PBSDC, we let n_b be 64.

From the results of Figure 5, we can see that PBSDC can be much faster than ELPA for matrices with small bandwidths and many deflations. However, PBSDC becomes slower than ELPA when the bandwidth is large, especially when there are few deflations, which is verified by the right subfigure of Figure 5. For matrices of Type 4 with semibandwidth $r = 15$, PBSDC becomes slower than ELPA when using more than 256 processes. For matrices of Type 3 with $r = 15$, PBSDC is competitive with ELPA and becomes slower than ELPA when using more than 4096 processes.

The results also show that the scalability of PBSDC is not as good as ELPA. This is because our implementation is based on ScaLAPACK routines, which only use one whole MPI communicator, while ELPA uses both row and column process communicators. We would like to rewrite our whole codes in the ELPA style and its scalability and performance should be improved. Some future works are summarized in the next section.

Example 2 In this example, we test the weak scalability of PBSDC by using a constructed Toeplitz matrix, which is defined as

- `BandedToeplitz`, real symmetric and with semibandwidth 5, with diagonal entries equal to two and all nonzero off-diagonal entries equal to one.

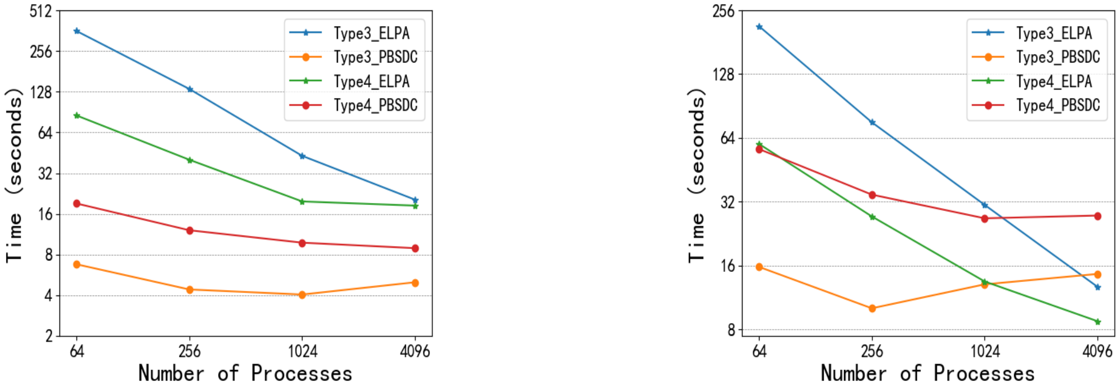

Figure 5: The execution times of PBSDC and ELPA. The left is for $r = 5$ and the right is for $r = 15$.

Matrix `BandedToeplitz` is very difficult for DC algorithm since only 14.6% of eigenvalues are computed by deflation. We test the weak scalability of PBSDC, and use different processes to solve problems of different sizes, and the relations are illustrated in Table 3. For these matrices, ELPA uses $n_b = r = 5$ when using its banded routine interface, and PBSDC chooses $n_b = 64$.

Table 3: The numbers of processes and the sizes of matrices.

n_p	64	256	1024	4096
n	5000	10000	20000	40000

The execution times of PBDC, PBSDC and ELPA are reported in Figure 6. Note that PBDC is based on our implementation of PBSDC without using PSMMA. We use this example to show that PSMMA makes PBSDC much faster than PBDC. From the results, we can see that PBSDC is always faster than ELPA, and PBDC becomes slower than ELPA when the number of processes is larger than 1024. This is because PBSDC uses PSMMA when the size of matrix $\hat{Q}$ is large and therefore requires fewer floating point operations than PBDC. PBSDC is competitive with respect to ELPA for matrices with small bandwidths even if there are very few deflations.

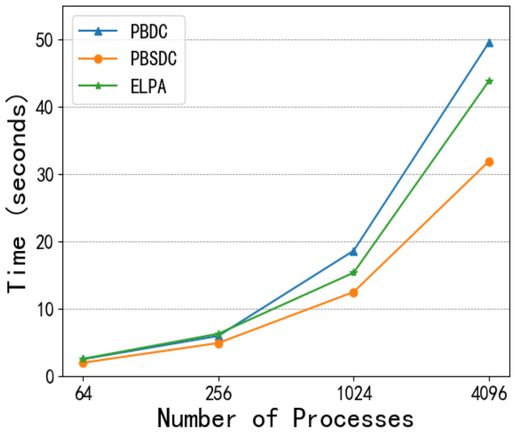

Figure 6: The execution times of PBDC, PBSDC and ELPA for BandedToeplitz.

Example 3 In this example, we use some matrices from real applications to test the proposed PBSDC algorithm, which are obtained from the SuiteSparse matrix collection [13]:

- `LF10000`: real symmetric, with $n = 19,998$ and $r = 3$;
- `linverse`: real symmetric, with $n = 11,999$ and $r = 4$.

We compare PBSDC with ELPA by using an increasing number of MPI processes, and the execution times are reported in Figure 7. For these two matrices, ELPA chooses n_b to be 3 and 4, respectively. For PBSDC, we also let $n_b = 64$. From it, we can see that PBSDC can be 4–163 times faster than the banded algorithm in ELPA.

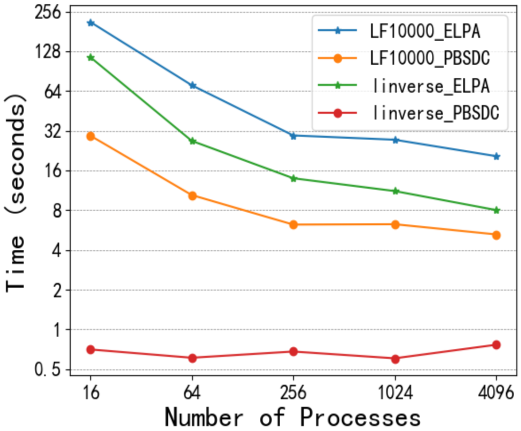

Figure 7: The execution times of PBSDC and ELPA.

Matrix `linverse` is easy for the DC algorithm since nearly 86% of the eigenvalues are computed by deflation. Therefore, PBSDC does not require more floating pointing operations than ELPA for matrix `linverse`. That is why PBSDC can have such a large speedup over the banded algorithm in ELPA. For matrix `LF10000`, only 30% of eigenvalues are computed by deflation, and the largest size of the secular equation is 12,524, and the computation of eigenvectors is accelerated by PSMMA.

Remark 3 *In this example, we choose two matrices with small bandwidths. Most matrices from real applications usually have bandwidths around hundreds or thousands. For matrices with relatively large r (say larger than 20), PBSDC would be slower than ELPA. Our initial aim is to use PBSDC to acclerate ELPA. Unfortunately, PBSDC cannot be used to accelerate ELPA that since ELPA usually chooses $r = n_b = 64$. See the next example.*

Example 4 In this example, we use some random banded matrices to show that the performance of ELPA highly depends on the parameter n_b. The size of the random matrix is 11,999, and its entries are from Gaussian distribution. The semibandwidth r is fixed to be 3 and 5, respectively. As opposed to previous examples which use the banded interface of ELPA, requiring $n_b = r$, in this example we use the dense interface, i.e., treat these banded matrices as dense matrices and apply ELPA to them by choosing different n_b.

Note that ELPA takes *five* steps to solve the eigenvalue problem of a symmetric dense matrix. Step I reduces a dense matrix to banded form. Step II transforms a banded matrix to tridiagonal form. Step III solves the tridiagonal eigenvalue problem via DC. Step IV back transforms to the intermediate, sparse banded form of the problem. Step V back transforms to the standard eigenvalue problem. In this example, we only measure the computation times of Step II-IV and sum them up. We choose n_b to be 5 and 64, respectively, and the results are shown in Table 4.

The results show that the performance of ELPA highly depends on the parameter n_b. Though the matrix is banded with $r = 5$ or 3, ELPA is much faster when treating it as a matrix with larger semibandwidth 64. Note that ELPA requires more flops when choosing $n_b = 64$ than choosing $n_b = 5$ or 3. When $r = 3$, PBSDC is competitive with ELPA with $n_b = 64$. For matrices with $r = 5$, PBSDC would be slower than ELPA with $n_b = 64$ when the number of processes is larger than 16. Therefore, PBSDC is only suitable for matrices with small bandwidths. Some

Table 4: The execution times (in seconds) of Step II-IV of ELPA compared to PBSDC.

n_p	$r=5$			$r=3$		
	$n_b=5$	$n_b=64$	PBSDC	$n_b=3$	$n_b=64$	PBSDC
16	50.51	31.70	**27.27**	76.80	30.60	**6.69**
64	18.27	**9.45**	11.47	27.21	9.17	**3.67**
256	9.81	**4.45**	5.43	12.95	4.25	**2.47**
1024	6.24	**3.39**	4.96	7.55	**3.27**	3.42

improvements are required to make it more useful for practical problems. For ELPA, this example shows that it is better to use a relatively large n_b even if the input is a symmetric banded matrix with very small bandwidth. This result has not been reported in any works before.

5 Conclusions and future works

A parallel banded structured DC (PBSDC) algorithm is proposed for symmetric eigenvalue problems by exploiting rank-structured techniques, which computes the eigenpairs directly without tridiagonalization. The tool that we use is a parallel structured matrix multiplication algorithm, PSMMA [35]. The main goal of this work is to evaluate the performance of PBSDC on distributed-memory machines. For large matrices with few deflations, PBSDC is much faster than the classical PBDC algorithm, see the results in Example 2. PSMMA saves a lot of floating point operations and reduces communication cost comparing with that using `PDGEMM`. We further compare PBSDC with ELPA [38]. Numerical results show that PBSDC can be $4-163\times$ faster than the banded eigensolver in ELPA for matrices with narrow bandwidths and/or many deflations. For matrices with large bandwidths, PBSDC would be slower than ELPA.

Numerical results show that PBSDC is suitable for matrices with small bandwidths and it is numerically stable. Another approach is applying spectrum slicing methods to the symmetric banded matrix such as multiple shift-and-invert Lanczos (MSIL) method [51]. Compared with PBSDC, MSIL may be faster but it may suffer from numerical stability problem such as some eigenvalues may be missing, eigenvectors computed from different slices may not be highly orthogonal, etc. Since spectrum slicing methods are quite different from PBSDC, we will not talk about those methods in this paper. Our implementation is based on ScaLAPACK, and only uses one MPI communicator. Similar to the implementation of ELPA, our codes can be modified by using both row and column process communicators. For matrices with relatively large bandwidths, PBSDC can be combined with successive band reduction [6] to reduce the computational cost, and it will our future work.

Acknowledgement

The authors would like to thank the referees for their valuable comments. This work is supported in part by NSFC (No. 2021YFB0300101, 61902411, 62032023, 12002382, 11275269, 42104078), 173 Program of China (2020-JCJQ-ZD-029), Open Research Fund from State Key Laboratory of High Performance Computing of China (HPCL) (No. 202101-01), Guangdong Natural Science Foundation (2018B030312002), and the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant (No. 2016ZT06D211). Jose E. Roman is supported by the Spanish Agencia Estatal de Investigación (AEI) under project SLEPc-DA (PID2019-107379RB-I00). On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

[1] Ambikasaran, S., Darve, E.: An $\mathcal{O}(n\log n)$ fast direct solver for partial hierarchically semi-separable matrices. Journal of Scientific Computing **57**(3), 477–501 (2013)

[2] Arbenz, P.: Divide-and-conquer algorithms for the bandsymmetric eigenvalue problem. Parallel Comput. **18**, 1105–1128 (1992)

[3] Auckenthaler, T., Blum, V., Bungartz, H.J., Huckle, T., Johanni, R., Krämer, L., Lang, B., Lederer, H., Willems, P.R.: Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. Parallel Computing **37**(12), 783–794 (2011)

[4] Bai, Y.H., Ward, R.C.: A parallel symmetric block-tridiagonal divide-and-conquer algorithm. ACM Trans. Math. Softw. **33**(4), 1–23 (2007)

[5] Bischof, C.H., Lang, B., Sun, X.: A framework for symmetric band reduction. ACM Trans. Math. Softw. **26**(4), 581–601 (2000)

[6] Bischof, C.H., Lang, B., Sun, X.B.: Algorithm 807: The SBR toolbox-software for successive band reduction. ACM Trans. Math. Softw. **26**(4), 602–616 (2000)

[7] Bunch, J.R., Nielsen, C.P., Sorensen, D.C.: Rank one modification of the symmetric eigenproblem. Numer. Math. **31**, 31–48 (1978)

[8] Cannon, L.E.: A cellular computer to implement the kalman filter algorithm. Ph.D. thesis, College of Engineering, Montana State Univesity (1969)

[9] Chandrasekaran, S., Dewilde, P., Gu, M., Lyons, W., Pals, T.: A fast solver for HSS representations via sparse matrices. SIAM J. Matrix Anal. Appl. **29**, 67–81 (2006)

[10] Chandrasekaran, S., Dewilde, P., Gu, M., Pals, T., Sun, X., van der Veen, A.J., White, D.: Some fast algorithms for sequentially semiseparable representation. SIAM J. Matrix Anal. Appl. **27**, 341–364 (2005)

[11] Choi, J., Walker, D.W., Dongarra, J.J.: Pumma: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. Concurr. Comput.: Pract. Exper. **6**(7), 543–570 (1994)

[12] Cuppen, J.J.M.: A divide and conquer method for the symmetric tridiagonal eigenproblem. Numer. Math. **36**, 177–195 (1981)

[13] Davis, T., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011)

[14] Demmel, J.: Applied numerical linear algebra. SIAM, Philadelphia (1997)

[15] Dhillon, I.S.: A new $o(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem. Ph.D. thesis, Computer Science Division, University of California, Berkeley, California (1997)

[16] Eidelman, Y., Gohberg, I.: On a new class of structured matrices. Integral Equations and Operator Theory **34**, 293–324 (1999)

[17] Fox, G.C., Otto, S.W., Hey, A.J.G.: Matrix algorithms on a hypercube I: matrix multiplication. Parallel Comput. **4**(1), 17–31 (1987)

[18] Francis, J.G.: The QR transformation–part 2. The Computer Journal **4**(4), 332–345 (1962)

[19] Gansterer, W.N., Ward, R.C., Muller, R.P.: An extension of the divide-and-conquer method for a class of symmetric block-tridiagonal eigenproblems. ACM Trans. Math. Softw. **28**(1), 45–58 (2002)

[20] Gansterer, W.N., Ward, R.C., Muller, R.P., III, W.A.G.: Computing approximate eigenpairs of symmetric block tridiagonal matrices. SIAM J. Sci. Comput. **25**, 65–85 (2003)

[21] Gu, M.: Studies in numerical linear algebra. Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT (1993)

[22] Gu, M., Eisenstat, S.C.: A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. SIAM J. Matrix Anal. Appl. **15**, 1266–1276 (1994)

[23] Gu, M., Eisenstat, S.C.: A divide-and-conquer algorithm for the bidiagonal SVD. SIAM J. Matrix Anal. Appl. **16**(1), 79–92 (1995)

[24] Gu, M., Eisenstat, S.C.: A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. SIAM J. Matrix Anal. Appl. **16**, 172–191 (1995)

[25] Hackbusch, L.G.W., Khoromskij, B.: Solution of large scale algebraic matrix Riccati equations by use of hierarchical matrices. Computing **70**, 121–165 (2003)

[26] Hackbusch, W.: A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. Computing **62**, 89–108 (1999)

[27] Hackbusch, W., Börm, S.: Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. Computing **69**, 1–35 (2002)

[28] Haidar, A., Ltaief, H., Dongarra, J.: Toward a high performance tile divide and conquer algorithm for the dense symmetric eigenvalue problem. SIAM J. Sci. Comput. **34**(6), C249–C274 (2012)

[29] Imamura, T., Yamada, S., Yoshida, M.: Development of a high-performance eigensolver on a peta-scale next-generation supercomputer system. Prog. Nucl. Sci. Technol. **2**, 643–650 (2011)

[30] Kressner, D., Susnjara, A.: Fast computation of spectral projectors of banded matrices. SIAM J. Matrix Anal. Appl. **38**(3), 984–1009 (2017)

[31] Li, S., Gu, M., Cheng, L., Chi, X., Sun, M.: An accelerated divide-and-conquer algorithm for the bidiagonal SVD problem. SIAM J. Matrix Anal. Appl. **35**(3), 1038–1057 (2014)

[32] Li, S., Rouet, F.H., Liu, J., Huang, C., Gao, X., Chi, X.: An efficient hybrid tridiagonal divide-and-conquer algorithm on distributed memory architectures. J. Comput. Appl. Math. **344**, 512–520 (2018)

[33] Li, S., Wu, X., Roman, J.E., Yuan, Z., Wang, R., Cheng, L.: A parallel direct eigensolver for sequences of hermitian eigenvalue problems with no tridiagonalization. https://arxiv.org/abs/2012.00506 (2020)

[34] Liao, X., Li, S., Cheng, L., Gu, M.: An improved divide-and-conquer algorithm for the banded matrices with narrow bandwidths. Comput. Math. Appl. **71**, 1933–1943 (2016)

[35] Liao, X., Li, S., Lu, Y., Roman, J.E.: A parallel structured divide-and-conquer algorithm for symmetric tridiagonal eigenvalue problems. IEEE Transactions on parallel and distributed systems **32**(2), 367–378 (2021)

[36] Liao, X., Pang, Z., Wang, K., Lu, Y., Xie, M., Xia, J., Dong, D., Suo, G.: High performance interconnect network for Tianhe system. J. Comput. Sci. Tech. **30**(2), 259–272 (2015)

[37] Liao, X., Xiao, L., Yang, C., Lu, Y.: Milkyway-2 supercomputer: system and application. Front. Comput. Sci. **8**(3), 345–356 (2014)

[38] Marek, A., Blum, V., Johanni, R., Havu, V., Lang, B., Auckenthaler, T., Heinecke, A., Bungartz, H., Lederer, H.: The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science. J. Phys.: Condens. Matter **26**, 1–15 (2014)

[39] Martin, R.M.: Electronic structure: Basic theory and practical methods. Cambridge University Press (2008)

[40] Parlett, B.N.: The symmetric eigenvalue problem. SIAM, Philadelphia (1998)

[41] Petschow, M., Peise, E., Bientinesi, P.: High-performance solvers for dense Hermitian eigenproblems. SIAM J. Numer. Anal. **35**, C1–C22 (2013)

[42] Pichon, G., Haidar, A., Faverge, M., Kurzak, J.: Divide and conquer symmetric tridiagonal eigensolver for multicore architectures. 2015 IEEE International Parallel and Distributed Processing Symposium pp. 51–60 (2015)

[43] Rouet, F., Li, X., Ghysels, P., Napov, A.: A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. ACM Trans. Math. Softw. **42**(4), 27:1–35 (2016)

[44] Šušnjara, A., Kressner, D.: A fast spectral divide-and-conquer method for banded matrices. arXiv preprint arXiv:1801.04175 (2018)

[45] Tisseur, F., Dongarra, J.: A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures. SIAM J. Sci. Comput. **20**(6), 2223–2236 (1999)

[46] Vandebril, R., Van Barel, M., Mastronardi, N.: Matrix computations and semiseparable matrices, Volume I: linear systems. Johns Hopkins University Press (2008)

[47] Vogel, J., Xia, J., Cauley, S., Balakrishnan, V.: Superfast divide-and-conquer method and perturbation analysis for structured eigenvalue solutions. SIAM J. Sci. Comput. **38**(3), A1358–A1382 (2016)

[48] Wilkinson, J.: The algebraic eigenvalue problem. Oxford University Press, New York (1965)

[49] Willems, P.R., Lang, B.: A Framework for the MR^3 Algorithm: Theory and Implementation. SIAM Journal on Scientific Computing **35**(2), A740–A766 (2013)

[50] Xia, J., Chandrasekaran, S., Gu, M., Li, X.S.: Superfast multifrontal method for large structured linear systems of equation. SIAM J. Matrix Anal. Appl. **31**, 1382–1411 (2009)

[51] Zhang, H., Smith, B., Sternberg, M., Zapol, P.: SIPs: Shift-and-invert parallel spectral transformations. ACM Transactions on Mathematical Software (TOMS) **33**(2), 1–19 (2007)