Computers and Mathematics with Applications 71 (2016) 1933–1943

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

An improved divide-and-conquer algorithm for the banded matrices with narrow bandwidths

Xiangke Liao [a,b], Shengguo Li [a,*], Lizhi Cheng [c,d], Ming Gu [e]

[a] College of Computer, National University of Defense Technology (NUDT), Changsha 410073, China
[b] Science and Technology on Parallel and Distributed Processing Lab, NUDT, China
[c] State Key Lab of High Performance Computing, NUDT, Changsha 410073, China
[d] College of Science, NUDT, Changsha 410073, China
[e] Department of Mathematics, University of California, Berkeley, CA 47920, USA

ARTICLE INFO

Article history:
Received 18 August 2015
Received in revised form 3 March 2016
Accepted 6 March 2016
Available online 11 April 2016

Keywords:
Divide and conquer algorithm
Banded matrices
SVD
HSS matrices
Eigenvalue problem

ABSTRACT

In this paper we propose a novel divide-and-conquer (DC) algorithm to compute the SVD of banded matrices, and further accelerate it by using rank-structured matrix techniques, especially the hierarchically semiseparable (HSS) matrix. The DC algorithm for the symmetric banded eigenvalue problem can also be accelerated similarly. For matrices with few deflations, the banded DC algorithms require more flops than the classical DC algorithm, and thus they are suitable for narrowly banded matrices. While, if there exist many deflations, the banded DC algorithms can be faster than the classical ones for matrices with relatively large bandwidths. Numerous experiments have been done to test the proposed algorithms. Some of the tested matrices are from construction and some are from real applications. Comparing with the DC algorithm in Intel MKL, our proposed algorithms can be hundreds times faster for matrices with narrow bandwidths or many deflations.

1. Introduction

The eigenvalue and SVD problems are involved in various computational science and engineering areas, such as information retrieval [1], quantum physics [2], chemistry [3], and image restoration. Depending on specific applications, the matrices may be sparse, dense or banded. In this work we focus on the banded case. Many matrices generated from electronic structure calculations can be approximated accurately by banded matrices even though they are dense [2,4]. The banded matrices also appear in the block Lanczos algorithm [5] and in the dense eigenvalue or SVD problems. A dense matrix is first reduced to a banded form and finally to the tridiagonal or bidiagonal form [6,7]. The main objective of this paper is to introduce an efficient *divide-and-conquer algorithm* for computing the SVD of a banded matrix. Unlike classic methods, this new method computes the SVD of a banded matrix directly without reducing it to a bidiagonal form. A similar method was originally proposed in [8] without showing any numerical results. In this paper we revisit it and propose some new techniques to accelerate it, and further show some numerical comparisons with the classical DC algorithms. These techniques can be similarly applied to the symmetric banded eigenvalue problem.

State-of-the-art SVD solvers for banded matrices are based on the bidiagonal reduction strategy, consisting of the following three stages. First, a banded matrix is reduced to an upper bidiagonal form by a sequence of two-sided orthogonal

* Corresponding author.
E-mail addresses: xkliao@nudt.edu.cn (X. Liao), nudtlsg@nudt.edu.cn (S. Li).

http://dx.doi.org/10.1016/j.camwa.2016.03.008
0898-1221/© 2016 Elsevier Ltd. All rights reserved.

transformations [9], and this step is called *bidiagonal reduction*. Second, the bidiagonal SVD problem is solved by any standard method such as DC [8], QR [10] or MRRR [11]. Finally, the singular vectors are computed by accumulating the orthogonal transformations from the bidiagonal reduction, and this process is usually called *back transformation*.

The banded DC (BDC) for the SVD problem introduced in this paper is quite similar to that for the symmetric banded eigenvalue problem [12–14] which avoids the tridiagonalization stage, see the works by Arbenz and coauthors [12,15], Gansterer and coauthors [16,13,17,4], Haidar and coauthors [14], etc. For the eigenvalue problem, it boils down to computing the eigendecomposition of a diagonal matrix with a rank-b modification, where b is the bandwidth. Arbenz [12] proposed two methods to solve this problem. One approach turns the rank-b modification into a $b \times b$ eigenproblem, and the eigenvalues can be computed by bisection-type algorithm, and it is suggested to compute the eigenvectors via inverse iteration. Another approach computes the rank-b modification as a sequence of rank-one modifications. The first approach requires fewer flops than the second one. Unfortunately, there exists no numerically stable implementation for the first approach. All the works [16,13,17,4,14] use the second approach. For the SVD case, it boils down to solving a rank-b updating SVD problem, see (10). Similarly, we compute it as a sequence of rank-one updating SVD problems.

One advantage of banded DC is that the *bidiagonal reduction* and *back transformation* steps are avoided, which are done with memory bandwidth limited BLAS2 routines. While, its disadvantage is that it requires much more floating point operations when the singular vectors are required, the complexity increases from $O(N^3)$ to $O(N^3b)$, see [12,8]. In this work we always assume the singular vectors are desired, and otherwise the classical approach is usually a better choice. Similar to the bidiagonal DC algorithm, the most expensive part of BDC lies in computing the singular vectors via matrix–matrix multiplications (MMM) in $O(N^3)$. BDC needs b times more MMM and solves b times more secular equations than the bidiagonal DC. The good news is that MMM can be performed efficiently by calling highly optimized BLAS libraries and that solving secular equation is relatively cheap, only costs $O(N^2)$ flops. Therefore, when b is small, the banded DC algorithm can be much faster than the classical approach. When b is relatively large, we can first reduce the matrix to a banded matrix with narrower bandwidth via *bulge*chasing and then use BDC to compute its SVD or eigendecomposition. Numerical results in Section 5 show that this strategy can be faster than the classical approach.

Recently, rank-structured matrices are playing a very important role in the design of fast or superfast algorithms. For example, they are involved in designing fast sparse direct solvers [18], solving matrix equations [19], integral equations [20,21] and eigenvalue problems [22,23]. In [23], the authors show that the computations of singular vectors can be accelerated by using HSS matrices. The fact is that the singular vector matrices of a broken arrow matrix can be *off-diagonally* low rank, which can be approximated accurately by HSS or other rank-structured matrices. As is well-known, the HSS matrix multiplication algorithm only costs $O(N^2r)$ instead of $O(N^3)$ flops, where N is the dimension and r is a relatively small number. In this paper, we similarly use this technique to accelerate the banded SVD problem. The symmetric banded DC algorithm is quite related, and we also show how to use the HSS technique to speedup its computations. The HSS matrix techniques make the banded DC algorithms much more efficient than without using them, which are verified by the numerical results in Section 5.

In summary, the contributions of this paper are the following:

- A banded DC algorithm is proposed for the SVD problem, which is similar to that in [8]. Some implementation details are included, and the numerical results show that it is efficient and numerically stable.
- The complexity of banded DC algorithm is analyzed in detail, and it turns out that it requires more flops than the standard DC algorithm for matrices with few deflations. Numerical results also show it is suitable for matrices with narrow bandwidths. For a matrix with relatively large bandwidth, we suggest reducing its bandwidth first via bulge chasing.
- The HSS matrix techniques are used to accelerate the banded DC algorithms, which make them even more efficient. The results for the banded eigenvalue problems are also included.

The paper is organized as follows. In Section 2, we briefly introduce some concepts of rank-structured matrices especially the HSS matrix. In Section 3, we present the banded DC algorithm for the SVD problem and analyze its complexity in detail. Section 4 briefly introduces the banded DC algorithm for the symmetric eigenvalue problem. Some numerical results are reported in Section 5 including results for constructed matrices and matrices from real applications [24].

2. Notation and HSS matrix

The rank-structured matrix computations have been the intensive focus of recent research. A matrix is called *rank-structured* if the ranks of all off-diagonal blocks are relatively small compared to the size of matrix. The HSS matrix [25,26] is an important kind of rank-structured matrices. The other kinds include $\mathcal{H}$ and $\mathcal{H}^2$ matrices [27,28], sequentially semiseparable (SSS) matrices [29], quasi-separable and semiseparable matrices [30,31], etc.

We follow the notation in [32] which is a little different from those used in [26,33]. The HSS matrix is represented by using a binary *tree*. Let $\mathcal{I} = \{1, 2, \ldots, N\}$ and $\mathcal{T}$ be a binary tree in postordering where node i is associated with a contiguous subset t_i of $\mathcal{I}$, which satisfies the following conditions:

- $t_i \cup t_{\mathrm{sib}(i)} = t_{\mathrm{par}(i)}$, where $\mathrm{sib}(i)$ and $\mathrm{par}(i)$ are the sibling and parent of a node i respectively;
- $t_{\mathrm{root}(\mathcal{T})} = \mathcal{I}$, where $\mathrm{root}(\mathcal{T})$ denotes the root of $\mathcal{T}$.
- $i_1 < i_2 < i$, where i_1 and i_2 are respectively the left and right child of parent node i.

Let t_i^c be the set of all indices less than those in t_i and t_i^r be the set of all indices greater than those in t_i, and thus $\mathcal{I}$ is partitioned into three sets, $\mathcal{I} = t_i^c \cup t_i \cup t_i^r$. For each node i, it associates with one *HSS block row* and one *HSS block column*, which are defined as, respectively,

$$H_i^{row} = [A(t_i, t_i^c)\; A(t_i, t_i^r)] \quad \text{and} \quad H_i^{col} = \begin{bmatrix} A(t_i^c, t_i) \\ A(t_i^r, t_i) \end{bmatrix}. \tag{1}$$

The maximum rank of these HSS blocks is called *HSS rank*.

The HSS matrix has two important properties: one is that the off-diagonal blocks are numerically low rank, and the other is the representations of HSS blocks at different levels are nested. Each node i has four *generators*, $\widehat{D}_i$, $\widehat{U}_i$, $\widehat{V}_i$, and $\widehat{B}_i$, and these generators are nested to present the whole matrix A,

$$\begin{aligned} \widehat{D}_i &= A(t_i \times t_i) = \begin{bmatrix} \widehat{D}_{i_1} & \widehat{U}_{i_1} B_{i_1} \widehat{V}_{i_2}^T \\ \widehat{U}_{i_2} B_{i_2} \widehat{V}_{i_1}^T & \widehat{D}_{i_2} \end{bmatrix}, \\ \widehat{U}_i &= \begin{bmatrix} \widehat{U}_{i_1} & \\ & \widehat{U}_{i_2} \end{bmatrix} U_i, \qquad \widehat{V}_i = \begin{bmatrix} \widehat{V}_{i_1} & \\ & \widehat{V}_{i_2} \end{bmatrix} V_i. \end{aligned} \tag{2}$$

For a leaf node i, $\widehat{D}_i = D_i$, $\widehat{U}_i = U_i$, $\widehat{V}_i = V_i$. A 4×4 block HSS matrix is usually written as,

$$A = \begin{bmatrix} \begin{bmatrix} D_1 & U_1 B_1 V_2^T \\ U_2 B_2 V_1^T & D_2 \end{bmatrix} & \widehat{U}_3 B_3 \widehat{V}_6^T \\ \widehat{U}_6 B_6 \widehat{V}_3^T & \begin{bmatrix} D_4 & U_4 B_4 V_5^T \\ U_5 B_5 V_4^T & D_5 \end{bmatrix} \end{bmatrix}, \tag{3}$$

which has been used in [34] and is equivalent to those used in [26,33].

2.1. HSS construction algorithm

For a general matrix with off-diagonally low-rank property, we need to construct an HSS approximation to it before using the fast HSS algorithms. Up to now, there exist many HSS construction algorithms.

A fast HSS construction algorithm is proposed in [25] which uses the truncated SVD [35] to compress the HSS blocks level by level. In [26], it uses the rank revealing QR method [36,37] and follows a postordering tree. Comparing these two methods, the level-by-level approach [25] is good for parallel while it costs more flops, and the postordering tree approach must run serially. The complexity of these methods are $O(N^2 r)$, where r is HSS rank. In [34], a randomized HSS construction algorithm (`RandHSS`) is proposed, which combines random sampling with *interpolative decomposition* (ID), see [38–40]. This method is especially suitable for matrices with fast matrix–vector multiplication algorithms. Its cost can be $O(Nr)$ if the matrix–vector multiplication only costs $O(N)$ flops. A distributed parallel HSS construction algorithm has been proposed in [41].

An HSS construction algorithm for Cauchy-like matrices has been proposed in [42,23], which works on four vectors, the generators of a Cauchy-like matrix and thus requires linear storage. It is shown in [23] that it can be much faster than that using the truncated SVD in [25]. This construction algorithm has been used to solve Toeplitz linear systems [43], the bidiagonal SVD problem [23], and the symmetric tridiagonal eigenvalue problem [32]. In [32], we further use `RandHSS` [34] to accelerate the symmetric tridiagonal eigenvalue problem, and propose a method to estimate the HSS rank. In this paper, we also use it to construct HSS matrix approximations in the banded SVD and symmetric banded eigenvalue problems. The singular vectors or eigenvectors would be updated by using fast HSS matrix multiplication algorithms [44,32]. The complexity of an $N \times N$ HSS matrix multiplying with an $N \times N$ general matrix is $O(N^2 r)$ instead of $O(N^3)$ flops. A *parallel* HSS matrix multiplication algorithm with a general matrix from right is proposed in [32], and its scalability is also analyzed there.

3. The banded DC for SVD

In this section we introduce a novel DC algorithm [8] to compute the SVD of a banded matrix and show how to use HSS matrix techniques to accelerate it. Without any loss of generality, we assume it is an upper banded matrix.

3.1. Partition the matrix

Assume that B is an $N \times (N + b)$ upper banded matrix with semibandwidth b $(1 < b < N)$. We recursively split B into several small subproblems by removing a block row, see the matrix B (4). Note that the original algorithm introduced in [8] is based on the column partition. In this work we use the row partition which is consistent with the bidiagonal DC algorithm implemented in LAPACK and the notation used in [23]. The row and column partition versions are equivalent.

To illustrate the main process, we assume that B is divided into the following two subproblems,

$$B = \left[\begin{array}{cc} B_{11} & \\ \hline B_{21} & B_{22} \\ \hline & B_{32} \end{array}\right], \tag{4}$$

where B_{11} and B_{32} are upper banded with dimensions $K \times (K+b)$ and $L \times (L+b)$ respectively, B_{21} is a $b \times b$ upper triangular matrix, and B_{22} is a $b \times b$ lower triangular matrix, $N = K + b + L$, $K \geq b$, and $L \approx K = \lfloor (N-b)/2 \rfloor$.

In general, B is partitioned into p subproblems and p is usually powers of two. For simplicity we assume $p = 2$.

3.2. Merge the subproblems

Use the classical QR or DC algorithm to compute the SVD of subproblems,

$$B_{11} = U_1 \begin{bmatrix} D_1 & 0 \end{bmatrix} V_1^T, \qquad B_{32} = U_2 \begin{bmatrix} D_2 & 0 \end{bmatrix} V_2^T, \tag{5}$$

which can be solved by calling the LAPACK routine DGESVD or DGESDD.

Then the SVD of B can be written as

$$B = \begin{bmatrix} & U_1 & \\ I & & \\ & & U_2 \end{bmatrix} \begin{bmatrix} Z_2^{(1)} & Z_1^{(1)} & Z_1^{(2)} & Z_2^{(2)} \\ & D_1 & & \\ & & D_2 & 0 \end{bmatrix} \begin{bmatrix} V_1(2)^T & \\ V_1(1)^T & \\ & V_2(1)^T \\ & V_2(2)^T \end{bmatrix}, \tag{6}$$

where $Z^{(1)} = \begin{bmatrix} Z_1^{(1)} & Z_2^{(1)} \end{bmatrix}$ is equal to the product of B_{21} and the last b rows of V_1, $Z^{(2)}$ is equal to the product of B_{22} and the first b rows of V_2, and $V_i(2)^T$ is the submatrix consisted of the last b rows of V_i^T, $i = 1, 2$.

Assume Q is an orthogonal matrix such that $QZ_2^{(1)}$ is upper triangular, and thus

$$Q^T \begin{bmatrix} Z_2^{(1)} & Z_1^{(1)} & Z^{(2)} \end{bmatrix} = \begin{bmatrix} \hat{R} & T_1 & T_2 & T_3 \end{bmatrix}, \tag{7}$$

where $\hat{R}$ is a $b \times b$ upper triangular matrix and $T = \begin{bmatrix} T_1 & T_2 & T_3 \end{bmatrix}$ is usually a dense $b \times N$ matrix. Compute the RQ factorization of $\begin{bmatrix} \hat{R} & T_3 \end{bmatrix}$, and we obtain a $2b \times 2b$ orthogonal matrix G such that

$$\begin{bmatrix} \hat{R} & T_3 \end{bmatrix} \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22}^T \end{bmatrix} = \begin{bmatrix} R & 0 \end{bmatrix}. \tag{8}$$

Substitute (7) and (8) into (6), we get

$$B = \begin{bmatrix} & U_1 & \\ Q & & \\ & & U_2 \end{bmatrix} \begin{bmatrix} R & T_1 & T_2 & 0 \\ & D_1 & & \\ & & D_2 & 0 \end{bmatrix} \begin{bmatrix} G_{11}^T V_1(2)^T & G_{21}^T V_2(2)^T \\ V_1(1)^T & \\ & V_2(1)^T \\ G_{12}^T V_1(2)^T & G_{22} V_2(2)^T \end{bmatrix}, \tag{9}$$

where $V_2 = \begin{bmatrix} V_2(1) & V_2(2) \end{bmatrix}$. As the bidiagonal SVD problem [8,23], the banded SVD problem is boiled down to computing the SVD of a *block broken arrow* matrix,

$$M = \begin{bmatrix} R & T \\ & D \end{bmatrix}, \tag{10}$$

where D is a nonnegative diagonal matrix and its diagonal entries are ordered increasingly. Computing the SVD of (10) is called the updating SVD problems in [45–47].

3.3. The SVD of a block broken arrow matrix

We first consider the last row of the block row $\begin{bmatrix} R & T \end{bmatrix}$, i.e., partition $\begin{bmatrix} R & T \end{bmatrix}$ as

$$\begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} R_1 & T_1 \\ 0 & z \end{bmatrix}, \tag{11}$$

where R_1 is a $(b-1) \times (b-1)$ upper triangular matrix and z is a row vector with dimension $K + L + 1$. Let $U\Omega V^T$ be the SVD of M_1 such that,

$$M_1 = \begin{bmatrix} z_1 & z_{2:K+1} & z_{K+2:K+L+1} \\ & D_1 & \\ & & D_2 \end{bmatrix} = U\Omega V^T. \tag{12}$$

Then (10) can be rewritten as

$$\begin{bmatrix} R_1 & T_1 \\ & U\Omega V^T \end{bmatrix} = \begin{bmatrix} I & \\ & U \end{bmatrix} \begin{bmatrix} R_1 & T_1 V \\ & \Omega \end{bmatrix} \begin{bmatrix} I & \\ & V^T \end{bmatrix}. \tag{13}$$

The matrix in the middle of the right-hand-side of (13) has the same structure as (10). Apply this process b times, the SVD of M (10) is obtained. During each iteration, we need to solve a secular equation and compute two matrix–matrix multiplications to compute the singular vectors. Therefore, banded DC algorithm requires roughly b times more flops than the standard bidiagonal DC algorithm [48].

The SVD of M_1 can be computed by solving a secular equation, which is well-known in the literature, see the Refs. [8,49].

Theorem 3.1 (*Jessup and Sorensen [49]*). *Let $U\Omega V^T$ be the SVD of M_1 with*

$$U = (u_1, \dots, u_N), \qquad \Omega = diag(\omega_1, \dots, \omega_N) \quad \text{and} \quad V = (v_1, \dots, v_N),$$

where $0 < \omega_1 < \cdots < \omega_N$. Then the singular values $\{\omega_j\}_{j=1}^N$ satisfy the interlacing property

$$0 \equiv d_1 < \omega_1 < d_2 < \cdots < d_N < \omega_N < d_N + \|z\|_2,$$

and the secular equation $\mathcal{F}(\omega) = 1 + \sum_{i=1}^N \frac{z_i^2}{d_i^2 - \omega^2} = 0$. The singular vectors satisfy

$$\begin{aligned} u_j &= \left(-1, \frac{d_2 z_2}{d_2^2 - w_j^2}, \dots, \frac{d_N z_N}{d_N^2 - w_j^2}\right)^T \Bigg/ \sqrt{1 + \sum_{i=2}^N \frac{d_i^2 z_i^2}{(d_i^2 - w_j^2)^2}}, \\ v_j &= \left(\frac{z_1}{d_1^2 - w_j^2}, \dots, \frac{z_N}{d_N^2 - w_j^2}\right)^T \Bigg/ \sqrt{\sum_{i=1}^N \frac{z_i^2}{(d_i^2 - w_j^2)^2}}, \end{aligned} \tag{14}$$

where $j = 1, \dots, N$.

As is illustrated in [23], the singular vector matrices U and V of M_1 are off-diagonally low rank, which can be approximated accurately by an HSS matrix. The ranks of off-diagonal blocks are related with the distribution of the singular values [23]. From the experiments we see that the ranks are usually around 50 and rarely larger than 100. Another important fact is that both U and V have Cauchy-like structure, which has been explored by the construction algorithm proposed in [42,23]. The randomized HSS construction algorithm [34] works for any HSS matrix, and has been used for the symmetric tridiagonal eigenvalue problem in [32]. In this work we use the same RandHSS algorithm as used in [32] to accelerate the banded DC algorithms.

After finding an HSS matrix approximation U_H and V_H to U and V in (13) respectively, (10) is approximated by

$$M \approx \begin{bmatrix} R_1 & T_1 \\ & U_H \Omega V_H^T \end{bmatrix} = \begin{bmatrix} I & \\ & U_H \end{bmatrix} \begin{bmatrix} R_1 & T_1 V_H \\ & \Omega \end{bmatrix} \begin{bmatrix} I & \\ & V_H^T \end{bmatrix}. \tag{15}$$

Similarly, applying the process above for b times, the SVD of (10) can be computed approximately as

$$M \approx \begin{bmatrix} I & \\ & U_H \end{bmatrix} \begin{bmatrix} I & \\ & U_H^{(2)} \end{bmatrix} \cdots \begin{bmatrix} I_b & \\ & U_H^{(b)} \end{bmatrix} \widehat{\Omega} \begin{bmatrix} I_b & \\ & V_H^{(b)T} \end{bmatrix} \cdots \begin{bmatrix} I_2 & \\ & V_H^{(2)T} \end{bmatrix} \begin{bmatrix} I & \\ & V_H^T \end{bmatrix}, \tag{16}$$

where $U_H^{(i)}$, $V_H^{(i)}$ are also HSS matrices, and I_i are identity matrices with appropriate dimensions, for $i = 2, \dots, b$. All these multiplications can be done using fast HSS matrix multiplication algorithms, which require $O(n^2 r)$ *flops* instead of $O(n^3)$.

3.4. The complexity analysis

The DC algorithm is organized as a binary tree, which is usually called a *DC tree*. If we follow the DC tree and summarize the cost level by level, the complexity of BDC [8] without using HSS matrices is known to be $O(N^3 b)$. In this subsection, we show the complexity reduces to $O(bN^2\kappa)$ when combining with HSS matrix techniques, where κ is the maximum HSS rank of all the intermediate singular vector matrices. The value of κ may depend on the distribution of singular values, but we find it is usually around 50 and less than 100 in practice. Our analysis is based on the row partition shown in (4).

Note that the complexities of HSS construction and HSS matrix multiplication algorithms [25,44,23] are $O(N^2 r)$ flops, where N is the size of matrix and r is HSS rank, $r \le \kappa$. Assume that there are $p = 2^L$ leaf nodes in the DC tree and it has $L+1$ levels from 0 to L, and the root node is at level 0. Note that only the last node at each level corresponds to a square matrix. For simplicity, we assume the dimensions of all bottom subproblems are $K \times (K + b)$ such that $N = pK + (p-1)b$, where b is the semibandwidth. The submatrix corresponding to the last leaf node is square with dimension $K \times K$. We assume that K and b are in the same order.

1. *Compute the SVD of bottom subproblems.* This step computes the SVD of submatrix corresponding to each leaf node as shown in (5), whose dimension is $K \times (K+b)$ and $K \geq b$. The standard QR or DC algorithm is used, and the complexity is about $O(\frac{28}{3}K^3)$ flops (bidiagonal reduction: $O(\frac{8}{3}K^3)$, the bidiagonal DC algorithm: $O(\frac{8}{3}K^3)$ and backtransform: $O(4K^3)$). Thus, it requires at most $O(p\frac{28}{3}K^3)$ flops.
2. *Merge the subproblems level by level.* This step merges the child nodes of each parent at each level, and traverses the DC tree from bottom to top. Each merging solves a block updating SVD problem, see (13), which consists of the following steps:
   (a) *compute the block vector Z in* (6). Since $Z = \begin{bmatrix} Z^{(1)} & Z^{(2)} \end{bmatrix}$, $Z^{(1)}$ and $Z^{(2)}$ are defined as in (6), the complexity for level ℓ is $O(4b^2(k_\ell + b))$, where k_ℓ is the row dimension of submatrix at current level ℓ. The dimension of Z is $b \times (k_\ell + b)$ or $b \times k_\ell$ for the last node at current level.
   (b) *triangularize the block vector Z.* It first computes an orthogonal matrix Q from the QR factorization of a $b \times b$ matrix and multiplies its transpose with Z, see (7). Its cost is about $O(2b^2(k_\ell + b))$ flops for each node at level ℓ.
   (c) *eliminate T_3 in* (7). The orthogonal matrix G in (8) can computed from a sequence of Householder or Givens transforms. In our implementation, we used the LAPACK routine `dtzrzf`. Since b could be much smaller than k_ℓ, the major operations lie in updating the right singular vectors, see (9). Thus, it requires $O(2 \times 4b^2(k_\ell + b))$ flops.
   (d) *compute the rank-b updating SVD problem.* The SVD of M (10) is computed from b rank-one updating SVD problems, computing the SVD of b *broken arrow* matrices. The SVD of each broken arrow matrix is computed by Theorem 3.1, and the singular vectors are updated by using HSS matrices. It is consisted of the following computation phases:
   - *compute the singular values*. It solves b secular equations and its complexity is in order of $O(bk_\ell^2)$ flops.
   - *update the block vector Z*. After each rank-one updating, the block vector Z should be updated by the right singular vectors, T_1V_H in (15). The complexity is about $O(b(k_\ell^2\kappa + k_\ell\kappa))$ flops, where k_ℓ is the size of the square matrix M.
   - *update the singular vectors*. For each broken arrow matrix, its singular vectors are first approximated by two HSS matrices, and the singular vectors of M are updated by using HSS matrix multiplication algorithms. For simplicity, we assume there are no deflations. For updating the left singular vectors, the dimensions of two involved matrices are $k_\ell \times (k_\ell - i)$ and $(k_\ell - i) \times (k_\ell - i)$ respectively, for $i = 0, \ldots, b-1$. For a $(k_\ell - i) \times (k_\ell - i)$ matrix, its HSS matrix approximation costs $O((k_\ell - i)^2\kappa)$ flops and multiplying a $k_\ell \times (k_\ell - i)$ matrix from right costs $O(k_\ell(k_\ell - i)\kappa)$ flops. Therefore, updating the left singular vectors costs $O(\sum_{i=0}^{b-1}(k_\ell - i)^2\kappa + k_\ell(k_\ell - i)\kappa) = O(2bk_\ell^2\kappa - \frac{3}{2}b(b-1)k_\ell\kappa)$ flops. The cost of updating the singular vectors would be roughly twice of updating the left ones.

   Note that k_ℓ satisfies $k_\ell = 2^{L-\ell}(K+b) - b$. Add the costs at all levels of DC tree, from level $L-1$ to 0, and the cost for the parent nodes is

$$\begin{aligned}
&\sum_{\ell=0}^{L-1} 14b^2(k_\ell + b) + bk_\ell^2 + b(k_\ell^2 + k_\ell)\kappa + 4bk_\ell^2\kappa - 3b(b-1)k_\ell\kappa \\
&\approx \sum_{\ell=0}^{L-1}(5\kappa + 1)bk_\ell^2 - (3b\kappa - 14b - 4\kappa)bk_\ell \\
&\approx \sum_{\ell=1}^{L}(5\kappa + 1)b[(K+b)2^\ell]^2 - (3b\kappa - 14b - 4\kappa)b[(K+b)2^\ell] \\
&\approx \frac{4}{3}bN^2(5\kappa + 1) - 2N(3b^2\kappa - 14b^2 - 4b\kappa)
\end{aligned}$$

   where we used $k_\ell \approx (K+b)2^{L-\ell}$ when ℓ is small.

Therefore, the total cost of HSS-BDC is in order of $O(\frac{4}{3}(5\kappa+1)bN^2)$ flops. Note that the standard DC algorithm requires $O(N^3)$ flops [49,8]. Therefore, it requires $b(5\kappa + 1) < N$ to let HSS-BDC more efficient than the standard DC. Through experiments, we find κ is around 50–100, and thus b is better to be not larger than 20 if $N = 10{,}000$, which is in consistent with the numerical results in Example 1. Though the HSS techniques can significantly reduce the operations of computing singular vectors, we find that most time of HSS-BDC still spends on computing the singular vectors.

There are no routines in LAPACK of computing the SVD of a general banded matrix directly. The LAPACK routine DGESDD implements the standard DC algorithm for general matrices, uses the bidiagonalization approach and costs $O(N^3)$ flops. It first reduces a general matrix to a bidiagonal form, and then uses the classical bidiagonal DC algorithm to compute its SVD. The bidiagonalization-based approach requires fewer flops than BDC, but its drawback is that most operations in the bidiagonalization stage are memory bound, are difficult to use the highly optimized BLAS library and difficult to run in parallel. Most operations of BDC are BLAS3, that explains why BDC can be faster than the bidiagonalization-based algorithm when b is small.

4. The banded DC for SEP

In this section, we briefly introduce the banded DC algorithm for the symmetric eigenvalue problem. For the details, we refer to [12,14,17]. The main purpose is to show how to combine it with HSS matrix techniques.

Table 1
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	$\lambda_i = 1 - (\frac{i-1}{n-1})(1 - \frac{1}{k}),\ i = 2, 3, \dots, n$
Type 5	n random numbers in the range $(\frac{1}{k}, 1)$; their logarithms are uniformly distributed
Type 6	n random numbers from a specified distribution

Assume that A is a symmetric banded matrix, and is partitioned into two parts,

$$A = \begin{bmatrix} B_1 & C_1^T \\ C_1 & B_2 \end{bmatrix}, \tag{17}$$

where C_1 is a $b \times b$ upper triangular matrix. Furthermore, assume the SVD of C_1 is $X\Sigma Y^T$. Then, A can be split into the following form,

$$A = \begin{bmatrix} \hat{B}_1 & 0 \\ 0 & \hat{B}_2 \end{bmatrix} + Z\Sigma Z^T, \tag{18}$$

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 (18),

$$A = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix}\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix}\begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix} + \sum_{i=1}^{b} \sigma_i z_i z_i^T = Q^{(0)} D Q^{(0)T} + \sum_{i=1}^{b} \sigma_i z_i z_i^T$$
$$= Q^{(0)} \left\{ D + \sigma_1 u_1 u_1^T + \sum_{i=2}^{b} \sigma_i u_i u_i^T \right\} Q^{(0)T}, \tag{19}$$

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

As is well-known, the eigenvalues of diagonal plus rank-one perturbation can be computed by solving a secular equation, and its eigenvectors can be expressed explicitly, see Theorem 2.1 in [48], Lemma 5.2 in [9], and also [50,51]. As for the SVD case, the eigenvectors of $D + \sigma_1 u_1 u_1^T = Q_1 D_1 Q_1^T$ also have off-diagonally low-rank structure if the diagonal entries of D are ordered increasingly. Therefore, Q_1 can be approximated by an HSS matrix Q_H ($\approx Q_1$) accurately, and the products of Q_H, $Q^{(0)}$ and $\{u_i\}_{i=2}^{b}$ can be done by using the fast HSS matrix multiplication algorithms. Similarly, apply the process for b times and we obtain the eigendecomposition of $D + \sum_{i=1} \sigma_i u_i u_i^T$ in (19) approximately.

5. Performance results

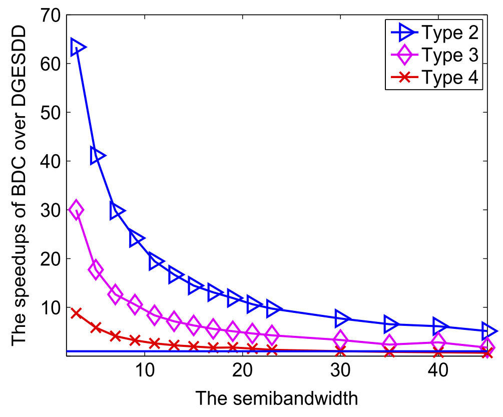
Our experiments[1] are performed on a server with 128 GB memory and two Intel(R) Xeon(R) CPU E5-2670, which has 16 cores in total. We used Intel compiler (ifort) with the optimization flags `-O2 -openmp -mavx` to compile the codes, and then linked the codes to Intel MKL with linking flag `-mkl=parallel`.

Example 1. In this example we use some matrices from LAPACK testing, whose eigenvalues or singular values are illustrated in Table 1. These matrices have also been used in [14,52]. In our experiments, the parameter k is arbitrarily set to 1.0e8, and the dimensions of matrices are $N = 10{,}000$. We use the LAPACK routine DLATMS to generate an upper banded or symmetric banded matrix.

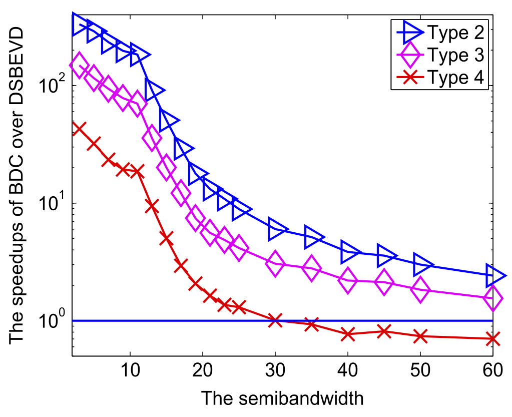
The matrices of type 1, 2, 5 and 6 have many deflations, nearly 100%. The percent of deflations for type 3 is about 50% and type 4 is about 20%. The deflation percentages of these matrices are all relatively high. As in [52], we use the matrices of type 2, 3 and 4 to present the results of BDC. The results for the SVD problem are illustrated in Fig. 1(a), which are obtained by letting OMP_NUM_THREADS and MKL_NUM_THREADS equal to 16.

The results of BDC in comparison with DSYEVD and DSBEVD are shown in Figs. 2(a) and 1(b), respectively. When the semibandwidth is small, it is interesting to see that BDC can be about one hundred times faster than DSYEVD and three hundreds times faster than DSBEVD. BDC becomes slower than DSYEVD when the semibandwidth is larger than 20, and

[1] Our implementation can be found in HSSPACK, which will be released on GitHub.

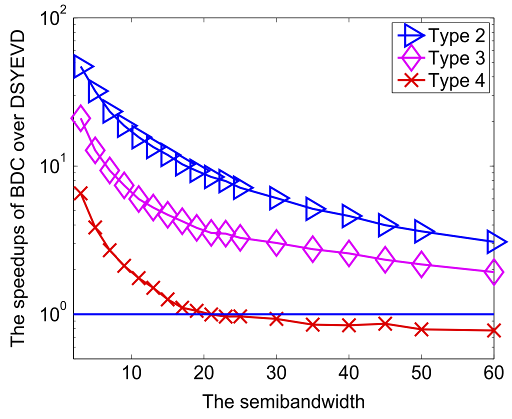


(a) Results for the SVD problem.

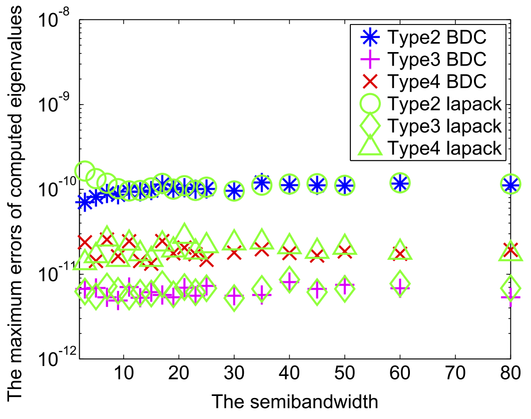


(b) Results for the eigenvalue problem.

Fig. 1. The speedups of BDC over DGESDD and DSBEVD.



(a) The speedups of BDC over DSYEVD.



(b) The accuracy of computed eigenvalues.

Fig. 2. Some more comparison results.

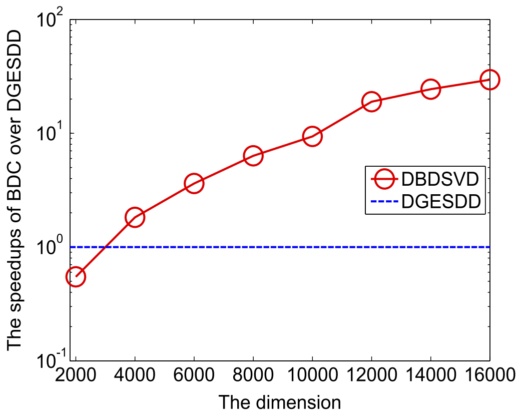
slower than DSBEVD when the semibandwidth is larger than 30 for matrix of type 4. For matrix of type 2 and 3, BDC can be faster than DSYEVD and DSBEVD even when the semibandwidth is larger than 60. Fig. 2(b) shows the maximum errors of the computed eigenvalues by BDC and the LAPACK routine DSBEVD, which are compared with the exact ones in Table 1. From the plots in Fig. 2(b), we conclude that the eigenvalues computed by BDC nearly have the same accuracy as those computed by DSBEVD.

Remark 1. In this paper we use RandHSS [34] to construct HSS matrix approximations during all the experiments. RandHSS uses random sampling and ID to compute the low-rank approximations, and ID is quite related to the strong rank-revealing QR factorization [37]. We stop ID when the norm of the pivoted row or column is less than 1e−14.

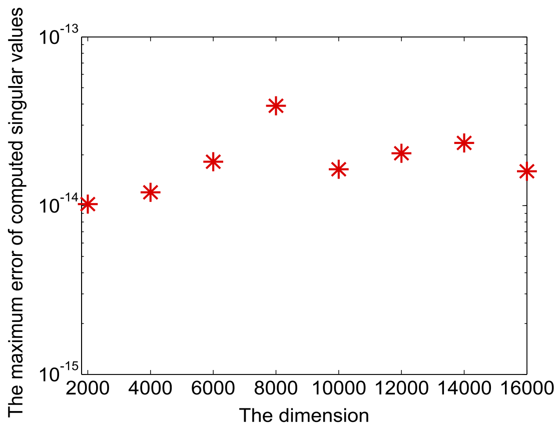
Example 2. In this example, we test the algorithms by using some random banded matrices with different leading dimensions, whose nonzero entries are Gaussian random numbers. The semibandwidth is 5.

The LAPACK routine DGBBRD could reduce a general banded matrix to its bidiagonal form. We find that it is not efficient, even slower than DGESDD. Therefore we only compare our accelerated DC algorithm (denoted by DBDSVD) with DGESDD. Fig. 3(a) shows the speedups of DBDSVD over DGESDD, and the maximum errors of singular values computed by DBDSVD are shown in Fig. 3(b), compared with those by DGESDD. When the dimension is large, DBDSVD can be more than 29x times faster than DGESDD in LAPACK.

The complexity of the banded DC algorithm increases as the semibandwidth increases. We use a random matrix with dimension 10,000 to show that DBDSVD is suitable for the banded matrices with narrow bandwidth. For the SVD case, the results are shown in Fig. 4(a). The banded DC algorithm without using the HSS matrix techniques is denoted by DBDSVD_CLASSIC. For comparison, we also denote DBDSVD by DBDSVD_HSS. From Fig. 4(a), we can see that the HSS matrix version is much faster than the one without using the HSS matrix techniques. When the semibandwidth is large, they both becomes slower than DGESDD. The results for the eigenvalue problem are shown in Fig. 4(b), where MDSBEVD_CLASSIC and MDSBEVD_HSS denote the banded DC for the eigenvalue problem without and with HSS matrix techniques, respectively.

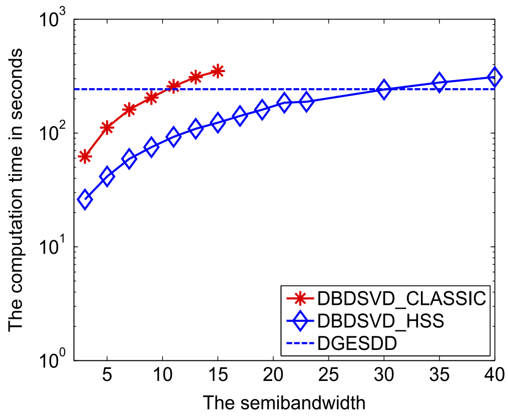


(a) The speedup.

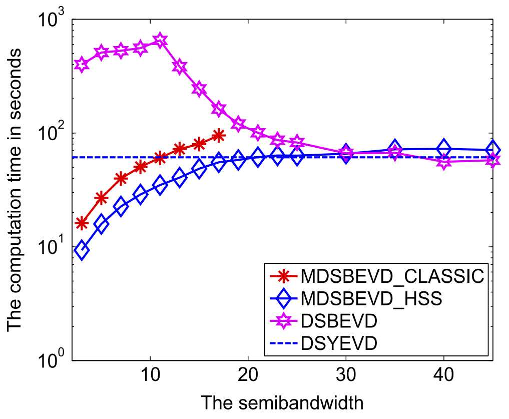


(b) The accuracy compared with DGESDD.

Fig. 3. The results for random matrices.



(a) Results for the SVD problem.



(b) Results for the eigenvalue problem.

Fig. 4. The wall times of different methods.

Table 2
The comparison of computing SVD for matrices from applications.

Matrix	Time (s)		Backward error	
	DBDSVD	DGESDD	DBDSVD	DGESDD
Mat1	250.3	1857.4	3.3e−18	8.9e−18
Mat2	19.4	421.4	5.6e−19	5.2e−19

Example 3. In this example, we use some matrices from real applications to test the proposed algorithms, which are obtained from the sparse matrix collection [24],

- 'LF10000': real symmetric, and the dimension is 19,998 and the semibandwidth is 3;
- 'linverse': real symmetric, and the dimension is 11,999 and the semibandwidth is 4.

This experiment also used 16 threads by letting OMP_NUM_THREADS and MKL_NUM_THREADS = 16. For simplicity, we denote matrix LF10000 by Mat1 and linverse by Mat2.

For the SVD problem, we first computed the QR factorization of Mat1 (Mat2) to obtain an upper banded matrix R with doubled semibandwidth, and then used matrix R to compare DBDSVD with DGESDD. Table 2 shows the execution times in seconds, and the backward errors of the computed SVD, which is defined as $\frac{\|R-U\Sigma V^T\|}{\|R\|\times N}$. From the results, we can see that DBDSVD is about $6\times$–$20\times$ times faster than the LAPACK routine DGESDD.

To compute the eigendecomposition of Mat1 and Mat2, we can use the LAPACK routine DSBEVD, DSYEVD or our banded DC algorithm for MDSBEVD. Their comparisons are presented in Table 3. For the eigenvalue problem, our accelerated BDC algorithm MDSBEVD is about $20\times$–$30\times$ times faster than the LAPACK routine DSYEVD and $100\times$–$200\times$ times faster than DSBEVD.

Table 3
The comparisons of computing the eigenvalue problem.

Matrix	Time (s)			Backward error		
	MDSBEVD	DSBEVD	DSYEVD	MDSBEVD	DSBEVD	DSYEVD
Mat1	25.6	2626.7	624.4	5.8e−18	4.9e−17	1.0e−18
Mat2	3.68	804.8	113.2	2.8e−19	3.4e−17	2.2e−19

Table 4
The computation time in seconds of MDSBEVD combined with SBR.

Matrix	MDSBEVD	DSBEVD	DSYEVD
Random	386.52	485.32	267.27
ted_B	19.86	51.23	61.28

Example 4. In this example we show that combining band reduction technique with the BDC algorithm is good for the banded matrices with relative large bandwidth. We use the symmetric eigenvalue problem to show this point. We first use SBR [7] to reduce a banded matrix to one with relatively small semibandwidth, and then use BDC to compute the eigendecomposition of this intermediate matrix, and finally the eigenvectors are computed via matrix–matrix multiplications.

We used two matrices to do the experiment: one is a 10,000 $\times$ 10,000 upper banded matrix with Gaussian random nonzero entries, and the other is from application, matrix ted_B in the sparse matrix collection [24]. Since the top-left part of ted_B is diagonal, we used the bottom-right part of ted_B with dimension 6361 and semibandwidth 48. The semibandwidth of the random matrix is 60. Since the routines in SBR [7] are essentially sequential, this example was performed only on one core. The run times in seconds of MDSBEVD are shown in Table 4. From it, we can see that MDSBEVD after combined with SBR can be faster than DSBEVD for the random matrix, though a little slower than DSYEVD. For the matrix ted_B, MDSBEVD is faster than both DSBEVD and DSYEVD.

6. Conclusion and future works

In this work, we use the HSS matrix techniques to accelerate the banded DC algorithms for the SVD and eigenvalue problems. Our algorithms are implemented using OpenMP. From the numerical results, we can see that our accelerated BDC can be much faster than the classical DC algorithms in LAPACK for some matrices with semibandwidths less than 20. One good research direction for future work is to combine HSS techniques with the runtime schedule systems to fully explore the multicore architecture of modern computers. The runtime systems include SMPSs [53] and QUARK (part of PLASMA [54]).

Acknowledgments

The authors would like to thank Jianlin Xia, Jie Liu and Qingfeng Hu for some helpful discussions. This work is partially supported by National Natural Science Foundation of China (Nos. 11401580, 91430218, 61133005 and 91530324), and 863 Program of China under grant 2012AA01A301.

References

[1] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, R. Harshman, Indexing by latent semantic analysis, J. Soc. Inf. Sci. 41 (1990) 391–407.
[2] R.M. Martin, Electronic Structure: Basic Theory and Practical Methods, Cambridge University Press, Cambridge, 2008.
[3] W. Su, J. Schrieffer, A.J. Heeger, Solition excitations in polyacetylene, Phys. Rev. B 22 (1980) 2099–2111.
[4] Y.-H. Bai, R.C. Ward, A parallel symmetric block-tridiagonal divide-and-conquer algorithm, ACM Trans. Math. Softw. 33 (4) (2007) 1–23.
[5] B.N. Parlett, The Symmetric Eigenvalue Problem, SIAM, Philadelphia, 1998.
[6] C.H. Bischof, B. Lang, X. Sun, A framework for symmetric band reduction, ACM Trans. Math. Software 26 (4) (2000) 581–601.
[7] C.H. Bischof, B. Lang, X.B. Sun, Algorithm 807: The SBR toolbox-software for successive band reduction, ACM Trans. Math. Software 26 (4) (2000) 602–616.
[8] M. Gu, S.C. Eisenstat, A divide-and-conquer algorithm for the bidiagonal SVD, SIAM J. Matrix Anal. Appl. 16 (1) (1995) 79–92.
[9] J. Demmel, Applied Numerical Linear Algebra, SIAM, Philadelphia, 1997.
[10] J.W. Demmel, W.M. Kahan, Accurate singular values of bidiagonal matrices, SIAM J. Sci. Comput. 11 (1990) 873–912.
[11] P.R. Willems, B. Lang, C. Vömel, Computing the bidiagonal SVD using multiple relatively robust representations, SIAM J. Matrix Anal. Appl. 28 (4) (2006) 907–926.
[12] P. Arbenz, Divide-and-conquer algorithms for the bandsymmetric eigenvalue problem, Parallel Comput. 18 (1992) 1105–1128.
[13] 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 (1) (2002) 45–58.
[14] A. Haidar, H. Ltaief, J. Dongarra, Toward a high performance tile divide and conquer algorithm for the dense symmetric eigenvalue problem, SIAM J. Sci. Comput. 34 (6) (2012) C249–C274.
[15] P. Arbenz, G.H. Golub, On the spectral decomposition of Hermitian matrices modified by low rank perturbations with applications, SIAM J. Matrix Anal. Appl. 9 (1988) 40–58.
[16] W.N. Gansterer, J. Schneid, C.W. Ueberhuber, A low-complexity divide-and-conquer method for computing eigenvalues and eigenvectors of symmetric band matrices, BIT 41 (2001) 967–976.

[17] W.N. Gansterer, R.C. Ward, R.P. Muller, W.A. Goddard III, Computing approximate eigenpairs of symmetric block tridiagonal matrices, SIAM J. Sci. Comput. 25 (2003) 65–85.
[18] J. Xia, S. Chandrasekaran, M. Gu, X.S. Li, Superfast multifrontal method for large structured linear systems of equation, SIAM J. Matrix Anal. Appl. 31 (2009) 1382–1411.
[19] L.G.W. Hackbusch, B. Khoromskij, Solution of large scale algebraic matrix riccati equations by use of hierarchical matrices, Computing 70 (2003) 121–165.
[20] R. Vandebril, M. Van Barel, N. Mastronardi, Matrix Computations and Semiseparable Matrices, Volume I: Linear Systems, Johns Hopkins University Press, 2008.
[21] M. Bebendorf, Hierarchical Matrices, in: Lecture Note in Computational Science and Engineering, vol. 63, Springer-Verlag, 2008.
[22] R. Vandebril, M. Van Barel, N. Mastronardi, Matrix Computations and Semiseparable Matrices, Volume II: Eigenvalue and Singular value Methods, Johns Hopkins University Press, 2008.
[23] S. Li, M. Gu, L. Cheng, X. Chi, M. Sun, An accelerated divide-and-conquer algorithm for the bidiagonal SVD problem, SIAM J. Matrix Anal. Appl. 35 (3) (2014) 1038–1057.
[24] T. Davis, Y. Hu, The Univeristy of Florida sparse matrix collection, ACM Trans. Math. Software 38 (1) (2011) 1:1–1:25.
[25] S. Chandrasekaran, M. Gu, T. Pals, A fast ULV decomposition solver for hierarchical semiseparable representations, SIAM J. Matrix Anal. Appl. 28 (2006) 603–622.
[26] J. Xia, S. Chandrasekaran, M. Gu, X. Li, Fast algorithm for hierarchically semiseparable matrices, Numer. Linear Algebra Appl. 17 (2010) 953–976.
[27] W. Hackbusch, A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices, Computing 62 (1999) 89–108.
[28] W. Hackbusch, B. Khoromskij, S. Sauter, On $\mathcal{H}^2$-matrices, in: Hans-Joachim Bungartz, Ronald H.W. Hoppe, Christoph Zenger (Eds.), Lecture on Applied Mathematics, Springer, Berlin, 2000, pp. 9–29.
[29] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.J. van der Veen, D. White, Some fast algorithms for sequentially semiseparable representation, SIAM J. Matrix Anal. Appl. 27 (2005) 341–364.
[30] Y. Eidelman, I. Gohberg, On a new class of structured matrices, Integral Equations Operator Theory 34 (1999) 293–324.
[31] R. Vandebril, M.V. Barel, G. Golub, N. Mastronardi, A bibliography on semiseparable matrices, Calcolo 42 (2005) 249–270.
[32] S. Li, X. Liao, J. Liu, H. Jiang, New fast divide-and-conquer algorithm for the symmetric tridiagonal eigenvalue problem, Numer. Linear Algebra Appl. (2016) http://dx.doi.org/10.1002/nla.2046.
[33] S. Li, M. Gu, C.J. Wu, J. Xia, New efficient and robust HSS Cholesky factorization of symmetric positive definite matrices, SIAM J. Matrix Anal. Appl. 33 (2012) 886–904.
[34] P.G. Martinsson, A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix, SIAM J. Matrix Anal. Appl. 32 (2011) 1251–1274.
[35] G.H. Golub, C.F.V. Loan, Matrix Computations, third ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[36] T.R. Chan, Rank revealing QR factorizations, Linear Algebra Appl. 88/89 (1987) 67–82.
[37] M. Gu, S.C. Eisenstat, Efficient algorithms for computing a strong-rank revealing QR factorization, SIAM J. Sci. Comput. 17 (1996) 848–869.
[38] H. Cheng, Z. Gimbutas, P. Martinsson, V. Rokhlin, On the compression of low rank matrices, SIAM J. Sci. Comput. 26 (4) (2005) 1389–1404.
[39] N. Halko, P.G. Martinsson, J.A. Tropp, Finding structure with randomness probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2011) 217–288.
[40] E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, M. Tygert, Randomized algorithms for the low-rank approximation of matrices, Proc. Natl. Acad. Sci. 104 (51) (2007) 20167–20172.
[41] S. Wang, X.S. Li, J. Xia, Y. Situ, M.V.D. Hoop, Efficient scalable algorithms for hierarchically semiseparable matrices, SIAM J. Sci. Comput. 35 (2013) C519–C544.
[42] M. Gu, A numerically stable superfast Toeplitz solver, LAPACK Seminar, 2009. URL: http://math.berkeley.edu/~mgu/Seminar/Fall2009/ToeplitzSeminarTalk.pdf.
[43] M. Gu, J. Xia, A multi-structured superfast Toeplitz solver, preprint, 2009.
[44] W. Lyons, Fast algorithms with applications to PDEs (Ph.D. thesis), University of California, Santa Barbara, 2005.
[45] J.R. Bunch, C.P. Nielsen, Updating the singular value decomposition, Numer. Math. 31 (1978) 111–129.
[46] M. Gu, S. Eisenstat, A stable and fast algorithm for updating the singular value decomposition, Tech. Rep. RR-966, Yale University, 1994.
[47] M. Moonen, P. Dooren, J. Vandewalle, A singular value decomposition updating algorithm for subspace tracking, SIAM J. Matrix Anal. Appl. 13 (4) (1992) 1015–1038.
[48] J.J.M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, Numer. Math. 36 (1981) 177–195.
[49] E.R. Jessup, D.C. Sorensen, A parallel algorithm for computing the singular value decomposition of a matrix, SIAM J. Matrix Anal. Appl. 15 (2) (1994) 530–548.
[50] J.R. Bunch, C.P. Nielsen, D.C. Sorensen, Rank one modification of the symmetric eigenproblem, Numer. Math. 31 (1978) 31–48.
[51] J. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University press, New York, 1965.
[52] G. Pichon, A. Haidar, M. Faverge, J. Kurzak, Divide and conquer symmetric tridiagonal eigensolver for multicore architectures, in: The 29th IEEE International Parallel & Distributed Processing Symposium, IEEE, 2015, pp. 142–151.
[53] J. Perez, R. Badia, J. Labarta, A dependency-aware task-based programming environment for multi-core architectures, in: Cluster Computing, 2008 IEEE International Conference, IEEE, 2008, pp. 142–151.
[54] E. Agullo, A. Buttari, J. Dongarra, M. Faverge, B. Hadri, A. Haidar, J. Kurzak, J. Langou, H. Ltaif, P. Luszczek, A. YarKhan, PLASMA users' guide, Tech. Rep., University of Tennessee, Knoxville, TN, 2010.