Journal of Computational and Applied Mathematics 344 (2018) 512–520

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

An efficient hybrid tridiagonal divide-and-conquer algorithm on distributed memory architectures

Shengguo Li [a,*], François-Henry Rouet [b,c], Jie Liu [a,d], Chun Huang [a,d], Xingyu Gao [e], Xuebin Chi [f]

[a] College of Computer, National University of Defense Technology (NUDT), Changsha 410073, China
[b] Livermore Software Technology Corporation, Livermore, CA 94550, USA
[c] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[d] State Key Laboratory of High Performance Computing, NUDT, China
[e] Institute of Applied Physics and Computational Mathematics, Beijing 100094, China
[f] Computer Network Information Center, Chinese Academy of Science, Beijing 100190, China

ARTICLE INFO

Article history:
Received 9 December 2016
Received in revised form 29 November 2017

MSC:
65F15, 68W10

Keywords:
ScaLAPACK
Divide-and-conquer
HSS matrix
Distributed parallel algorithm

ABSTRACT

In this paper, we propose an efficient divide-and-conquer (DC) algorithm for symmetric tridiagonal matrices based on ScaLAPACK and the hierarchically semiseparable (HSS) matrices. HSS is an important type of rank-structured matrices. The most computationally intensive part of the DC algorithm is computing the eigenvectors via matrix–matrix multiplications (MMM). In our parallel hybrid DC (PHDC) algorithm, MMM is accelerated by using HSS matrix techniques when the intermediate matrix is large. All the HSS computations are performed via the package STRUMPACK. PHDC has been tested by using many different matrices. Compared with the DC implementation in MKL, PHDC can be faster for some matrices with few deflations when using hundreds of processes. However, the gains decrease as the number of processes increases. The comparisons of PHDC with ELPA (the Eigenvalue soLvers for Petascale Applications library) are similar. PHDC is usually slower than MKL and ELPA when using 300 or more processes on the Tianhe-2 supercomputer.

1. Introduction

Symmetric tridiagonal eigenvalue problems are usually solved by the divide and conquer (DC) algorithm both on shared memory multicore platforms and parallel distributed memory machines. The DC algorithm is fast and stable, and well-studied in numerous Refs. [1–6]. It is now the default method in LAPACK [7] and ScaLAPACK [8] when the eigenvectors of a symmetric tridiagonal matrix are required.

Recently, the authors [9] used hierarchically semiseparable (HSS) matrices [10] to accelerate the tridiagonal DC in LAPACK, and obtained about 6x speedups in comparison with the LAPACK version for some large matrices on a shared memory multicore platform. The bidiagonal and banded DC algorithms for the SVD problem are accelerated similarly [11,12]. The main point is that some intermediate eigenvector matrices are rank-structured matrices [10,13]. HSS representations are used to approximate them, and fast HSS algorithms are used to update the eigenvectors. HSS is an important type of rank-structured matrices; others include $\mathcal{H}$-matrix [13], $\mathcal{H}^2$-matrix [14], quasiseparable [15] and sequentially semiseparable (SSS) [16,17]. In this paper, we extend the techniques used in [11,9] to the distributed memory environment, in order to

* Corresponding author.
E-mail address: nudtlsg@nudt.edu.cn (S. Li).

https://doi.org/10.1016/j.cam.2018.05.051
0377-0427/

accelerate the tridiagonal DC algorithm in ScaLAPACK [8]. To integrate HSS algorithms into ScaLAPACK routines, an efficient distributed HSS construction routine and an HSS matrix multiplication routine are required. In our experiments, we use the routines in STRUMPACK (STRUctured Matrices PACKage) package [18], which is designed for computations with both *sparse* and *dense* structured matrices. STRUMPACK has two main components: a *dense matrix computation package* and a *sparse direct solver and preconditioner*. In this work we only use its dense matrix operation part.[1] It is written in C++ and uses MPI for distributed-memory parallelism, and it implements a parallel HSS construction algorithm with randomized sampling [19,20]. Note that some implementations are available for sequential HSS algorithms [21,22] or parallel HSS algorithms on shared memory platforms such as HSSPACK [12].[2] However, STRUMPACK is the only one that provides distributed parallel HSS algorithms. More details about it and HSS matrices will be introduced in Section 2.

The ScaLAPACK routine implements the rank-one update version of Cuppen's DC algorithm [1]. We briefly introduce the main processes. Assume that T is a symmetric tridiagonal matrix,

$$T = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & a_{N-1} & b_{N-1} \\ & & b_{N-1} & a_N \end{pmatrix}. \tag{1}$$

Cuppen introduced the decomposition

$$T = \begin{pmatrix} T_1 & \\ & T_2 \end{pmatrix} + b_k v v^T, \tag{2}$$

where $T_1 \in R^{k\times k}$ and $v = [0, \ldots, 0, 1, 1, 0, \ldots, 0]^T$ with ones at the kth and $(k+1)$th entries. Let $T_1 = Q_1 D_1 Q_1^T$ be $T_2 = Q_2 D_2 Q_2^T$ be eigen decompositions, and then (1) can be written as

$$T = Q\left(D + b_k z z^T\right) Q^T, \tag{3}$$

where $Q = \mathrm{diag}(Q_1, Q_2), D = \mathrm{diag}(D_1, D_2)$ and $z = Q^T v = \begin{pmatrix} \text{last col. of } Q_1^T \\ \text{first col. of } Q_2^T \end{pmatrix}$.The problem is reduced to computing the spectral decomposition of the diagonal plus rank-one

$$D + b_k u u^T = \widehat{Q}\,\Lambda\,\widehat{Q}^T. \tag{4}$$

By Theorem 2.1 in [1], the eigenvalues λ_i of matrix $D + b_k z z^T$ are the root of the secular equation

$$f(\lambda) = 1 + b_k \frac{z_k^2}{d_k - \lambda} = 0,$$

where z_k and d_k are the kth component of z and the kth diagonal entry of D, respectively, and its corresponding eigenvector is given by $\hat{q}_i = (D - \lambda_i)^{-1} z$. Eigenvectors computed this way may loss orthogonality. To ensure orthogonality, Sorensen and Tang [23] proposed to use extended precision. This extra precision approach was used by Gates and Arbenz [5] in their implementation. However, the implementation in ScaLAPACK uses the Löwner theorem approach, instead of the extended precision approach [23].

Remark 1. The extra precision approach is "embarrassingly" parallel with each eigenvalue and eigenvector computed without communication, but it is not portable to every platform. The Löwner approach requires information about all the eigenvalues, requiring a broadcast. However, the length of communication message is $O(n)$ which is trivial compared with the $O(n^2)$ communication of eigenvectors.

The excellent performance of the DC algorithm is partially due to *deflation* [2,1], 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 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. All the deflated eigenvalues would be permuted back to D by a permutation matrix, so do the corresponding eigenvectors. Then (3) reduces to, after deflation,

$$T = Q(GP)\begin{pmatrix} \bar{D} + b_k \bar{z}\bar{z}^T & \\ & \bar{D}_d \end{pmatrix}(GP)^T Q^T, \tag{5}$$

where G is the product of all rotations, and P is a permutation matrix and $\bar{D}_d$ are the deflated eigenvalues.

According to (4), the eigenvectors of T are computed as

$$U = Q(GP)\begin{pmatrix} \widehat{Q} & \\ & I_d \end{pmatrix} = \left[\begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} GP\right]\begin{pmatrix} \widehat{Q} & \\ & I_d \end{pmatrix}. \tag{6}$$

[1] The current version is STRUMPACK-Dense-1.1.1, which is available at http://portal.nersc.gov/project/sparse/strumpack/.

[2] Some Fortran and Matlab codes are available at Jianlin Xia's homepage, http://www.math.purdue.edu/~xiaj/, and HSSPACK is available at GitHub.