TOWARD HIGH PERFORMANCE DIVIDE AND CONQUER EIGENSOLVER FOR DENSE SYMMETRIC MATRICES *

AZZAM HAIDAR †, HATEM LTAIEF ‡, AND JACK DONGARRA §

Abstract. This paper presents a high performance eigensolver for dense symmetric matrices on multicore architectures. Based on the well-known divide and conquer (D&C) methodology introduced by Cuppen, this algorithm computes all the eigenvalues of the symmetric matrix. The general D&C can be expressed in three stages: (1) Partitioning into subproblems, (2) Computing the solution of the subproblems and (3) Merging the subproblems. It is therefore well-suited for data parallel algorithmic technics due to the number of independent computational tasks which can potentially run concurrently. In particular, **tile algorithms** have recently shown very promising performance results for solving linear systems of equations. The idea consists of splitting the input matrix into small square tiles and reorganizing the data within each tile to be contiguous in memory for efficient cache reuse. The authors propose to extend this idea to the D&C eigensolver algorithm. The *tile* DC (TD&C) eigensolver algorithm described in this paper takes a dense symmetric matrix in tile layout as input, reduces it to symmetric band form by applying orthogonal transformations and finally, applies the D&C approach on the symmetric band matrix to calculate all eigenvalues. The whole execution flow can then be represented as a directed acyclic graph where nodes are tasks and edges represent dependencies between them. A light weighted runtime system environment is used to dynamically schedule the different tasks in order to ensure the data dependencies are not violated. The tasks are scheduled in an out-of-order fashion with a special emphasis on the data locality and the pursuit of the critical path. The performance results obtained for large matrix sizes and certain matrix types are staggering. The proposed TD&C symmetric eigensolver reaches up to 14X speed up compared to the state-of-the-art numerical open source library LAPACK and up to 4X speed up against the commercial numerical library Intel MKL V10.2. Performance results are also reported comparing the TD&C symmetric eigensolver with other standard methods such as the bisection, the QR iteration and MRRR. Last but not least, a study on the accuracy of the computed eigenvalues is provided which gives a certain confidence on the quality of the overall TD&C symmetric eigensolver presented in this paper.

Key words. divide and conquer, symmetric eigensolver, tile algorithms, dynamic scheduling

AMS subject classifications. 15A18, 65F15, 65F18, 65Y05, 65Y20, 68W10

1. Introduction.

The objective of this paper is to introduce a new high performance *Tile Divide and Conquer* (TD&C) eigensolver for dense symmetric matrices on homogeneous multicore architectures. The necessity of calculating eigenvalues emerges from various computational science disciplines e.g., in quantum physics [32], chemistry [34] and mechanics [23] as well as in statistics when computing the principal component analysis of the symmetric covariance matrix. As multicore systems continue to gain ground in the high performance computing community, linear algebra algorithms have to be redesigned or new algorithms have to be developed in order to take advantage of the architectural features brought by these processing units.

The TD&C eigensolver algorithm for symmetric matrices described in this paper takes root from the well-known D&C methodology introduced by Cuppen [10]. Many serial and parallel Cuppen-based eigensolver implementations for shared and distributed memory have been proposed in the past [16, 20, 24, 25, 35, 36, 38]. However, the symmetric matrix A has first to be transformed into a tridiagonal matrix T by applying successive orthogonal transformations. The D&C approach can then be expressed in three stages: (1) Partitioning T into subproblems, (2) Computing the solution of the subproblems which are defined by a

*Research reported here was partially supported by the National Science Foundation, Department of Energy and Microsoft Research.

†Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville. (haidar@eecs.utk.edu).

‡KAUST Supercomputing Laboratory, Thuwal, Saudi Arabia. (Hatem.Ltaief@kaust.edu.sa).

§Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville. Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee. School of Mathematics & School of Computer Science, University of Manchester. (dongarra@eecs.utk.edu).

rank-one modification of a diagonal matrix and (3) Merging the subproblems in a bottom-up fashion. This approach is therefore well-suited for data parallel algorithmic techniques due to the number of independent computational tasks which can potentially run concurrently. In particular, **tile algorithms** have recently shown very promising performance results for solving linear systems of equations using Cholesky, QR/LQ and LU factorizations available in PLASMA [39] library and other similar projects like FLAME [40]. The PLASMA concept consists of splitting the input matrix into square tiles and reorganizing the data within each tile to be contiguous in memory (block data layout) for efficient cache reuse. The authors propose to extend this idea to the symmetric TD&C eigensolver case. The standard algorithms may then have to be redesigned to match the tile layout underneath.

The TD&C symmetric eigensolver framework takes a dense symmetric matrix as input, translates it into block data layout and reduces it to symmetric band form by applying successive orthogonal transformations. This strategy of partially reducing the matrix allows us the casting of most level 2 BLAS operations (memory-bound) during the panel factorization required in the standard tridiagonalisation step to level 3 BLAS operations (compute-bound) for high performance execution purposes. Indeed, the most time consuming part of any standard symmetric eigensolvers is the full reduction to tridiagonal form. This reduction phase can take up to 90% of the elapsed time if only eigenvalues are needed and 50% if both, eigenvalues and eigenvectors are calculated. There are mainly two ways to compute the eigenvalues of the resulting symmetric band matrix: a) further reduce the matrix to tridiagonal form using a bulge chasing procedure as developed by Luszczek et al. [30], and then apply the standard D&C algorithm (or any other methods) to calculate the eigenvalues from the tridiagonal form, or b) directly compute the eigenvalues from the symmetric band matrix itself. The latter approach is used in our TD&C eigensolver framework to calculate all eigenvalues by extending the three stages of the standard D&C eigensolver, originally designed for the symmetric *tridiagonal* matrix, to the symmetric *band* matrix. The whole execution flow can then be represented as a directed acyclic graph where nodes are tasks and edges represent dependencies between them. An efficient and light weighted runtime system environment named QUARK [28] (internally used by the PLASMA library) is exploited to dynamically schedule the different tasks and to ensure that the data dependencies are not violated. The execution flow is now driven by the data dependencies. As soon as the dependencies are satisfied, QUARK initiates and executes the corresponding tasks on the available computational resources. This engenders an out-of-order execution of tasks which removes unnecessary synchronization points and allows the different TD&C stages to overlap. In fact, by hinting QUARK, we are able to resolve two interrelated issues which come into play with regard to scheduling optimality: data locality and the pursuit of the critical path.

The TD&C symmetric eigensolver algorithm has been extensively evaluated across many matrix types and against similar D&C symmetric eigensolver implementations from state-of-the-art numerical libraries. The performance results obtained asymptotically and for some matrix types are very impressive. Performance results are also reported comparing the TD&C symmetric eigensolver with other standard methods such as the bisection, the QR iteration and MRRR. Last but not least, a study on the accuracy of the computed eigenvalues is provided which gives a certain confidence on the quality of the overall TD&C symmetric eigensolver framework presented in this paper.

The remainder of this paper is organized as follows: Section 2 gives a detailed overview of previous projects in this area. Section 3 recalls the standard D&C symmetric eigensolver algorithm. Section 4 describes the TD&C symmetric eigensolver framework and provides its algorithmic complexity. Section 5 gives some details about the parallel implementations of the algorithm and highlights our contributions. Section 6 presents performance results of

the overall algorithm on shared-memory multicore architectures against the corresponding standard D&C routines from LAPACK [2] and the Intel vendor library MKL V10.2 [1] on various matrix types. Also, performance comparisons against other numerical methods are illustrated along with a study on the accuracy of the computed eigenvalues. Finally, Section 7 summarizes the results of this paper and discusses the ongoing work.

2. Related Work. This section describes previous works on D&C eigensolvers for symmetric band matrices. Band matrices naturally arise in many areas such as the electronic simulations in quantum mechanics, vibrating systems approximated by finite elements or splines and also in the block Lanczos algorithm where a sequence of increasingly larger band symmetric eigenvalue problems are generated. Besides the overhead of the tridiagonalisation step, this motivates attempts to compute the eigen decomposition directly from the band form using variants of the standard D&C algorithm.

Arbenz et al. [3, 4, 5] investigated a generalized D&C approach for computing the eigenpairs of symmetric band matrices. The authors provide many important theoretical results concerning the eigenanalysis of a low rank modification of a diagonal matrix. Arbenz [3], proposed two methods for computing eigenpairs of a rank-b modification of a diagonal matrix. The first approach consists of computing the rank b modification as a sequence of rank-one modifications. The second approach lies in compressing the rank b modification to a small $b \times b$ eigenproblem, solves it and then reconstructs the solution of the original problem [4, 5]. The first approach requires more floating point operations than the second one, which has serial complexity in the $\mathcal{O}(n^3)$ term. The author opted for the second approach. Unfortunately, numerical instabilities in the computation of the eigenvectors have been observed [3], and currently no numerically stable implementation of the second approach exists.

Also, Gansterer et al. [18] developed a D&C algorithm for band symmetric matrices which computes the eigen decomposition. Their approach is based on the separation of the eigenvalue and the eigenvector computations. The eigenvalues are computed recursively by a sequence of rank-one modifications of the standard D&C technique. Once the eigenvalues are known, the corresponding eigenvectors are computed using modified QR factorizations with restricted column pivoting.

Later, Gansterer et al. [19] and Ward et al. [41] proposed another alternative for the generalized D&C algorithm, which rather computes *approximate* eigenvalues and eigenvectors of symmetric block tridiagonal matrices. Bai and Ward [6] proposed a parallel and distributed version of it. Their algorithm calculates all eigenvalues and eigenvectors of a block-tridiagonal matrix to reduced accuracy by approximating the rank deficient off-diagonal blocks of lower magnitudes with rank-one matrices.

The tile D&C symmetric eigensolver algorithm (TD&C) presented in this paper differs from Arbenz's algorithm [4, 5] in the fact that it uses the first approach (sequence of b rank-one updates) for the solution of the low-rank modifications. The first approach, based on a sequence of b rank-one updates, has also been used by Gansterer et al [18]. However, our TD&C algorithm differs from [18] in the representation of the subdiagonal blocks, which are not rank deficient. Although, our proposed algorithm has been developed in close analogy to Ward et al. [41] algorithm, but without the use of an approximation of the subdiagonal blocks. The details are discussed in Section 3.

3. The *Standard* D&C Symmetric Eigensolver Algorithm. This section recalls the methodology underlying the standard Cuppen's D&C algorithm for symmetric eigensolver.

The symmetric dense matrix A first has to be reduced into a tridiagonal matrix, T, using successive orthogonal transformations. The D&C approach can then be expressed in three phases:

- Partition the tridiagonal matrix T into several subproblems in a recursive manner.

- Compute the eigen decompositions of each subproblem, which are defined by a rank-one modification of a diagonal matrix.
- Merge the subproblems and proceed to the next level with a bottom-up fashion.

This method amounts to constructing a binary tree with each node representing a rank-one tear and hence, a partition into two subproblems. Figure 3.1 shows a tree of depth three representing a splitting of the original problem into eight smaller subproblems. There are two *simple* eigenvalue problems to be solved at each leaf of the tree. Each of these problems may be considered as an independent problem without any data dependencies with the other leaves of the tree. The tree is then traversed in reverse order where at each node the results are merged from the left son and the right son calculations.

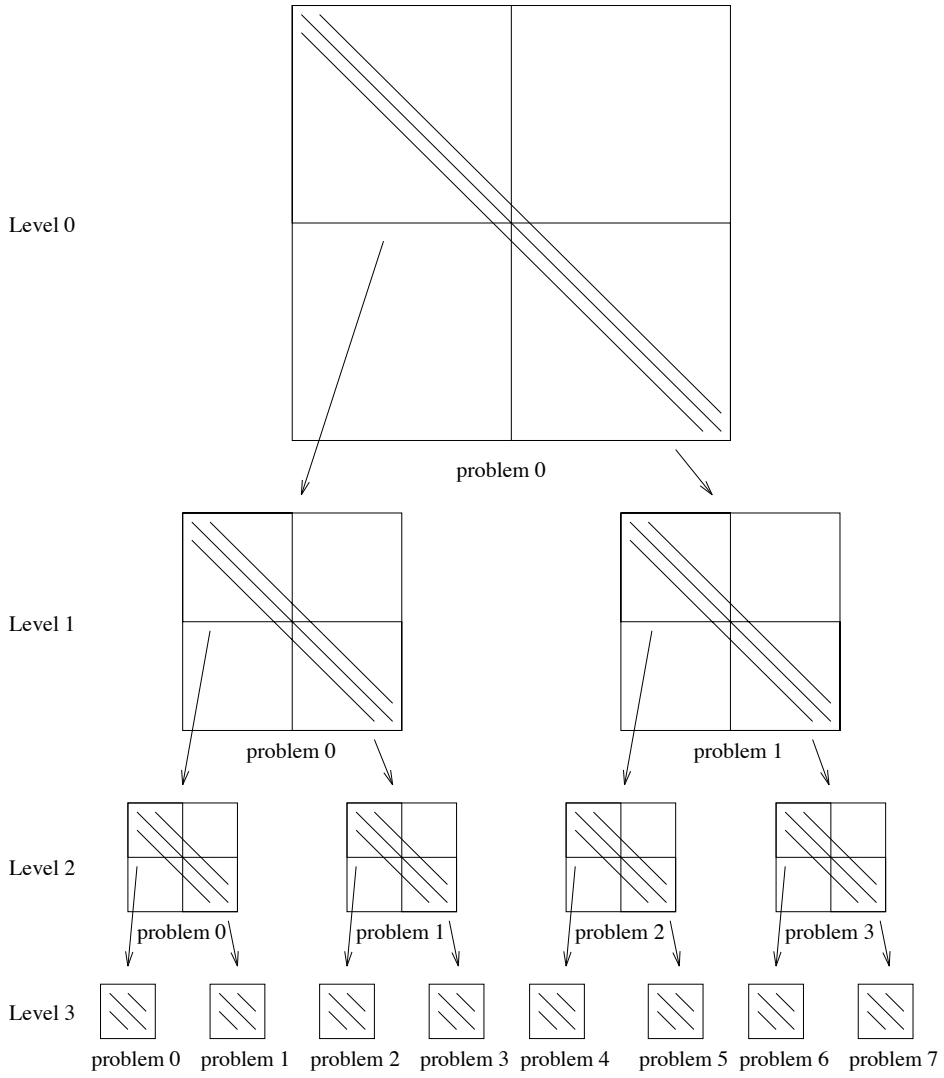

FIG. 3.1. *A Divide and Conquer tree*

The D&C approach is sequentially one of the fastest methods currently available if all

eigenvalues and eigenvectors are to be computed [11]. It also has attractive parallelization properties as shown in [38]. Finally, it is noteworthy to mention the *deflation* process, which occurs during the computation of the low rank modifications. It consists of preventing the computation of an eigenpair of a submatrix (matrix of a son in the tree) that is acceptable to be an eigenpair of the larger submatrix (matrix of a father in the tree). Therefore, the greater the amount of deflations, the lesser the number of required operations, which leads to a better performance. The amount of deflations depends on the eigenvalue distribution as well as the structure of the eigenvectors. In practice, most of the application matrices arising from engineering areas provide a reasonable amount of deflations, and so the D&C algorithm runs at less than $\mathcal{O}(n^{2.5})$ instead of $\mathcal{O}(n^3)$.

The next section explains how the standard D&C symmetric eigensolver has been extended to work on a symmetric band matrix using tile algorithms (TD&C).

4. The *Tile* D&C Symmetric Eigensolver Algorithm. This section highlights the skeleton of the divide and conquer eigensolver for dense symmetric matrices using tile algorithms (TD&C). The dense matrix is first reduced to band form with a given bandwidth b. The three stages of the standard D&C symmetric eigensolver are then developed to be applied directly on the band matrix to calculate the eigenvalues. The general algorithm can then be represented as a *Directed Acyclic Graph* (DAG) where nodes represent tasks, either the panel factorization or update of a tile, and edges represent dependencies among them. The last subsection details the algorithmic complexity of the overall framework.

4.1. Matrix Reduction to Symmetric Band Form. Designated as a solution to overcome the overhead of fork-join approaches, tile algorithms consist of breaking the panel factorization, and trailing submatrix update steps into smaller tasks that operate on a square tile (i.e., a set of b contiguous columns where b is the tile size) in order to fit the small core caches of the underlying hardware.

The matrix reduction to symmetric band form follows this algorithmic strategy and relies on highly optimized compute-intensive kernels to achieve high performance. It is composed of eight kernels total. Four kernels come directly from the one-sided QR factorization [9] and the four others have been recently implemented to handle the symmetry property of the matrix when updating the trailing submatrix around the diagonal. The complete algorithm of the symmetric band reduction can be found in Section 5 from Luszczek et al. [30]. The reduced band matrix has a semibandwidthof size b. Now, there are two possibilities to get the eigenvalue from the band matrix. The matrix can be further reduced to achieve the tridiagonal form using a bulge chasing procedure as in [30]. The tile size, b, becomes a critical parameter to be tuned as the bulge chasing technique requires a matrix with a small bandwidth in order to efficiently run. Another way is to compute the eigen decomposition from the band form and again, a particular attention on the tile size, b, is necessary as explained later in Section 4.5.

The next subsections show how the same concepts of the standard D&C method described by Cuppen [10] hold for calculating the eigenvalues of a symmetric band matrix.

4.2. Partitioning into Subproblems. Similarly to Cuppen's D&C for tridiagonal matrices, the band matrix A can be divided into p parts. Let:

$$A = \begin{pmatrix} B_1 & C_1^T & & & \\ C_1 & B_2 & C_2^T & & \\ & C_2 & \ddots & \ddots & \\ & & \ddots & B_{p-1} & C_{p-1}^T \\ & & & C_{p-1} & B_p \end{pmatrix} \in \mathbb{R}^{n \times n}, \tag{4.1}$$

For simplicity purposes, we describe the technique for $p = 2$ but it is easy to generalize for any $p < n$ with n being the size of the matrix. If we split A into two parts, A can be written:

$$A = \begin{pmatrix} B_1 & C_1^T \\ C_1 & B_2 \end{pmatrix}, \tag{4.2}$$

where $B_i \in \mathbb{R}^{p_i \times p_i}$, $\quad p_1 + p_2 = n$, and C_1 is the upper triangular tile of A which is to be "cut out". Thus, A can be written in such form:

$$A = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix} + R, \quad \textit{where} \quad B_i \in \mathbb{R}^{p_i \times p_i}, \quad p_1 + p_2 = n \tag{4.3}$$

There are several ways to define the matrix R. Arbenz [3] shows that it is possible to have the rank of $R = b$ if one defines:

$$R = \sum_{j=1}^{b} \delta_j v_j v_j^T, \tag{4.4}$$

where

$$\delta_j \neq 0, \quad \textit{and} \quad v_j := \theta_j e_{p_1-b+j} + \frac{1}{\delta_j \theta_j} \sum_{m=1}^{j} a_{p_1+m,p_1-b+j} e_{p_1+m}.$$

This is equivalent in a matrix notation to:

$$R = V \Delta V^T, \qquad V = \begin{pmatrix} O \\ \Theta \\ C_1 \Theta^{-T} \Delta^{-1} \\ 0 \end{pmatrix}, \tag{4.5}$$

where Θ, Δ are diagonal matrices and where

$$C_1 = \begin{pmatrix} a_{p_1+1,p_1-b+1} & \cdots & a_{p_1+1,p_1} \\ & \ddots & \vdots \\ & & a_{p_1+b,p_1} \end{pmatrix} \in \mathbb{R}^{b \times b}.$$

More generally, R can be represented by:

$$R = \begin{pmatrix} O_{p_1-b} & O & O & O \\ O & M & C_1^T & O \\ O & C_1 & C_1 M^{-1} C_1^T & O \\ O & O & O & O_{p_2-b} \end{pmatrix} \tag{4.6}$$

where $M = \Theta \Delta \Theta^T$, $\Delta = \Delta^T$ and Δ and Θ are any regular matrices.

Gansterer and al. [18] have chosen $M = I$. For this choice, R is in most cases unbalanced, meaning that $\| C_1^T M^{-1} C_1 \|$ can be much smaller or larger than $\| M \| = I$. However, if $C_1 = X \Sigma Y^T$ is the singular value decomposition (SVD) of C_1, then $M = -Y \Sigma Y^T$ and $C_1^T M^{-1} C_1 = -X \Sigma X^T$. So both these matrices are symmetric negative definite and have the same norm and condition number as C_1. The modification R obtained this way is balanced and has the form (4.5) with $\Theta = Y$, and $\Delta = -\Sigma$. Muller and Ward [6, 19] choose to "*approximate*" the off-diagonal tile C_1 by lower rank matrices using their singular value decomposition.

Our TD&C eigensolver implementation is designed along those lines. The choice of the matrix R is based on the computation of the singular value decomposition of the off-diagonal tile C_1. We note that computing the SVD of C_1 costs $\mathcal{O}(b^3)$ floating point operations. Assuming that $b \ll n$, the computation cost for the SVD is negligible compared to the overall cost of the entire algorithm. Furthermore, the SVD yields the actual rank of C_1 and makes it possible to take advantage of eventual rank deficiencies.

Going back to the standard Cuppen's algorithm, we plug in the new notations and A can then be rewritten such as:

$$\begin{aligned} A &= \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix} + R, \\ &= \begin{pmatrix} \tilde{B}_1 & 0 \\ 0 & \tilde{B}_2 \end{pmatrix} + \Sigma Z Z^T \end{aligned} \tag{4.7}$$

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

In a general form ($p > 2$):

$$B = diag(\tilde{B}_i) + \sum_{i=1}^{p-1} \Sigma_i Z_i Z_i^T \tag{4.8}$$

where,
$\tilde{B}_1 = B_1 - Y_1\Sigma_1 Y_1^T$,
$\tilde{B}_i = B_i - Y_{i-1}\Sigma_{i-1}Y_{i-1}^T - X_i\Sigma_i X_i^T, \quad \text{for } 2 \le i \le p-1$,
$\tilde{B}_p = B_p - X_{p-1}\Sigma_{p-1}X_{p-1}^T$,
and
$Z_1 = \begin{pmatrix} Y_1 \\ X_1 \\ 0 \\ 0 \end{pmatrix}$, $Z_i = \begin{pmatrix} 0 \\ Y_i \\ X_i \\ 0 \end{pmatrix}$, $\quad for \quad 2 \le i \le p-2$, and $Z_{p-1} = \begin{pmatrix} 0 \\ 0 \\ Y_{p-1} \\ X_{p-1} \end{pmatrix}$.
It is noteworthy to mention that the SVD of the off-diagonal upper triangular tiles are independent from each other.

The next step is the computation of the eigen decompositions for each updated block $\tilde{B}_i$.

4.3. Solution of the subproblems. The second phase corresponds to the spectral decomposition of each symmetric diagonal tile ($\tilde{B}_i$). This is done through calling the appropriate LAPACK routine for symmetric matrices. As a result, we can write:

$$\tilde{B}_i = Q_i D_i Q_i^T. \tag{4.9}$$

The diagonal matrices D_i contains the eigenvalues of $\tilde{B}_i$ and the matrices Q_i are the corresponding eigenvectors.
Thus, the original matrix A can be rewritten as follows:

$$\begin{aligned} A &= \begin{pmatrix} \tilde{B}_1 & 0 \\ 0 & \tilde{B}_2 \end{pmatrix} + \Sigma Z Z^T \\ &= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + \Sigma U U^T \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T \end{aligned} \tag{4.10}$$

where $U = Q^T Z$.

The eigenvalues of A are therefore equal to those of $\left\{ \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + \Sigma U U^T \right\}$. This is equivalent to $(D + UU^T)y = \lambda y$ where $U \in \Re^{n \times b}$ is a matrix of maximal rank $b \leq n$.

Having computed the local spectral decompositions of each tree leaf, the global spectral decomposition can then be calculated by merging the solution subproblem as described in the next subsection.

4.4. Amalgamation of the subproblems. The amalgamation phase consists of traversing the tree in a bottom-up fashion where at each node the results are merged from the left son and the right son calculations. The merging step handles the computation of the spectral decomposition of $(D + UU^T)y = \lambda y$. Here again, it is obvious that all the merging operations, which perform a *rank-b* updating process between two adjacent eigenvalue subproblems, can concurrently run within the corresponding levels.

There are different methods for computing the eigen decomposition of a low rank modification of an Hermitian matrix $(D + UU^T)y = \lambda y$. Two main strategies can be distinguished:

- "$b \times 1$" approach that computes a sequence of b rank-one modifications of a diagonal matrix.
- "$1 \times b$" approach that computes a rank-b modification of a diagonal matrix,

Arbenz [3] followed the "$1 \times b$" approach to compute the eigenpairs of a rank-b modification by transforming the problem into a $b \times b$ smaller eigenproblem. Unfortunately, numerical instabilities in the computation of the eigenvectors have been observed [3]. The orthogonality of the eigenvectors may be lost if close eigenvalue are present. The resulting algorithm lose 2 to 4 decimal digits compared to LAPACK.

We propose to take advantage of the stability of the existing methodology of the standard D&C algorithm to rather follow the first approach and compute a sequence of b rank-one modifications for calculating the eigenvalues of $(D + UU^T)y = \lambda y$.

The rank-one modification:
Each rank-one modification consists of computing the eigensystem of a matrix of the form

$$\tilde{Q}\tilde{D}\tilde{Q}^T = (D + \sigma uu^T). \tag{4.11}$$

where D is a real $n \times n$ diagonal matrix, σ is a nonzero scalar, and u is a real vector of order n and has Euclidean norm equal to 1. We seek a formula or an eigenpair of the system for the matrix on the right hand side of (4.11). The eigenpair can be computed based on the following theorem 4.1.

THEOREM 4.1. *Let $B = D + \sigma uu^T$, where D=diag$\{d_1, d_2, ..., d_n\}$, $n \geq 2$, and $\| u \|_2 = 1$. Let $d_1 \leqq d_2 \leqq d_3 \leqq ... \leqq d_n$ be the eigenvalues of D, and let $\lambda_1 \leqq \lambda_2 \leqq \lambda_3 \leqq ... \leqq \lambda_n$ be the eigenvalues of B.*
Then as Wilkinson [42] showed, we can $\lambda_i = d_i + \sigma\mu_i$, where $1 \leqq i \leqq n$, $\sum_{i=1}^{n} \mu_i = 1$ and $0 \leqq \mu_i \leqq 1$. Moreover, an important property is that:

- *$d_1 \leqq \lambda_1 \leqq d_2 \leqq \lambda_2 \leqq ... \leqq d_n \leqq \lambda_n$, if $\sigma > 0$,*
- *$\lambda_1 \leqq d_1 \leqq \lambda_2 \leqq d_2 \leqq ... \leqq \lambda_n \leqq d_n$, if $\sigma < 0$,*

Finally, if the d_i are distinct and all the elements of u are nonzero, then the eigenvalues of B strictly separate those of D. Golub [22] has shown that for the above situation (d_i are distinct and all the elements of $u \neq 0$), the eigenvalues of B are equal to the root of the rational

function of

$$\begin{aligned} w(\lambda) &= 1+\sigma u^T (D-\lambda I)^{-1} u \\ &= 1+\sigma \sum_{i=1}^{n} \frac{u_j^2}{d_j-\lambda} \end{aligned} \tag{4.12}$$

The corresponding eigenvectors $q_1, q_2, ..., q_n$ *are given by*

$$q_i = (D-\lambda_i I)^{-1} u / \parallel (D-\lambda_i I)^{-1} u \parallel_2 \tag{4.13}$$

This theorem restates results of [8, 22, 37, 42], where we refer the reader for more details on the proof.

The problem we are requiring to solve is that of the merging of the right and the left subproblem. For simplicity, let us describe the merging process for $p = 2$. However it is easy to generalize for $p > 2$, and we will give the general form at the end of this subsection. As mentioned above, each merging step consists of a sequence of rank-one updates.

$$\begin{aligned} A &= \begin{pmatrix} \tilde{B}_1 & 0 \\ 0 & \tilde{B}_2 \end{pmatrix} + \Sigma Z Z^T \\ &= \begin{pmatrix} \tilde{B}_1 & 0 \\ 0 & \tilde{B}_2 \end{pmatrix} + \sigma_1 z_1 z_1^T + \sigma_2 z_2 z_2^T + \cdots + \sigma_b z_b z_b^T \end{aligned} \tag{4.14}$$

Substituting (4.9) into (4.14) yields

$$A = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \sigma_1 z_1 z_1^T + \sigma_2 z_2 z_2^T + \cdots + \sigma_b z_b z_b^T \tag{4.15}$$

Then, the rank-b modification will be computed recursively as a sequence of b rank-one updates.

First rank-one update:

$$\begin{aligned} A &= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + \sigma_1 u_1 u_1^T \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \sigma_2 z_2 z_2^T + \cdots + \sigma_b z_b z_b^T \\ &= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left\{ \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix} \begin{pmatrix} \tilde{D}_1 & 0 \\ 0 & \tilde{D}_2 \end{pmatrix} \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix}^T \right\} \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \sigma_2 z_2 z_2^T + \cdots + \sigma_b z_b z_b^T \end{aligned} \tag{4.16}$$

where,

$$u_1 = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T z_1$$

and $\tilde{D}_i$ is the rank-one updated eigenvalues of D_i, and $\tilde{Q}$ are the eigenvectors resulting from the rank-one update.

Second rank-one update:

$$
\begin{aligned}
A &= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix} \left\{ \begin{pmatrix} \tilde{D}_1 & 0 \\ 0 & \tilde{D}_2 \end{pmatrix} + \sigma_2 u_2 u_2^T \right\} \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix}^T \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \cdots + \sigma_b z_b z_b^T \\
&= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix} \left\{ \begin{pmatrix} \hat{Q}_{11} & \hat{Q}_{12} \\ \hat{Q}_{21} & \hat{Q}_{22} \end{pmatrix} \begin{pmatrix} \hat{D}_1 & 0 \\ 0 & \hat{D}_2 \end{pmatrix} \begin{pmatrix} \hat{Q}_{11} & \hat{Q}_{12} \\ \hat{Q}_{21} & \hat{Q}_{22} \end{pmatrix}^T \right\} \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix}^T \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T + \cdots + \sigma_b z_b z_b^T
\end{aligned}
\tag{4.17}
$$

where,

$$
u_2 = \begin{pmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{21} & \tilde{Q}_{22} \end{pmatrix}^T \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T z_2
$$

and $\hat{D}_i$ is the rank-one updated eigenvalues of $\tilde{D}_i$ and $\hat{Q}$ are the eigenvectors resulting of the rank-one updates.

This process of rank-one updates is recursively applied to all the z_i rank-one vectors $1 \leq i \leq b$. Thus, at the

b[th] rank-one update:

$$
\begin{aligned}
A &= Q_1 \cdots Q_{b-2} Q_{b-1} \{D\} Q_{b-1}^T Q_{b-2}^T \cdots Q_1^T + \sigma_b z_b z_b^T \\
&= Q_1 \cdots Q_{b-2} Q_{b-1} \left\{ D + \sigma_b u_b u_b^T \right\} Q_{b-1}^T Q_{b-2}^T \cdots Q_1^T \\
&= Q_1 \cdots Q_{b-2} Q_{b-1} Q_b \{\Lambda\} Q_b^T Q_{b-1}^T Q_{b-2}^T \cdots Q_1^T
\end{aligned}
\tag{4.18}
$$

where,
$u_b = Q_1^T \cdots Q_{b-2}^T Q_{b-1}^T z_b$ and $\Lambda = diag\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ are the eigenvalues of A.

As a result, once the sequence of b rank-one updates has been computed, we obtain $\Lambda = diag\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$, which are the eigenvalues of the matrix A.

The whole algorithm for $p > 2$ can be easily described now. In this case, the binary tree has a depth $k = log_2(p)$ if p is a power of 2; $k = log_2(p) + 1$ otherwise. Thus, the tree consists of k levels. The leaves at the bottom of the tree (level=k) are the matrices $\tilde{B}_i$. Thus, as we mention above, the tree is traversed in a bottom-up fashion, where at each level l, 2^{l-1} independent merging problems are computed. Again, we notice that each merging step consists of a sequence of rank-one updates. Thus, after each rank-one update of the merging step, a sequence of matrix-vector products should be applied to take into account the synthesis of the intermediate eigenvector matrices. The desired eigenvalues of A are then computed at the top level $l = 1$ i.e., the root of the tree. Figure 4.1 depicts the amalgamation phase of the TD&C algorithm for a tree of depth $k = 3$, where the original matrix is split into eight subproblems.

4.5. Algorithmic Complexity. In this section, we calculate the number of floating-point operations (flops) required by the tile DC (TD&C) symmetric eigensolver algorithm. We recall that b corresponds to the tile and the bandwidth sizes. The matrix is divided into p blocks/tiles i.e., $p = n/b$ with n being the size of the matrix.

The algorithmic complexity of the full tridiagonal reduction is $\mathcal{O}(4/3n^3)$. The calculation of the number of flops for the band reduction step is then straightforward. It requires $4/3n \times (n-b) \times (n-b)$ flops and gets therefore closer to $\mathcal{O}(4/3n^3)$ for small bandwidth size b.

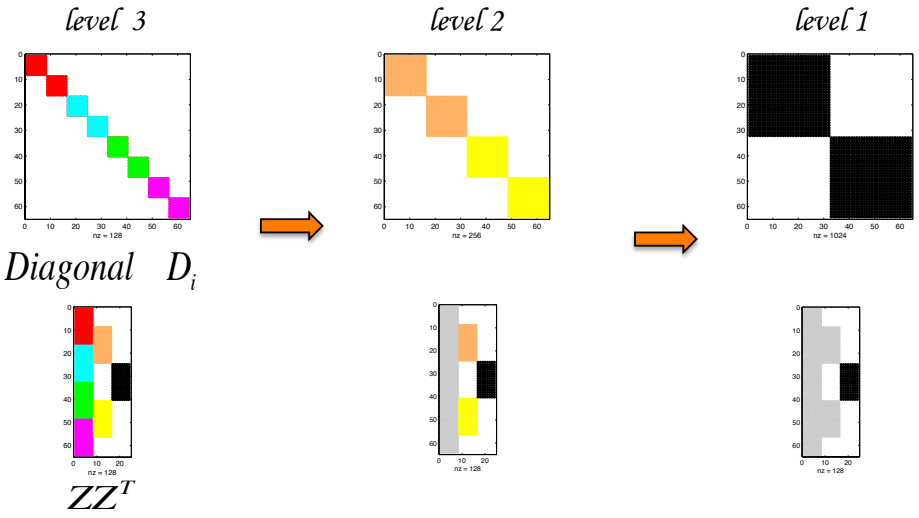

FIG. 4.1. *A description of the amalgamation procedure.*

The flop calculation for the band D&C algorithm to get the actual eigenvalues is more complicated. The flop count for each phase is reported below:

- *Partitioning phase:* this phase is characterized by the singular value decomposition of the off-diagonal blocks and the construction of the new diagonal blocks $\tilde{A}_i$ as defined by equation (4.7). The number of flops is then at most:
  $(p-1)\times(8/3)b^3+(p-1)\times 2\times 2b^3$ flops.
- *Subproblems solution phase:* this phase computes the spectral decompositions of each symmetric $\tilde{A}_i$ as described by Equation (4.9), which requires:
  $p\times(14/3)b^3$ flops.
- *Amalgamation phase:* each merging step of two-by-two leaves involves the application of a sequence of b rank-one modifications of diagonal matrices and thus, necessitates the computation of:
  - a rank-one vector $u_i = Q_i^T z_i$,
  - an eigen solution of $D+\sigma_i u_i u_i^T$.

  The cost of computing u_i is a matrix-vector product with all the previous Q_i^T accumulated from the deeper levels of the tree. The eigen decomposition of a rank-one modification problem $D+\sigma uu^T \in \mathbb{R}^{n\times n}$ demands two distinct computation steps: (1) the zero finding algorithm, which requires an average of $5n^2$ operations and (2) the calculation of n eigenvectors, which costs $2n^2$ operations. The number of flops involved in the eigen decomposition of a rank-one modification is $\mathcal{O}(7n^2)$. In practice, this number is the worst case scenario as the remarkable phenomenon of deflation takes place and may reduce dramatically this cost depending on the number of non-deflated vectors. Now, if we sum up all eigen decomposition computations occurring at all tree levels with a depth $k=\lfloor log_2(p)\rfloor$, the cost of the amalgamation step is:

$$=\sum_{i=0}^{k-1} 2^i\times\tfrac{b}{2}(b+1)\times 2(\tfrac{n}{2^i})^2+\sum_{i=1}^{k-1} 2^i\times b\times ib\times 2(\tfrac{n}{2^i})^2+\sum_{i=0}^{k-1} 2^i\times b\times 7(\tfrac{n}{2^i})^2$$

$$= \sum_{i=0}^{k-1} \frac{b}{2}(b+1) \times 2n^2(\tfrac{1}{2})^i + \sum_{i=1}^{k-1} b \times i \times b \times 2n^2(\tfrac{1}{2})^i + \sum_{i=0}^{k-1} b \times 7n^2(\tfrac{1}{2})^i$$

$$= 2b^2n^2 + 4b^2n^2 + 14bn^2$$

$$\approxeq \mathcal{O}(6\, b^2n^2) \text{ flops.}$$

The total arithmetic complexity of the band D&C algorithm will be dominated by the amalgamation phase and will require at most $\mathcal{O}(6\, b^2n^2)$ flops.

Finally, the cost of the TD&C for computing all eigenvalues of a dense symmetric matrix is $\mathcal{O}(4/3n^3 + 6\, b^2n^2)$. It is straightforward to see that the size of the bandwidth b will have a huge impact on the overall algorithm complexity. The TD&C symmetric eigensolver algorithm will be cost-effective if and only if $b \lll n$. Also, if all corresponding eigenvectors are to be calculated, the TD&C symmetric eigensolver will not be suitable as the cost will be dominated by the accumulation of the intermediate eigenvector matrices for each single rank-one modification. This would involve matrix-matrix multiplications and thus, the overall cost will substantially increase and become dominated by $\frac{8}{3}bn^3$ flops. Other methods, such as the inverse iteration, need to be explored if the eigenvectors are required.

The next section gives some details about the parallel implementation of the TD&C symmetric eigensolver algorithm and highlights our contributions.

5. Parallel Implementation Details. This section describes the major drawbacks of the standard D&C symmetric eigensolver parallel implementations found in the state-of-the-art numerical libraries as well as in some previous works. Also, our threefold main contributions are clearly laid out in this section.

5.1. The Standard D&C Parallel Implementations. Over the last decade, great efforts have been expended in building efficient numerical libraries for solving eigenvalue problems. Among these, the best known are LAPACK (for shared-memory machines) and SCALAPACK (extension to distributed memory). They rely on optimized BLAS and PBLAS respectively, which can be optimized for each computing environment. LAPACK and SCALAPACK are designed on top of the level 1, 2, and 3 BLAS/PBLAS, and nearly all of the parallelism in the LAPACK/SCALAPACK routines is contained in the BLAS/PBLAS. Their D&C symmetric eigensolvers require as a first step the transformation of the symmetric dense matrix into a tridiagonal form. This first reduction step is actually common with respect to other well-known methods i.e., QR iteration, bisection and MRRR. More details about these latter algorithms can be found in [14, 15, 16, 17, 21, 26, 27, 29, 33, 38]. There are mainly two bottlenecks with those approaches: (1) because of the underlying BLAS/PBLAS libraries, they follow the expensive fork/join paradigm which impedes the parallel efficiency and creates unnecessary synchronization points and (2) the panel factorization occurring during the reduction phase is completely memory-bound – the entire unreduced part of the matrix is loaded into memory to only perform at most level 2 BLAS operations. In other words, the processing time to get the data loaded to memory is actually higher than the computing time itself, which is not an option on multicore architectures. Some parallel implementations have been proposed to overcome some of these drawbacks [6]. However, the lack of flexibility of their runtime system environment do not permit a systematic way of removing the synchronization points and resolving load balancing issues that may occur while computing the eigenvalues.

5.2. Our Threefold Contributions. This section describes our contributions in three bullets: (1) Integration of a dynamic runtime system environment, (2) Improving the data locality while pursuing the critical path and (3) Removal of undesired synchronization points.

5.2.1. Dynamic Scheduling with QUARK. The main issue when writing a parallel algorithm, and perhaps the most critical, is how to distribute the data and the work across the processing units in an efficient way. We have integrated QUARK [28] – a dynamic runtime system environment – into our TD&C symmetric eigensolver algorithm. The whole algorithm can be represented as a DAG. The goal of QUARK is to dynamically schedule the different sequential tasks (nodes of the DAG) and to ensure the data dependencies are not violated (edges of the DAG). However, QUARK does not explicitly build the DAG prior to the execution for scalability purposes but rather unrolls it on the fly. Therefore, the execution flow is solely driven by the data dependencies. As soon as the dependencies are satisfied, QUARK initiates and executes the corresponding tasks on the available worker threads. Also, a dynamic-type scheduler is preferred over a static one especially, because the work of the TD&C algorithm depends strongly on the amount of deflations. QUARK is thus capable of reducing any load imbalances happening during runtime by applying for instance, work stealing strategies.

There are many other details about the internals of the scheduler including its dependency analysis, memory management, and other performance enhancements that are not covered here. However, information about this scheduler can be found in [28].

5.2.2. Enhancing Data Locality by Grouping Tasks. The reduction of the dense matrix to band form using tile algorithms is extremely parallel and relies on high performance kernels, which are composed by successive level 3 BLAS calls. In fact, the sequential kernel performance is contingent to the size of the tile i.e., the bandwidth b. Table 5.1 and Table 5.2 show the timing in seconds to reduce a dense matrix to band form with different tile/bandwidth sizes and the total number of tasks required for this reduction, respectively. The critical impact of b is reported in the second column of both tables. A small bandwidth b will dramatically affect the efficiency of the reduction phase and increase the elapsed time due to (1) the huge bus traffic required to load all the kernel calls into memory, (2) the inefficiency of the kernels on small b and (3) the runtime system overhead of scheduling very fine granularity tasks. At the same time, as analyzed in Section 4.5, the performance of the TD&C symmetric eigensolver on a band matrix strongly depends on the band size b. The smaller the b, the less the number of flops. This trade-off between achieving high performance for the reduction step and diminishing the number of flops of the TD&C symmetric eigensolver then has to be addressed.

matrix size	without task grouping technique					with task grouping technique				
	4000	8000	12000	16000	20000	4000	8000	12000	16000	20000
$b = 200$	2.6	13	40	90	169	2.6	13	40	90	170
$b = 100$	2.5	13	42	96	210	2.6	13	40	91	170
$b = 50$	8	62	214	524	>1000	2.6	15	44	102	194
$b = 25$	70	480	>1000	>1500	>2500	3.0	18	57	135	258
$b = 20$	112	765	>2000	>2500	>2500	3.0	20	65	144	277

TABLE 5.1
Timing comparison in seconds of the reduction phase, with and without the task grouping technique.

To overcome those three limitations, we have implemented a *task grouping technique*, which assembles small computational tasks and their data together, and feeds the runtime system environment QUARK with the new aggregated task instead. In Figure 5.1, we represent a schematic view of the task grouping technique between steps of the reduction phase. The different colored blocks of tiles correspond to different task groups. However, accumulating tasks may decrease the level of parallelism. Therefore, it is important to parametrize

the size of the tile block in order to sustain a sufficient number of concurrent tasks. This is also achieved by providing hints to the scheduler with task priority levels to ensure the pursuit of the critical path is not delayed and to release lots of concurrent aggregated tasks as soon as their dependencies are satisfied. The third column of Table 5.1 and Table 5.2 presents the results once the task grouping is enabled. The elapsed time is reduced by one order of magnitude and the number of tasks is decreased by two orders of magnitude compared to the original version. The task grouping technique permits to substantially improve the reduction phase while keeping a small bandwidth b and thus, allows the removal of all three constraints defined above.

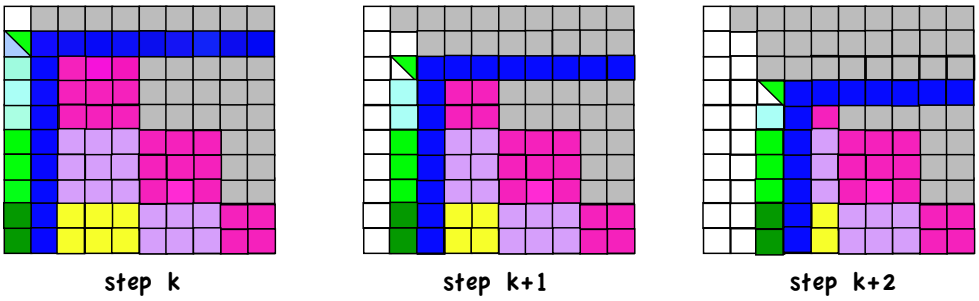

FIG. 5.1. *A schematic description of the task grouping technique.*

Finally, we illustrate in Figure 5.2 the snapshots of an execution trace of the reduction of a dense matrix with size $n = 4000$ to a band form of bandwidth $b = 20$, running on 8 threads using the standard TD&C algorithm (Top) without the task grouping technique or the modified TD&C algorithm with task grouping technique (Bottom). The Top figure shows the overhead of scheduling small tiles with lots of gaps between tasks. The Bottom figure depicts a tightly packed scheduling which corresponds to the effect of the task grouping technique.

5.2.3. Removing Synchronization Points. Last but not least, the standard D&C parallel implementation mainly requires two synchronization points located: (1) inside the fork/join paradigm and (2) between the different phases of the algorithm. Tile algorithms alleviate (1) by bringing the parallelism to the fore and relying on sequential high performance kernels with a smaller granularity. Since the execution flow of the TD&C symmetric eigensolver through QUARK is solely driven by the data dependencies, a task immediately proceeds after the dependencies coming from its parent are satisfied. This could potentially engender an out-of-order task execution where different phases of the algorithm could overlap, as long

matrix size	without task grouping technique					with task grouping technique				
	4000	8000	12000	16000	20000	4000	8000	12000	16000	20000
$b = 200$	3	21	72	170	333	3	21	72	170	333
$b = 100$	21	170	575	1,365	2,666	6	45	151	354	686
$b = 50$	170	1,365	4,607	10,922	21,332	14	98	317	735	1,414
$b = 25$	1,365	10,922	36,863	87,380	170,665	28	197	635	1,470	2,829
$b = 20$	2,666	21,332	71,999	170,665	333,332	35	246	794	1,837	3,536

TABLE 5.2

Comparison of the number of tasks during the reduction phase with and without the task grouping technique. Note that the value in the table are in thousands of tasks order.

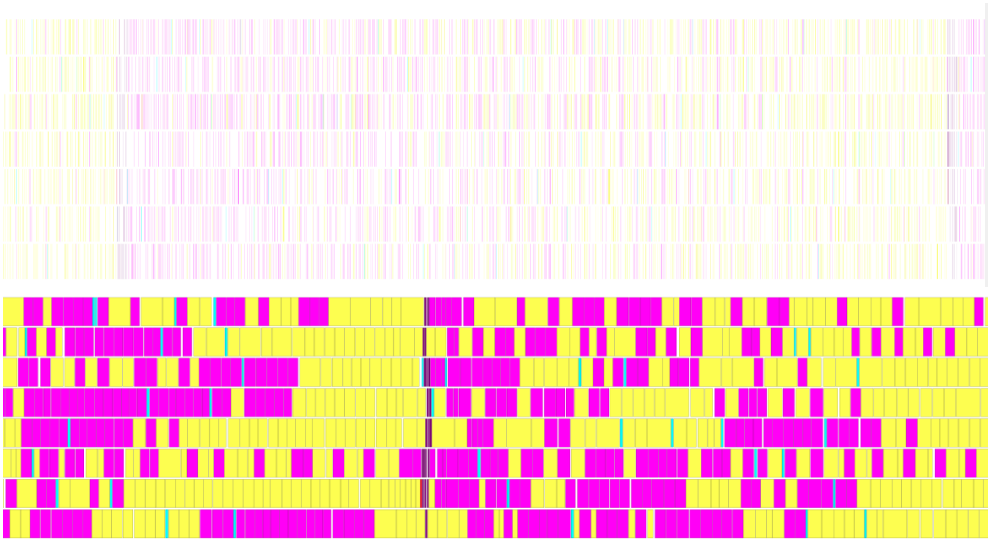

FIG. 5.2. *Execution trace of the reduction phase of a dense matrix of size* $n = 4000$ *with* $b = 25$ *on* 8 *threads. Top: TD&C algorithm without task grouping technique – Bottom: TD&C algorithm using task grouping technique.*

as the dependencies are not violated. Therefore, the unnecessary synchronization points between the different phases of the algorithm (2) are completely removed. Figure 5.3 shows a portion of the execution trace of the entire algorithm. Tasks from the reduction and the band D&C phases are completely overlapped. The band D&C phase starts before the dense matrix has completely achieved the band form.

FIG. 5.3. *The trace of the execution of the whole algorithm*

6. Performance Results and Accuracy Analysis. This section summarizes our main performance results of the tile DC (TD&C) symmetric eigensolver algorithm.

6.1. Machine Description. All of our experiments have been run on a shared-memory multicore architecture composed by a quad-socket quad-core Intel Xeon EMT64 E7340 processor operating at 2.39 GHz. The theoretical peak is equal to 9.6 Gflop/s/ per core or 153.2 Gflop/s for the whole node, composed of 16 cores. The practical peak achieved with DGEMM on a single core is equal to 8.5 Gflop/s or 136 Gflop/s for the 16 cores. The level-1 cache, local to the core, is divided into 32 kB of instruction cache and 32 kB of data cache. Each

quad-core processor is actually composed of two dual-core Core2 architectures and the level-2 cache has 2×4 MB per socket (each dual-core shares 4 MB). The machine is a NUMA architecture and it provides Intel Compilers 11.0 together with the Intel MKL V10.2 vendor library.

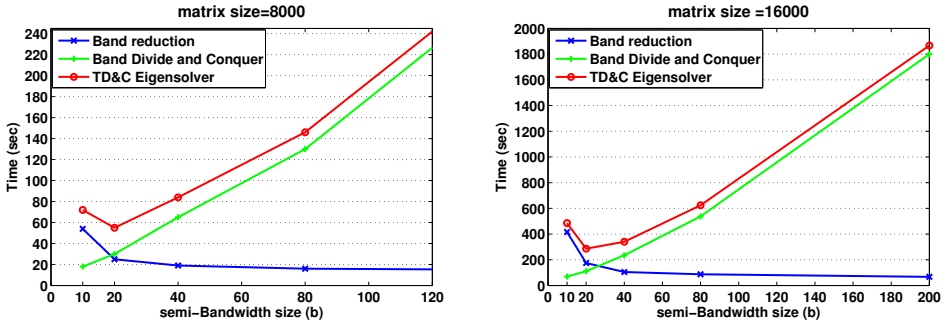

FIG. 6.1. *Effect of the tile/band size.*

6.2. Tuning the Tile/Band Size b. From Sections 4 and 5, it is clear that the tile/band size is the paramount parameter. Figure 6.1 highlights the effect of this parameter on the reduction phase (blue curve), on the band D&C phase (green curve) and on the overall TD&C eigensolver (red curve). For a very large b, the band D&C phase is the dominant part of the general algorithm and reciprocally, for a very small b, the reduction phase governs the whole application. From these experiments, a tile size $b = 20$ looks to be the best compromise for a matrix of order less than 20000, while $30 < b < 40$ appears to be the best choice for matrix of larger order.

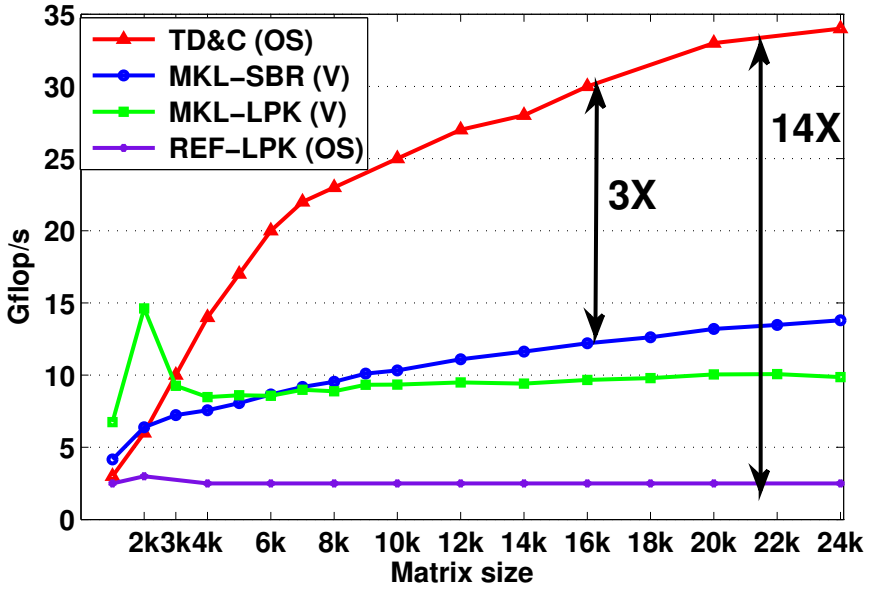

FIG. 6.2. *Performance comparisons between open-source (OS) and vendor (V) libraries.*

6.3. Performance Comparisons with other Symmetric D&C Implementations. This section shows the performance results of our TD&C symmetric eigensolver and compares them against the similar routine (DSTEDC) available in the state-of-the-art open source and vendor numerical libraries i.e., multi-threaded reference LAPACK V3.2 with optimized Intel MKL BLAS and Intel MKL V10.2, respectively. Since DSTEDC operates on a tridiagonal matrix, the dense symmetric matrix first needs to be reduced to tridiagonal form. Intel MKL actually provides two interfaces to reduce the symmetric dense matrix to tridiagonal form. The first corresponds to the optimized version of DSYTRD from LAPACK (MKL-LAPACK) and the second one integrates the optimized version of the corresponding routine named DSYRDD form the Successive Band Reduction toolbox (SBR) [7] (MKL-SBR). Figure 6.2 summarizes the experiments in Gflop/s. The number of flops used as a reference is $4/3n^3$, which is roughly the total number of flops for the tridiagonal reduction, since the D&C eigensolver of a tridiagonal matrix is of order n^2. All experiments have been performed on 16 cores with random matrices in double precision arithmetic. The performance results show that the proposed TD&C symmetric eigensolver achieves up to 14X speed up compared to the reference LAPACK implementation and up to 3X speed up as compared to the vendor Intel MKL library.

The next sections describe a collection of different matrix types and compare our TD&C symmetric eigensolver against D&C (DSTEDC) as well as other methods to compute the eigenvalues of a symmetric dense matrix e.g., the bisection BI (DSTEVX), QR (DSTEV) and MR (DSTEMR). For simplicity, we compare in the experiments shown in the next sections, our implementation with those four routines from the Intel MKL library, and skip the comparison with the reference LAPACK library.

6.4. Description of the Matrix Collections. In this section, we present the different matrix collections used during the extensive testing to get performance results (Section 6.5) as well as the numerical accuracy analysis (Section 6.6). There are three matrix collections:

- The matrices from the first set represented in Table 6.1, are "*synthetic testing matrices*" chosen to have extreme distributions of eigenvalues as well as other specific properties that exhibit the strengths and weaknesses of a particular algorithm.
- The second set of matrices are "*matrices with interesting properties*", which are represented in Table 6.2. Those matrices will only be used for the accuracy analysis in Section 6.6 since they are already in tridiagonal form.
- The third set are "*practical matrices*" which are based on a variety of practical applications, and thus are relevant to a large group of users. Some of these matrices are described in Table 6.3.

More detailed information about these matrix collections can be found in [12, 13, 31].

6.5. Performance Comparisons with other Symmetric Eigensolver Implementations.

6.5.1. Understanding the Various Graphs. All experiments have been performed on 16 cores with matrix types from Table 6.1 and Table 6.3. The matrix sizes vary from 1000 up to 24000 and the computation is done in double precision arithmetic. Our TD&C symmetric eigensolver is compared against the bisection BI (DSTEVX), QR (DSTEV), MRRR or MR (DSTEMR) as well as the D&C (DSTEDC) symmetric eigensolvers. Similarly to DSTEDC, the other eigensolver implementations necessitate the matrix being reduced into tridiagonal form using either the MKL LAPACK-implementation routine DSYTRD or the MKL SBR-implementation routine DSYRDD.

There are two types of graphs: (1) *speed up graphs* obtained by computing the ratio between the elapsed time of these four methods over the TD&C total execution time – in

Type	Description
Type 1	$\lambda_1 = 1,\ \lambda_i = \frac{1}{k},\ i = 2,3,\cdots n$
Type 2	$\lambda_i = 1,\ i = 2,3,\cdots n-1,\ \lambda_n = \frac{1}{k}$
Type 3	$\lambda_i = k^{-(\frac{i-1}{n-1})},\ i = 2,3,\cdots n$
Type 4	$\lambda_i = 1-(\frac{i-1}{n-1})(1-\frac{1}{k}),\ i = 2,3,\cdots n$
Type 5	n random numbers in the range $(\frac{1}{k},k)$, their logarithms are uniformly distributed
Type 6	n random numbers from a specified distribution
Type 7	$\lambda_i = ulp \times i,\ i = 2,3,\cdots n-1,\ \lambda_n = 1$
Type 8	$\lambda_i = ulp,\ \lambda_i = 1+i\times\sqrt{ulp},\ i = 2,3,\cdots n-1,\ \lambda_n = 2$
Type 9	$\lambda_1 = 1,\ \lambda_i = \lambda_{i-1}+100\times ulp,\ i = 2,3,\cdots n$

TABLE 6.1

Synthetic testing matrices from LAPACK *testing (Type 1-6) and from [13] (Type 7-9). Note that for distributions of Type 1-5, the parameter k has been chosen to be equal to the machine precision ulp in double precision arithmetic. We use the* LAPACK *routines* DLATMS *or* DLAGSY *to generate* $A = QDQ^T$. *Given the eigenvalues* λ, *the dense matrix A is generated by multiplying* $D = diag(\lambda)$ *by an orthogonal matrix Q generated from random entries.*

Type	Description
Type 10	(1,2,1) tridiagonal matrix
Type 11	Wilkinson-type tridiagonal matrix
Type 12	Clement-type tridiagonal matrix
Type 13	Legendre-type tridiagonal matrix
Type 14	Laguerre-type tridiagonal matrix
Type 15	Hermite-type tridiagonal matrix

TABLE 6.2

Matrices with interesting properties.

Type	Description
Type 15	Matrices from application on quantum chemistry and electronic structure
Type 16	The bcsstruc1 set in Harwell-Boeing collection
Type 18	Matrices from Alemdar, NASA, and cannizzo, ... etc. sets in the University of Florida

TABLE 6.3

Matrices from real life applications.

other words, a "*value t above* 1" means that our TD&C is "*tx faster*" than the corresponding algorithm and a "*value t below* 1" means that our TD&C is "*tx slower*" than the corresponding algorithm and (2) *performance graphs* in Gflop/s, which show how much of the peak performance the different symmetric eigensolvers are able to reach. Also, we note that the number of flops used as a **reference is 4/3**n^3. The color and symbol codes used for all plots are as follows: BI curves are green, using "x", QR curves are blue, using "*", MR curves are magenta using "$\diamond$", D&C curves are black, using "o", and TD&C curves are red and are either considered as a reference (speed up graphs) or use "o" to mark data points (performance graphs).

6.5.2. Synthetic testing matrices (Table 6.1). Figure 6.3 and Figure 6.4 show respectively the speed up and the performance in Gflop/s obtained by the TD&C symmetric eigensolver as compared to BI, QR, MR and D&C when the MKL LAPACK-implementation routine DSYTRD is used to reduce the first stage.

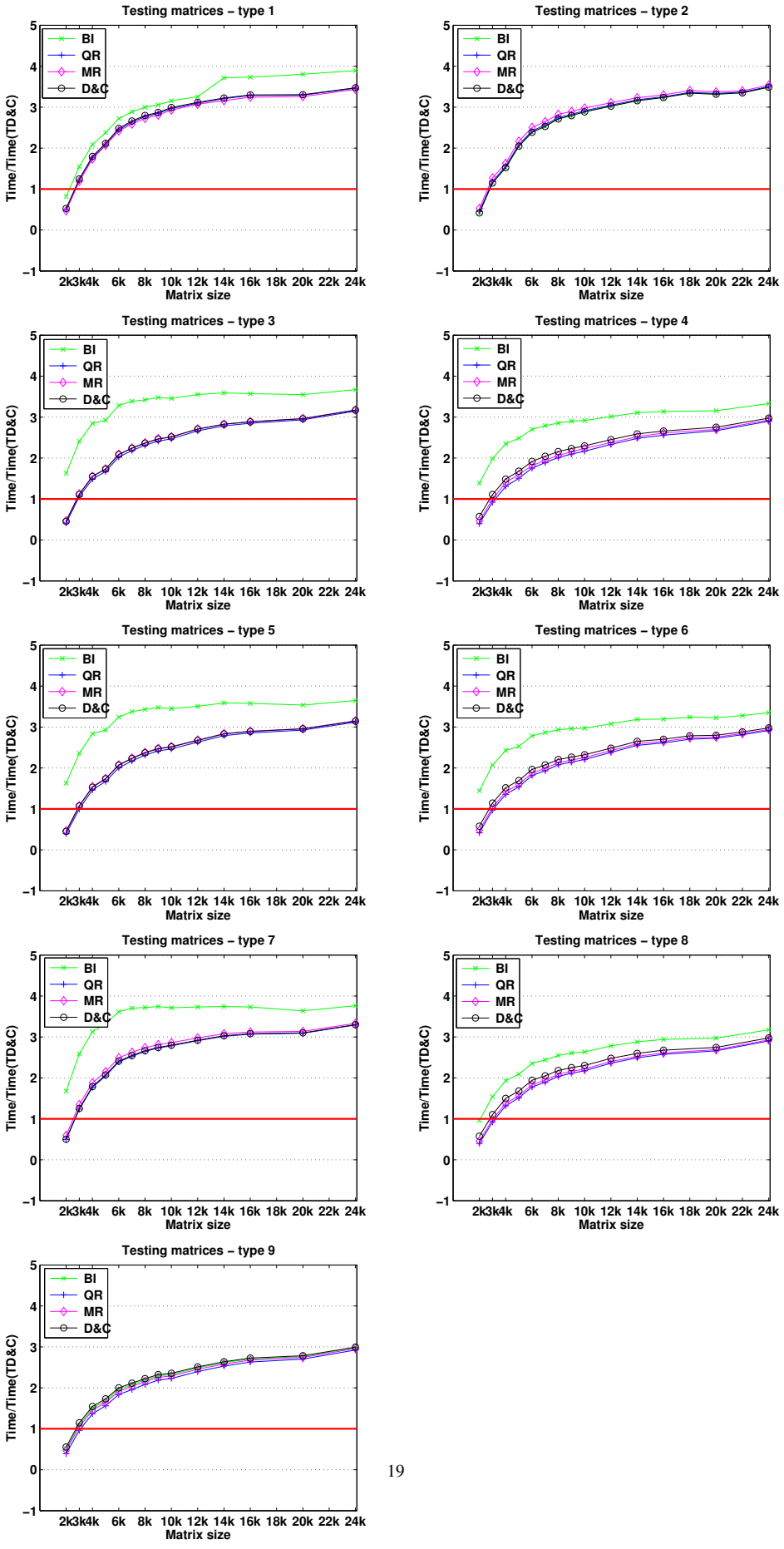

FIG. 6.3. *TD&C Speed up as compared to BI, QR, MR, and D&C for all synthetic testing matrices from Table 6.1. For BI, QR, MR, and D&C, the MKL LAPACK-implementation routine DSYTRD is used to transform the dense symmetric matrix A to tridiagonal.*

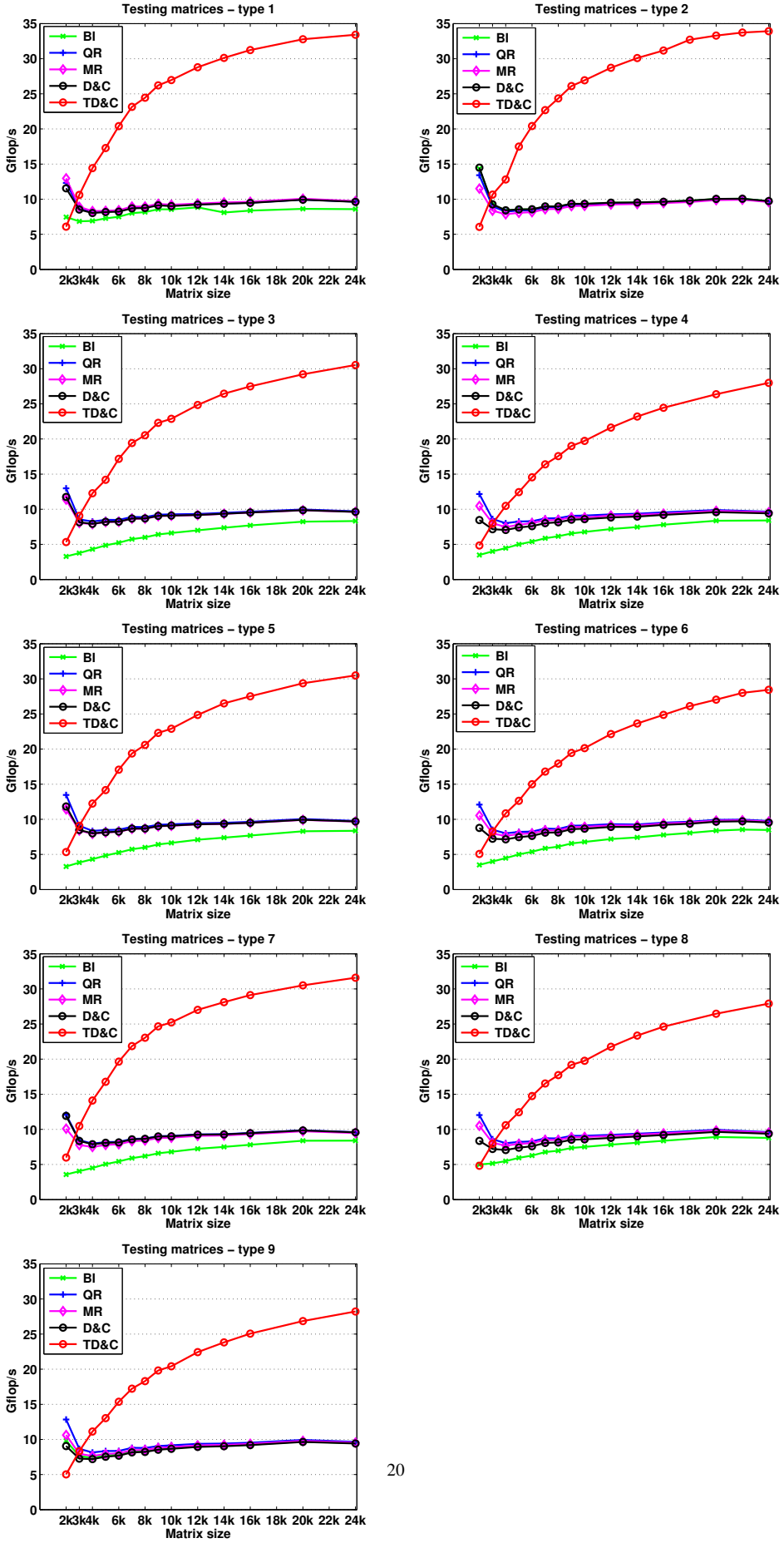

FIG. 6.4. *TD&C Performance comparison in Gflop/s for all synthetic testing matrices from Table 6.1. For BI, QR, MR, and D&C, the MKL LAPACK-implementation routine DSYTRD is used to transform the dense symmetric matrix A to tridiagonal.*

Figure 6.5 and Figure 6.6 show respectively the speed up and the performance in Gflop/s obtained by the TD&C symmetric eigensolver as compared to BI, QR, MR and D&C when the MKL SBR-implementation routine DSYRDD is used to reduce the first stage.

The results show that the TD&C symmetric eigensolver algorithm performs better than all other represented algorithms and for all synthetic testing matrices for $n \geq 3000$. The results also indicate that the TD&C speed up ranges from 2x faster for matrices of size $3000 \leq n \leq 10000$ (27 Gflop/s) to 4x faster for matrices of size $n \geq 10000$ (35 Gflop/s). For small matrix sizes with $n \leq 3000$, TD&C runs slower than the other eigensolvers. It is clear, from the algorithmic complexity study in Section 4.5, that our TD&C algorithm requires $4/3n^3 + 6b^2n^2$ flops, which for example, for a $b = 20$ and $n = 3000$ means $4/3n^3 + 2400n^2 \approxeq 2n^3$ flops, while the other algorithms roughly perform $4/3n^3$ flops.

On the other hand, BI seems to be the slowest algorithm for computing eigenvalues although its efficiency mostly depends on the distribution of the eigenvalues. Indeed, for matrices with strongly clustered eigenvalues (e.g., type 2), BI performance is comparable with the other symmetric eigensolver methods.

Furthermore, the TD&C symmetric eigensolver is able to take full advantage of a situation where a significant amount of deflations occurs for special matrices of type 1 and type 2. By avoiding unnecessary calculations especially in the amalgamation phase thanks to deflations, the TD&C runs even at higher performance. This is also emphasized in Table 6.4 and Table 6.5. These tables report the detailed execution time to compute all eigenvalues with BI, QR, MR, D&C and TD&C using matrices of type 2 and type 6, respectively. While a type 6 matrix represents the worst case scenario for the TD&C symmetric eigensolver in which less than 2% of deflations happen, a type 2 matrix corresponds to a matrix where a considerable amount of deflation occurs.

Matrix	MKL SBR tridiag DSYRDD +				MKL LAPACK tridiag DSYTRD +				TD&C
size	BI	QR	MR	D&C	BI	QR	MR	D&C	
2000	1.6	1.7	1.8	1.6	0.8	0.8	0.9	0.7	1.7
3000	5.0	5.2	5.4	5.0	4.0	4.0	4.3	3.9	3.3
4000	11	11	12	11	10	10	10	10	6.6
5000	20	20	21	20	19	19	20	19	9.5
6000	33	33	34	33	34	34	35	33	14
7000	50	50	52	50	51	51	53	50	20
8000	71	72	74	71	77	77	79	76	28
9000	95	96	99	95	105	105	107	103	37
10000	127	129	132	127	144	144	147	142	49
12000	209	211	215	209	244	244	249	242	80
14000	315	318	324	315	386	386	392	383	121
16000	446	449	458	446	570	570	578	566	**175**
18000	616	620	631	616	798	798	809	794	237
20000	820	825	838	820	1068	1068	1080	1062	324
22000	1054	1062	1074	1054	1417	1417	1430	1410	421
24000	1339	1345	1362	1337	1878	1878	1896	1870	553

TABLE 6.4

Elapsed time in seconds for computing all eigenvalues of matrices with strongly clustered eigenvalues (type 2), when varying the size from 2000 to 24000.

The band D&C considerably performs less flops in the latter case and thus, it runs more than twice as fast as when no deflations take place. For example, for a matrix of size $n =$

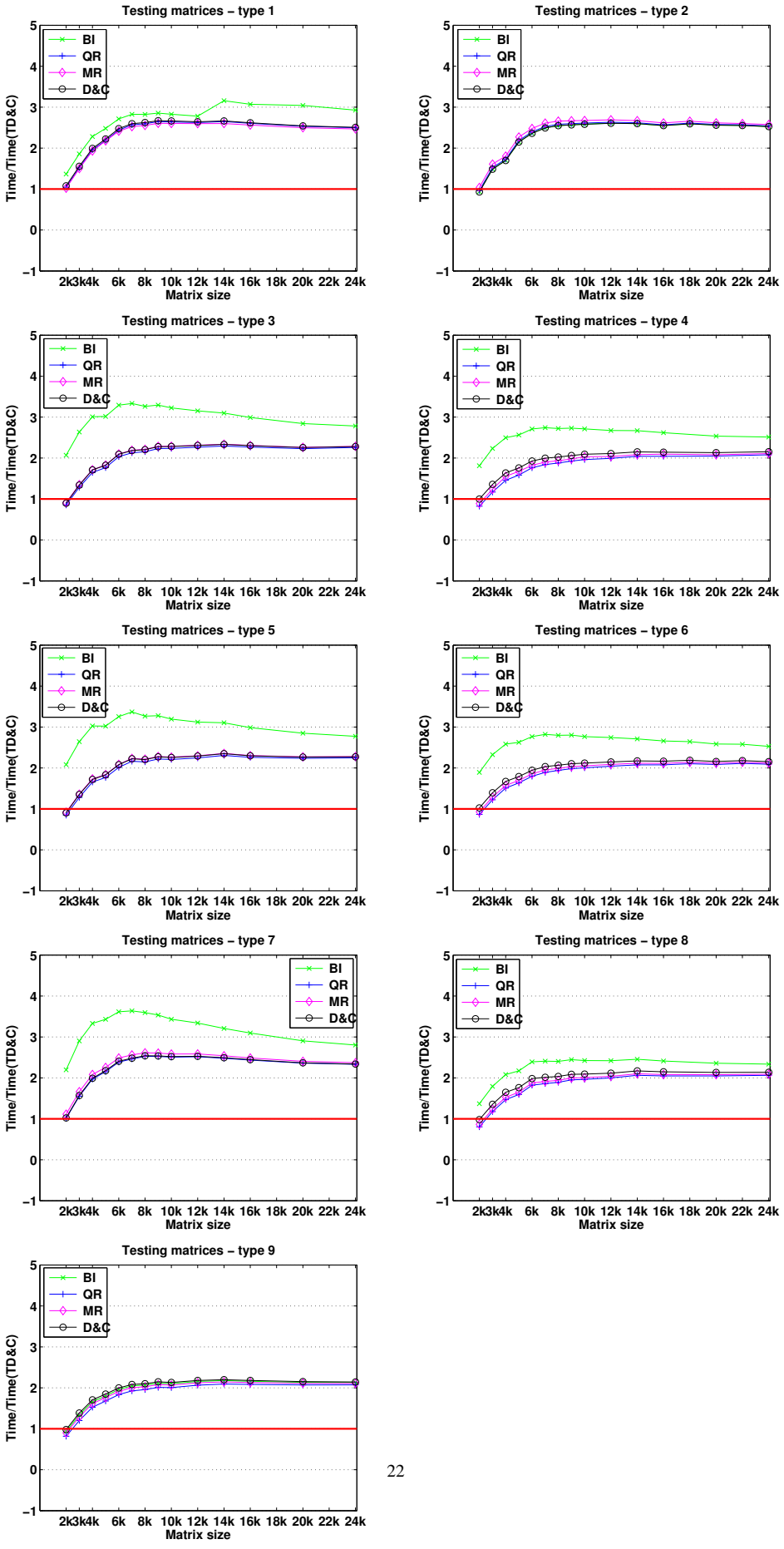

FIG. 6.5. *TD&C Speed up as compared to BI, QR, MR, and D&C for all synthetic testing matrices from Table 6.1. For BI, QR, MR, and D&C, the MKL SBR-implementation routine DSYRDD is used to transform the dense symmetric matrix A to tridiagonal.*

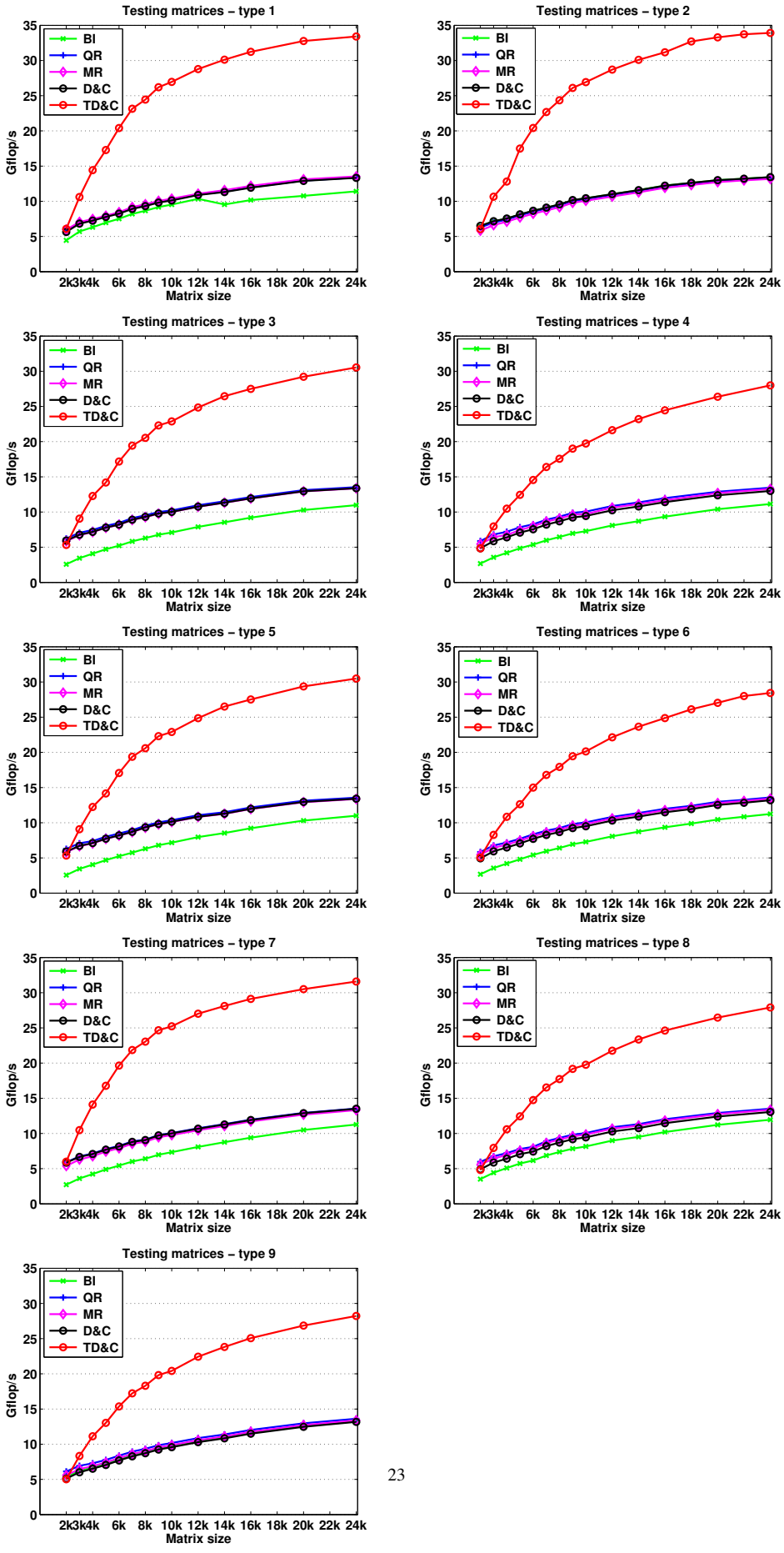

FIG. 6.6. *TD&C Performance comparison in Gflop/s for all synthetic testing matrices from Table 6.1. For BI, QR, MR, and D&C, the MKL SBR-implementation routine DSYRDD is used to transform the dense symmetric matrix A to tridiagonal.*

16000, the reduction to band form requires 145 seconds. The band D&C phase with lots of deflations requires 30 seconds instead of 74 seconds when no deflations occur, giving a total execution time of 175 seconds (see Table 6.4) and 219 seconds (see Table 6.5), respectively (1.3X overall speed up).

From a global point of view, the elapsed time and the performance results of the other methods (BI, QR, MR and D&C) are very similar to one another. In addition to that, in the presence of deflation, D&C has only improved relatively compared to TD&C. Therefore, once again, it is important to mention that when only eigenvalues are computed, BI, QR, MR and D&C count only for $O(n^2)$ on the overall algorithm complexity and the leading term corresponding to the tridiagonal reduction phase is about $4/3n^3$. The time spent in this reduction phase can be as high as 90% of the total execution time when only eigenvalues are needed but more than 50% when eigenvectors are to be computed. Thus, no matter how D&C or the other methods get improved, the impact on the overall algorithm performance is rather negligible for large matrix sizes.

6.5.3. Practical matrices (Table 6.3). The most important testing matrices are perhaps those, which come from real applications. Figure 6.7 and Figure 6.8 draw respectively the speed up and the performance in Gflop/s obtained by the TD&C symmetric eigensolver as compared to BI, QR, MR and D&C for a set of more than 100 matrices arising in different scientific and engineering areas. Similarly to synthetic testing matrices, the TD&C symmetric eigensolver algorithm out-performs all the other symmetric eigensolvers, between two to three times faster, and achieves up to 36 Gflop/s.

6.6. Accuracy Analysis. This section is dedicated to the analysis of the TD&C symmetric eigensolver accuracy as compared to the other four symmetric eigensolvers: QR, BI, MR and D&C.

Matrix	MKL SBR tridiag DSYRDD +				MKL LAPACK tridiag DSYTRD +				TD&C
size	BI	QR	MR	D&C	BI	QR	MR	D&C	
2000	3.9	1.8	1.9	2.1	3.0	0.9	1.0	1.2	2.1
3000	10	5.3	5.6	6.1	9.0	4.2	4.5	4.9	4.3
4000	20	11	12	13	19	10	11	11	7.8
5000	34	21	22	23	33	20	21	22	13
6000	53	34	35	37	53	34	36	37	19
7000	77	51	53	55	77	52	54	56	27
8000	106	73	75	78	111	79	81	84	38
9000	139	98	100	104	148	106	109	113	49
10000	181	131	134	138	196	146	149	153	66
12000	287	214	218	224	320	247	252	258	104
14000	420	322	328	337	493	395	401	410	154
16000	583	454	462	474	701	573	581	593	**219**
18000	786	626	636	651	965	804	814	829	297
20000	1031	833	845	863	1272	1074	1086	1104	394
22000	1307	1070	1084	1105	1663	1425	1440	1461	507
24000	1639	1355	1372	1397	2172	1888	1906	1930	648

TABLE 6.5

Elapsed time in seconds for computing all eigenvalues of matrices with uniform eigenvalue distributions (type 6), when varying the size from 2000 to 24000.

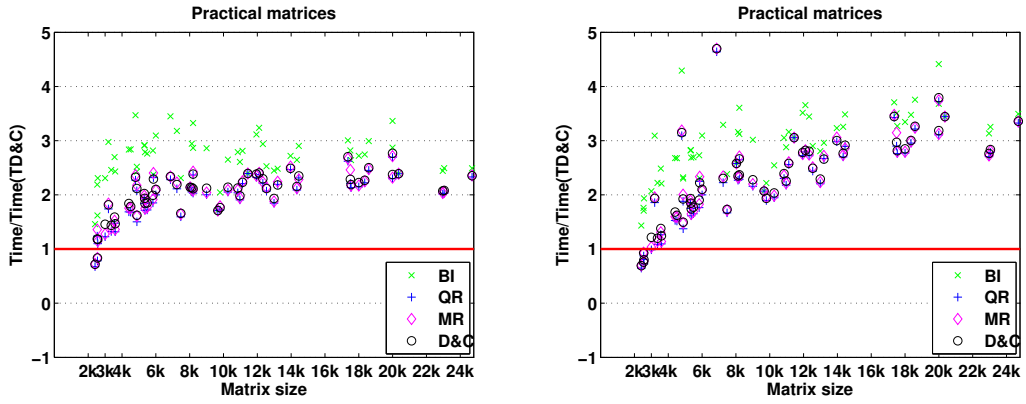

FIG. 6.7. *Timings comparison for practical matrices. The tridiagonalisation has been done either by the MKL-SBR (left) or by the MKL-LAPACK (right).*

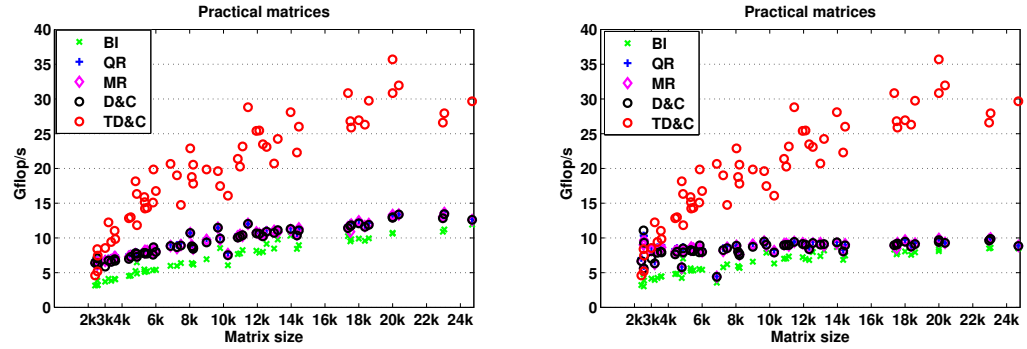

FIG. 6.8. *Performance comparison for practical matrices. The tridiagonalisation has been done either by the MKL-SBR (left) or by the MKL-LAPACK (right).*

6.6.1. Metric Definitions. For a given symmetric matrix B, computed eigenvectors $Q = [q_1, q_2, \cdots, q_n]$ and their corresponding eigenvalues $\Lambda = diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$ we use the following accuracy tests by using the LAPACK testing routines (DLANSY, DSTT21 for tridiagonal and DSYT21 for dense symmetric):

$$\frac{\|I - QQ^T\|}{n \times ulp} \tag{6.1}$$

which measures the orthogonality of the computed eigenvectors, and

$$\frac{\|B - Q\Lambda Q^T\|}{\|B\| n \times ulp} \tag{6.2}$$

which measures the accuracy of the computed eigenpairs, and

$$\frac{\|\lambda_i - \delta_i\|}{\|\lambda_i\| \times ulp} \tag{6.3}$$

which measures the accuracy of the computed eigenvalues compared to the reference eigenvalue δ which are the exact eigenvalues (analytically known) or the ones computed by the

QR iteration routine using LAPACK (DSTEV). The value ulp represents the machine precision computed by the LAPACK subroutine DLAMCH. Its value on the Intel Xeon where the accuracy experiments were conducted is equal to 2.220446049259313E-16.

For most of the tests presented here, the original matrix is either generated from a multiplication of a given diagonal eigenvalue matrix by an orthogonal matrix or given directly as a symmetric matrix. Then, for the standard algorithms (QR, BI, MR and D&C), the symmetric dense matrix is first reduced to tridiagonal form using the DSYTRD routine and then solved using one of the algorithms cited above. While for our TD&C symmetric eigensolver, the original matrix is reduced to band form, and then solved using the band D&C method according to the approach described in this paper. Therefore, the matrix B considered in the metrics (6.1), (6.2), (6.3), is either tridiagonal obtained from DSYTRD routine or band tridiagonal obtained from our TD&C symmetric eigensolver.

Type	Residual (6.1)					orthogonality (6.2)					eigenvalue (6.3)				
	QR	BI	MR	D&C	TD&C	QR	BI	MR	D&C	TD&C	QR	BI	MR	D&C	TD&C
Type 1	.001	.001	.001	.001	.001	.64	.01	20	.16	.10	6	4	9	5	9
Type 2	.48	.05	18.8	.275	.112	.42	.02	27	.16	.36	86	31	58	31	42
Type 3	.13	.02	1.35	.055	.083	.67	.27	157	.32	1.0	52	23	24	24	26
Type 4	.62	.28	7.62	.570	.571	.89	.36	47	.77	1.3	17	14	12	12	14
Type 5	.09	.02	5.52	.101	.129	.59	.36	87	.21	.96	20	20	19	18	22
Type 6	.60	.38	45.7	.528	1.02	.83	.33	132	.73	1.6	28	27	28	28	29
Type 7	.001	.38	.003	.001	.439	.93	.10	21	.15	.39	4	2	8	4	5
Type 8	.40	.15	10.5	.196	.723	.83	.34	23	.43	1.6	18	17	17	16	17
Type 9	.81	.34	44.7	.128	.955	.74	.33	45	.12	.94	40	37	37	37	39

TABLE 6.6
Summary of accuracy results observed on testing matrices from LAPACK *and [13].*

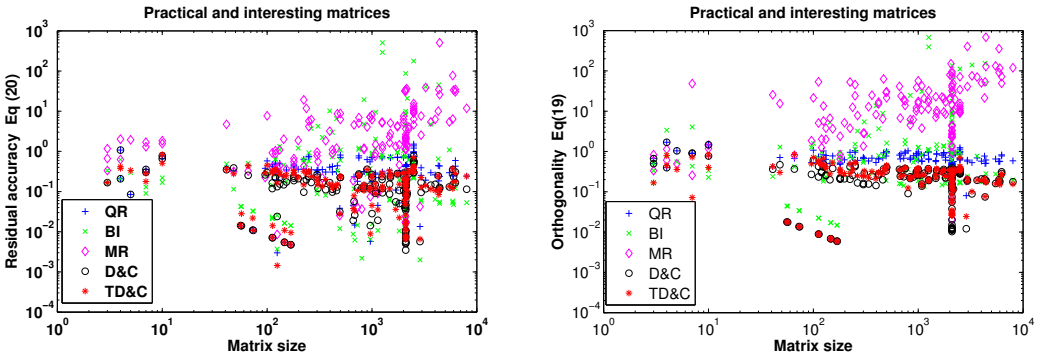

FIG. 6.9. *Summary of accuracy results observed on matrices from Table 6.2 and Table 6.3.*

6.6.2. Accuracy Evaluations. Table 6.6 presents the results obtained using the accuracy metrics defined above, for all sets of the testing matrices described in the subsection 6.4. Our TD&C symmetric eigensolver provides the three metrics with the same order of magnitude as compared to the other eigensolvers.

Figure 6.9 and Figure 6.10 depict the maximal residual norm (6.1) and the losses of orthogonality (6.2) for practical and interesting matrix categories and synthetic testing matrix type, respectively. These figures allow the study of the errors with respect to the matrix size

n. The error of our TD&C symmetric eigensolver decreases when n increases, similarly to QR and D&C eigensolvers. The observed accuracy is of order of $\mathcal{O}(\sqrt{n\varepsilon})$. However, it is clear that MR and BI do not achieve the same level of accuracy as QR, D&C and TD&C. Finally, after measuring the residual norms (Eq 6.1), the eigenvector orthogonality (Eq 6.2) and the eigenvalue accuracy (Eq 6.3), our TD&C framework is one of the most accurate symmetric eigensolver algorithms which gives us a certain confidence on the quality of the overall TD&C symmetric eigensolver presented in this paper.

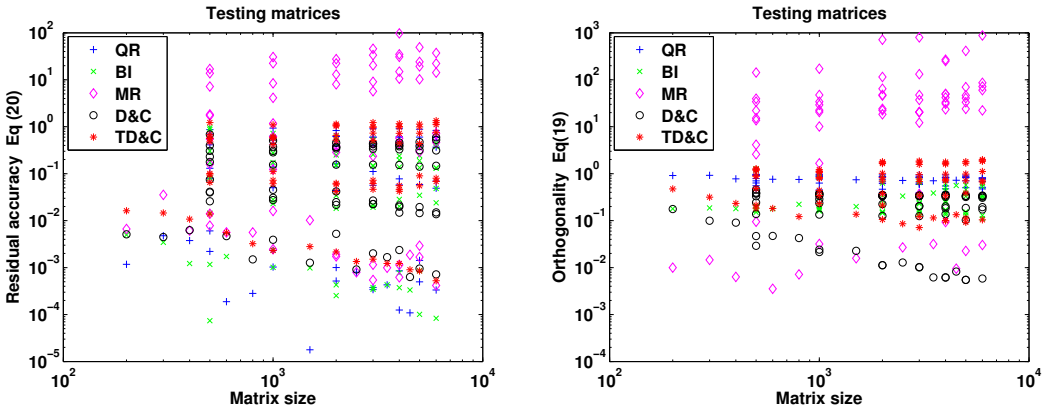

FIG. 6.10. *Summary of accuracy results observed on synthetic testing matrices from Table 6.1.*

7. Summary and Future Work. The Tile Divide and Conquer symmetric eigensolver (TD&C) presented in this paper shows very promising results in terms of performance on multicore architecture as well as in terms of numerical accuracy. TD&C has been extensively tested using different matrix types against other well-known symmetric eigensolvers such as the QR iteration (QR), the bisection (BI), the standard divide and conquer (D&C), and the multiple relatively robust representations (MRRR). The performance results obtained for large matrix sizes and certain matrix types are very impressive. The proposed TD&C symmetric eigensolver reaches up to 14X speed up compared to the state-of-the-art numerical open source library LAPACK V3.2 and up to 4X speed up against the commercial numerical library Intel MKL V10.2. Our TD&C symmetric eigensolver also proves to be one of the most accurate symmetric eigensolvers available. The authors plan to eventually integrate this symmetric eigensolver within the PLASMA library [39]. The authors are also currently looking at the eigenvector calculations. Although the TD&C symmetric eigensolver is not really appropriate for that purpose, a possible hardware solution to overcome this bottleneck would be the use of an accelerator such as a GPU, which could substantially improve the overall performance in a case when the eigenvectors are required. A software solution would potentially be the inverse iteration method in which the eigenvectors are computed from the eigenvalues using an iterative algorithm.

REFERENCES

[1] `http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm`.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, third edition, 1999.

[3] P. Arbenz. Divide and conquer algorithms for the bandsymmetric eigenvalue problem. *Parallel Computing*, 18(10):1105–1128, 1992.

[4] P. Arbenz, W. Gander, and G. H. Golub. Restricted rank modification of the symmetric eigenvalue problem: Theoretical considerations. *Linear Algebra Appl.*, 104:75–95, 1988.

[5] P. Arbenz and G. H. Golub. On the spectral decomposition of hermitian matrices modified by low rank perturbations. *SIAM J. Matrix Anal. Appl.*, 9(1):40–58, 1988.

[6] Y. Bai and R. C. Ward. A parallel symmetric block-tridiagonal divide-and-conquer algorithm. *ACM Trans. Math. Softw.*, 33(4):25, 2007.

[7] C. H. Bischof, B. Lang, and X. Sun. The sbr toolbox - software for successive band reduction, 1996.

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

[9] A. Buttari, J. Langou, J. Kurzak, and J. J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parellel Comput. Syst. Appl.*, 35:38–53, 2009.

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

[11] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[12] J. W. Demmel, O. A. Marques, B. N. Parlett, and C. Vmel. Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers. Technical Report 183, LAPACK Working Note, Apr. 2007.

[13] I. S. Dhillon. *A new O (N(2)) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*. PhD thesis, Berkeley, CA, USA, 1998. UMI Order No. GAX98-03176.

[14] I. S. Dhillon and B. N. Parlett. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and its Applications*, 387:1 – 28, 2004.

[15] I. S. Dhillon, B. N. Parlett, and C. Vömel. The design and implementation of the mrrr algorithm. *ACM Trans. Math. Softw.*, 32:533–560, December 2006.

[16] J. J. Dongarra and D. C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Statist. Comput.*, 8:s139–s154, 1987.

[17] J. G. F. Francis. The QR transformation: A unitary analogue to the LR transformation, parts I and II. *Comput. J.*, 4:265–272, 332–345, 1961.

[18] W. N. Gansterer, J. Schneid, and C. Ueberhuber. A low-complexity divide-and-conquer method for computing eigenvalues and eigenvectors of symmetric band matrices. *BIT Numerical Mathematics*, 41:967–976, 2001. 10.1023/A:1021933127041.

[19] W. N. Gansterer, R. C. Ward, and 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):45–58, 2002.

[20] K. Gates and P. Arbenz. Parallel divide and conquer algorithms for the symmetric tridiagonal eigenproblem, 1994.

[21] W. J. Givens. Numerical computation of the characteristic values of a real symmetric matrix. Technical Report ORNL-1574, Oak Ridge National Laboratory, Oak Ridge, TN, USA, 1954.

[22] G. H. Golub. Some modified matrix eigenvalue problems. *SIAM Review*, 15(2):pp. 318–334, 1973.

[23] R. Grimes, H. Krakauer, J. Lewis, H. Simon, and S.-H. Wei. The solution of large dense generalized eigenvalue problems on the cray x-mp/24 with ssd. *J. Comput. Phys.*, 69:471–481, April 1987.

[24] M. Gu and S. C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.*, 16(1):172–191, 1995.

[25] L. C. F. Ipsen and E. R. Jessup. Solving the symmetric tridiagonal eigenvalues problem on the hypercube. *SIAM J. Sci. Stat. Comput.*, 11:203–229, March 1990.

[26] E. R. Jessup and D. C. Sorensen. A parallel algorithm for computing the singular value decomposition of a matrix. *SIAM J. Matrix Anal. Appl.*, 15(2):530–548, 1994.

[27] V. N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Computational Mathematics and Mathematical Physics*, 1(3):637 – 657, 1962.

[28] J. Kurzak and J. Dongarra. Fully dynamic scheduler for numerical scheduling on multicore processors. Technical Report LAWN (LAPACK Working Note) 220, UT-CS-09-643, Innovative Computing Lab, University of Tennessee, 2009.

[29] S. Lo, B. Philippe, and A. Sameh. A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 8:155–165, March 1987.

[30] P. Luszczek, H. Ltaief, and J. Dongarra. Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures. *IEEE International Parallel and Distributed Processing Symposium*, May 2011.

[31] O. A. Marques, C. Vömel, J. W. Demmel, and B. N. Parlett. Algorithm 880: A testing infrastructure for symmetric tridiagonal eigensolvers. *ACM Trans. Math. Softw.*, 35:8:1–8:13, July 2008.

[32] R. M. Martin. Electronic structure: Basic theory and practical methods. *Cambridge University Press*, 2008.

[33] B. N. Parlett. *The symmetric eigenvalue problem*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[34] N. Rösch, S. Krüger, V. Nasluzov, and A. Matveev. ParaGauss: The density functional program paragauss for complex systems in chemistry. pages 285–296, 2005. DOI: 10.1007/3-540-28555-5-25.

[35] J. Rutter and J. D. Rutter. A serial implementation of cuppen's divide and conquer algorithm for the symmetric

eigenvalue problem, 1994.
[36] D. C. Sorensen and P. T. P. Tang. On the orthogonality of eigenvectors computed by divide-and-conquer techniques. *SIAM J. Numer. Anal.*, 28(6):1752–1775, 1991.
[37] R. C. Thompson. The behavior of eigenvalues and singular values under perturbations of restricted rank. *Linear Algebra and its Applications*, 13(1-2):69 – 78, 1976.
[38] F. Tisseur and J. Dongarra. Parallelizing the divide and conquer algorithm for the symmetric tridiagonal eigenvalue problem on distributed memory architectures. *SIAM J. SCI. COMPUT*, 20:2223–2236, 1998.
[39] University of Tennessee Knoxville. *PLASMA Users' Guide, Parallel Linear Algebra Software for Multicore Architectures, Version 2.3*, November 2010.
[40] University of Texas Austin. *The FLAME Project*, April 2010.
[41] R. C. Ward and R. P. Muller. *Unpublished manuscript 1999*.
[42] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Claredon Press, Oxford, UK, 1965.