Proceedings of ALGORITMY 2009
pp. 459–466

# INTRODUCTION OF DOUBLE DIVIDE AND CONQUER AND THE RECENT PROGRESS

TARO KONDA AND YOSHIMASA NAKAMURA *

**Abstract.** An algorithm dubbed double Divide and Conquer is recently proposed, which first computes only singluar values by a part of Divide and Conquer and then computes the corresponding singular vectors by the twisted factorization. It accurately computes bidiagonal SVD in $O(n^2)$ *flops* and is suited for parallelization when singular values are isolated. However, the twisted factorization can fail if singular values are tightly clustered. In this paper, we discuss the adaptation of a reorthogonalization process for double Divide and Conquer to cope with such matrices.

**Key words.** Complexity and performance of numerical algorithms, Eigenvalues, singular values, and eigenvectors

**AMS subject classifications.** 65Y20, 15A18

**1. Introduction and examples.** Any given real $n \times m$ rectangular matrix $A, (n < m)$ can be converted into an upper bidiagonal matrix $\hat{B}$ by the Housholder transformation [6, 3]; $A = \hat{U}(\hat{B}\ \ 0)\hat{V}^T$, where $\hat{U}$ and $\hat{V}$ are suitable orthogonal matrices. This paper is concerned with singular value decomposition (SVD) of bidiagonal matrices, particularly an $n \times (n+1)$ bidiagonal matrix

$$B \equiv \begin{pmatrix} b_1 & b_2 & & & \\ & b_3 & b_4 & & \\ & & \ddots & \ddots & \\ & & & b_{2n-1} & b_{2n} \end{pmatrix}, \ {}^{\forall} b_j \neq 0, \tag{1.1}$$

whose SVD is $B = U(\Sigma\ \ 0)V^T$, where $U$ is an $n \times n$ orthogonal matrix whose columns are left singular vectors, $V$ is an $(n+1) \times (n+1)$ orthogonal matrix whose columns are right singular vectors, and $\Sigma$ is an $n \times n$ nonnegative definite diagonal matrix. The diagonal elements of $\Sigma$ are singular values of $B$. The condition for $b_j$ implies that all singular values are nonzero and different from each other with infinite precision.

Recently, we have proposed an algorithm dubbed *double Divide and Conquer* (dDC) for bidiagonal SVD. It first computes *only* singular values by a part of Divide and Conquer, and then computes their corresponding singular vectors by the *twisted factorization* [14, 15]. dDC has shown the competitive speed compared to the standard algorithms such as QR or Divide and Conquer (D&C). The working memory required by dDC is only $O(n)$, in contrast to $O(n^2)$ of D&C. Moreover, dDC has shown the high parallelism compared to other algorithms with the twisted factorization such as I–SVD [11, 20, 10] because dDC adopts a part of D&C to compute singular values.

However, the accuracy of the singular vectors computed by the twisted factorization can worsen if singular values are not relatively isolated [5]. To overcome this problem, we apply a reorthogonalization process to dDC. In Sections 2.1 and 2.2, we first introduce the details of dDC. Then the reorthogonalization process for dDC is discussed in Section 2.3. Finally, we show some numerical results in Section 3.

*Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan (konda@amp.i.kyoto-u.ac.jp), and SORST, Japan Science and Technology Agency.

**2. Proposed algorithm for singular value decomposition.** The twisted factorization is a fast and accurate algorithm to compute a singular vector for a given singular value [11, 20, 10]. It computes sufficiently orthogonal singular vectors when the given singular values are relatively accurate and are isolated from each other. Each singular vector is independently computed. This feature is an advantage for parallelization. However, the total parallelism of the algorithms which adopts the twisted factorization such as I–SVD is limited due to seriality in the task of singular value computation [14]. In contrast, dDC successfully improves its total parallelism using a part of Divide and Conquer for singluar value computation. It then computes the corresponding singular vectors by the twisted factorization.

**2.1. Computing singular values.** D&C was originally proposed for the solution of symmetric tridiagonal eigenvalue problems [2, 7, 9]. First, it partitions the targeted matrix into two submatrices. Then, the eigenvalues and eigenvectors of the targeted matrix are synthesized from the solutions of the submatrices. These processes are recursively applied until the size of each submatrix becomes sufficiently small. As shown below, we demonstrate that D&C can similarly be applied to bidiagonal SVD [12, 8]. In this paper, we omit details of an acceleration technique called *deflation* because it will not have considerable effect on the performance of our algorithm.

The $n \times (n+1)$ bidiagonal matrix $B$ is partitioned into two submatrices as

$$B = \begin{pmatrix} B_1 & 0 \\ b_{2k-1}\mathbf{e}_k{}^T & b_{2k}\mathbf{e}_1{}^T \\ 0 & B_2 \end{pmatrix} \tag{2.1}$$

such that $1 < k < n$ for a fixed $k$. Here $B_1$ is a $(k-1) \times k$ bidiagonal matrix, $B_2$ is an $(n-k) \times (n-k+1)$ bidiagonal matrix and $\mathbf{e}_j$ is the vector whose $j$–th element is 1 and the others are 0. The parameter $k$ is usually taken as $\lfloor n/2 \rfloor$. Now, given that the SVD of $B_i$ is

$$B_i = U_i \, (D_i \; 0) \, (V_i \; \mathbf{v}_i)^T \quad \text{for } i = 1, 2, \tag{2.2}$$

where $U_i$ and $(V_i \; \mathbf{v}_i)$ are suitable orthogonal matrices and $D_i$ are positive diagonal matrices, and assuming that for each $D_i$, all singular values are nonzero and different from each other, then, by substituting (2.2) into (2.1) and applying Givens rotation, we obtain

$$B = \tilde{U} \, (M \; 0) \left( \tilde{V} \; \tilde{\mathbf{v}} \right)^T. \text{ Here} \tag{2.3}$$

$$\tilde{U} \equiv \begin{pmatrix} 0 & U_1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & U_2 \end{pmatrix}, M \equiv \begin{pmatrix} r_0 & b_{2k-1}\mathbf{l}_1 & b_{2k}\mathbf{f}_2 \\ 0 & D_1 & 0 \\ 0 & 0 & D_2 \end{pmatrix}, \tilde{V} \equiv \begin{pmatrix} c_0\mathbf{v}_1 & V_1 & 0 \\ s_0\mathbf{v}_2 & 0 & V_2 \end{pmatrix},$$

$$\tilde{\mathbf{v}} \equiv \begin{pmatrix} -s_0\mathbf{v}_1 \\ c_0\mathbf{v}_2 \end{pmatrix}, r_0 = \sqrt{(b_{2k-1}\psi_1)^2 + (b_{2k}\phi_2)^2}, \quad c_0 = \frac{b_{2k-1}\psi_1}{r_0} \quad \text{and} \quad s_0 = \frac{b_{2k}\phi_2}{r_0}.$$

And $\mathbf{l}_1$ is the last row of $V_1$, $\psi_1$ is the last element of $\mathbf{v}_1$, $\mathbf{f}_2$ is the first row of $V_2$ and $\phi_2$ is the first element of $\mathbf{v}_2$. Hence, if we let the SVD of $M$ be $M = U_M \Sigma V_M^T$, then

$$B = \tilde{U} \left( U_M \Sigma V_M^T \; 0 \right) \left( \tilde{V} \; \tilde{\mathbf{v}} \right)^T = U \, (\Sigma \; 0) \, V^T, \text{ where } U = \tilde{U}U_M, V = \left( \tilde{V}V_M \; \tilde{\mathbf{v}} \right). \tag{2.4}$$

Time complexity of D&C ranges from $O(n^2)$ to $O(n^3)$ *flops* depending on the number of deflations since the dimension of matrix–matrix multiplication in (2.4) is reduced when the deflation is applied [3, 8]. In addition, D&C requires $O(n^2)$ extra working memory to hold all singular vectors of submatrices.

D&C process discussed above can be simplified, particularly when only singular values are desired [8]. Using (2.4), $B$ can be rewritten as

$$B = U\Sigma \left(\tilde{V}V_M \ \tilde{\mathbf{v}}\right)^T = U\Sigma \left( \begin{pmatrix} c_0\mathbf{v}_1 & V_1 & 0 \\ s_0\mathbf{v}_2 & 0 & V_2 \end{pmatrix} V_M \quad \begin{pmatrix} -s_0\mathbf{v}_1 \\ c_0\mathbf{v}_2 \end{pmatrix} \right)^T, \text{ where}$$

$$\mathbf{f} = (c_o\phi_1 \ \mathbf{f}_1 \ 0)\, V_M, \quad \mathbf{l} = (s_o\psi_2 \ 0 \ \mathbf{l}_2)\, V_M, \quad \phi = -s_0\phi_1 \text{ and } \psi = c_0\psi_2. \tag{2.5}$$

Here $\mathbf{f}_1$ and $\mathbf{f}$ are the first rows of $V_1$ and $V$, $\phi_1$ and $\phi$ are the first elements of $\mathbf{v}_1$ and $\mathbf{v}$, $\mathbf{l}_2$ and $\mathbf{l}$ are the last rows of $V_2$ and $V$, and $\psi_2$ and $\psi$ are the last elements of $\mathbf{v}_2$ and $\mathbf{v}$, respectively. Equation (2.5) shows that only $\mathbf{f}_1$, $\mathbf{l}_2$, $\phi_1$ and $\psi_2$ are needed to compute $\mathbf{f}$, $\mathbf{l}$, $\phi$ and $\psi$. In other words, there is no need to compute and hold all singular vectors of the submatrices.

Now we focus on computing $V_M$. Let us take $M$ as

$$M = \begin{pmatrix} z_1 & z_2 & \cdots & z_n \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix}. \tag{2.6}$$

Then the SVD of $M$ can be computed according to the following theorem.

THEOREM 2.1. [8] *Let $U\Sigma V^T$ be the SVD of $M$ with $U = (\mathbf{u}_1, ..., \mathbf{u}_n)$, $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_n)$ and $V = (\mathbf{v}_1, ..., \mathbf{v}_n)$. Then the singular values $\{\sigma_i\}_{i=1}^n$ satisfy the interlacing property*

$$0 \equiv d_1 < \sigma_1 < d_2 < \cdots < d_n < \sigma_n < d_n + ||z||_2 \tag{2.7}$$

*and the secular equation*

$$f(\sigma_i) \equiv 1 + \sum_{j=1}^{n} \frac{z_j^2}{d_j^2 - \sigma_i^2} = 0 \quad (i = 1, 2, ..., n). \tag{2.8}$$

*Let $\{\hat{\sigma}_i\}_{i=1}^n$ be the singular values computed by solving (2.8), then the singular vectors are given by*

$$\mathbf{u}_i = \frac{\left(-1, \frac{d_2\hat{z}_2}{d_2^2 - \hat{\sigma}_i^2}, \cdots, \frac{d_n\hat{z}_n}{d_n^2 - \hat{\sigma}_i^2}\right)^T}{\sqrt{1 + \sum_{j=2}^{n} \frac{(d_j\hat{z}_j)^2}{(d_j^2 - \hat{\sigma}_i^2)^2}}} \text{ and } \mathbf{v}_i = \frac{\left(\frac{\hat{z}_1}{d_1^2 - \hat{\sigma}_i^2}, \cdots, \frac{\hat{z}_n}{d_n^2 - \hat{\sigma}_i^2}\right)^T}{\sqrt{\sum_{j=1}^{n} \frac{\hat{z}_j^2}{(d_j^2 - \hat{\sigma}_i^2)^2}}}, \tag{2.9}$$

$$\textit{where } |\hat{z}_i| = \sqrt{(\hat{\sigma}_n^2 - d_i^2) \prod_{j=1}^{i-1} \frac{(\hat{\sigma}_j^2 - d_i^2)}{(d_j^2 - d_i^2)} \prod_{j=i}^{n-1} \frac{(\hat{\sigma}_j^2 - d_i^2)}{(d_{j+1}^2 - d_i^2)}}$$

*and the sign of $\hat{z}_i$ can be chosen arbitrarily.*

In this paper, we refer to this compact version as *Divide and Conquer for singular values (DCSV)*. The secular equation can be solved by using the interpolating rational functions [16]. Since the right singular vectors can be computed one by one as shown in (2.9), $\mathbf{f}$ and $\mathbf{l}$ in (2.5) can be computed element by element. As a result, we don't need to hold the whole $V_M$ and only $O(n)$ working memory is required.

**2.2. Computing singular vectors for isolated singular values.** Once a singular value is computed, we can apply the twisted factorization to obtain the corresponding singular vector. Let us now consider the following system

$$(B^TB - \hat{\sigma}^2 I)\mathbf{x}^{(k)} = \mathbf{e}_k\gamma_k, \tag{2.10}$$

where $\mathbf{x}^{(k)}$ is the desired vector, and $\gamma_k$ is set such that the $k$–th element of $\mathbf{x}^{(k)}$ is 1. We take $P$ to be a lower bidiagonal matrix and $Q$ to be an upper bidiagonal matrix such that $B^TB - \hat{\sigma}^2 I = PD^+P^T = QD^-Q^T$, where $P$ and $Q$ are given by Choleskey decomposition of $B^TB - \hat{\sigma}^2 I$. Let the $i$–th diagonal elements of $D^+$ and $D^-$ be $D_i^+$ and $D_i^-$, and $i$–th subdiagonal elements of $P$ and $Q$ be $P_i$ and $Q_i$, respectively. It follows that the factorizations $PD^+P^T$ and $QD^-Q^T$ are computable by *stdLVvs* and *rdLVvs* algorithms, respectively. See [21, 11, 17] for details.

Then the twisted factorization can be described as follows.

$$B^TB - \hat{\sigma}^2 I = N_kD_kN_k^T, \tag{2.11}$$

where $N_k$ is defined as

$$N_k = \begin{pmatrix} 1 & & & & & & \\ p_1 & \ddots & & & & & \\ & \ddots & 1 & & & & \\ & & p_{k-1} & 1 & q_k & & \\ & & & & 1 & \ddots & \\ & & & & & \ddots & q_{n-1} \\ & & & & & & 1 \end{pmatrix} \tag{2.12}$$

and $D_k$ is the diagonal matrix such that $D_k = \mathrm{diag}\left(D_1^+, ..., D_{k-1}^+, \gamma_k, D_{k+1}^-, ..., D_n^-\right)$. We choose the twist point $k$ so as to minimize $|\gamma_k|$ according to the following theorem.

THEOREM 2.2. [18] *$\gamma_k$ is given by $\gamma_k = D_k^+ + D_k^- - \left(g_{k-1}^{(0)} + h_k^{(0)} - \hat{\sigma}^2\right)$, where k=1,2,...,n, $h_k^{(0)} = (b_{2k-1})^2$ and $g_k^{(0)} = (b_{2k})^2$.*

By substituting (2.11) into (2.10), we obtain $N_kD_kN_k^T\mathbf{x}^{(k)} = \mathbf{e}_k\gamma_k$. Since $D_k\mathbf{e}_k = \mathbf{e}_k\gamma_k$ and $N_k\mathbf{e}_k = \mathbf{e}_k$, it is transformed into

$$N_k^T\mathbf{x}^{(k)} = \mathbf{e}_k. \tag{2.13}$$

Finally, we solve (2.13) by backward substitution in $O(n)$ *flops* time. Both *stdLVvs* and *rdLVvs* cost $O(n)$ *flops* [17], thus we compute all singular vectors in $O(n^2)$ *flops* by the twisted factorization. Both of them require only $O(n)$ working memory to hold the elements of $P$ and $Q$.

**2.3. Computing singular vectors for clustered singular values.** It is known that the orthogonality of singular vectors computed by the twisted factorization is guaranteed only if the corresponding singular values are high relatively isolated [5]. Consequently, the closer the singular values are, the worse the accuracy of the singular vectors computed by the twisted factorization can be. Therefore, to improve the accuracy of all singular vectors, we add a single step of the inverse iteration [3] after the twisted factorization.

However, the twisted factorization with the inverse iteration will fail when singular values are indistinguishable with finite precision. This can happen even when singular

values are computed with relative accuracy. Hence, to compute appropriate singular vectors for such cases, we need a reorthogonalization process as a backup. If a cluster of singular values is found, we can alternatively apply to each cluster the inverse iteration with the modified Gram–Schmidt [19] as shown below.

Let us consider a matrix $\check{B}$ such that

$$\check{B} = \begin{pmatrix} & B \\ B^T & \end{pmatrix}. \tag{2.14}$$

Here let the element $b_{2n}$ of $B$ be 0 and we henceforth regard $B$ as the $n \times n$ bidiagonal matrix for simplicity. Then, the eigenvalues of this matrix are $\lambda = \pm\sigma$, and it can be equivalently transformed into the following tridiagonal matrix $T_{GK}$ called the Golub–Kahan matrix [4].

$$T_{GK} = \begin{pmatrix} 0 & b_1 & & & & \\ b_1 & 0 & b_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & b_{2n-2} & 0 & b_{2n-1} & \\ & & & b_{2n-1} & 0 & \end{pmatrix}. \tag{2.15}$$

Let $\mathbf{z}_j$ be an eigenvector of $T_{GK}$, then $T_{GK}\mathbf{z}_j = \sigma_j \mathbf{z}_j$, where $\mathbf{z}_j$ consists of $\mathbf{u}_j$ and $\mathbf{v}_j$ such that $\mathbf{z}_j = (v_j(1), u_j(1), v_j(2), u_j(2), \cdots, v_j(n), u_j(n))^T$. Here $v_j(i)$ and $u_j(i)$ are the $i$–th elements of $\mathbf{v}_j$ and $\mathbf{u}_j$, respectively. If we explicitly construct $B^T B$, then the condition number can worsen. To circumvent this problem, we compute the singular vectors of $B$ without a change in the condition number by applying the inverse iteration with the modified Gram–Schmidt orthogonalization to $T_{GK}$.

Now we summarize the algorithm of dDC. It first computes singular values by the *compact* D&C, and then computes the corresponding singular vectors by the twisted factorization with a single step of the inverse iteration. For singular vectors corresponding to clustered singular values, the Golub–Kahan matrix is solved by the inverse iteration with the modified Gram–Schmidt for each cluster. We name our algorithm *double Divide and Conquer (dDC)* because it accomplishes two functions; (i) it computes singular values by **Divide and Conquer** and (ii) it computes singular values and singular vectors **separately**.

Since in D&C, matrix–matrix multiplication in (2.4) is the unique part to take $O(n^3)$ *flops* and the other parts cost only $O(n^2)$ *flops*, thus the time complexity of DCSV is $O(n^2)$ *flops* to compute all singular values. Moreover, both the twisted factorization to compute one singular vector and the single step of the inverse iteration for one singular vector cost $O(n)$ *flops* [3]. Therefore, the total complexity of dDC is $O(n^2)$ *flops* when all singular values are isolated from each other. If singular values are duplicated or tightly clustered, then reorthogonalization costs $O(nk^2)$ *flops*, where $k$ is the size of the largest cluster [3]. Thus in such cases, dDC takes $O(n^2 + nk^2)$ *flops*. As we have discussed already, the working memory required by DCSV is reduced to $O(n)$ when vectors $\mathbf{f}$ and $\mathbf{l}$ are computed element by element. Moreover, the twisted factorization also requires $O(n)$, and in practice, we can use $\mathbf{v}_i$ and $\mathbf{u}_i$ to hold vector $\mathbf{z}_i$ during the modified Gram–Schmidt. Thus dDC requires $O(n)$ working memory whereas D&C requires $O(n^2)$ to hold the matrices $U$ and $V$.

Furthermore, the twisted factorization can selectively compute the required vectors because each singular vector is computed independently of each other. Similarly, DCSV is capable of selecting the secular equation to be solved. In contrast, I–SVD

TABLE 3.1
*The shortest run time out of* 10 *executions to compute SVD for each bidiagonal matrix type and dimension.*

| Algorithm | Matrix type | Run time for $n \times n$ matrix (Seconds) $n = 1,000$ | $n = 2,000$ | $n = 3,000$ |
|---|---|---|---|---|
| dDC | Isolated | 0.53 | 2.12 | 4.84 |
| I–SVD | Isolated | 0.51 | 2.09 | 4.79 |
| D&C | Isolated | 4.98 | 40.66 | 135.54 |
| QR | Isolated | 42.34 | 382.21 | 1356.15 |
| dDC | Clustered | 1.26 | 10.37 | 38.79 |
| D&C | Clustered | 0.15 | 0.68 | 1.80 |
| QR | Clustered | 31.29 | 274.97 | 900.15 |

has to compute all singular values in order to find the targeted ones. Let us note that the computational cost of DCSV is not proportional to the number of the desired singular values because all subproblems other than the root must be fully solved.

**3. Numerical experiments.** In this section, we evaluate the speed and accuracy of dDC compared to the following three algorithms. We use LAPACK [1] with self–compiled BLAS (Basic Linear Algebra Subprograms).

1. I–SVD: I–SVD algorithm whose mdLVs's parameter $\delta^{(t)}$ is set as 1.0. A single step of the inverse iteration updates singular vectors computed by the twisted factorization.
2. D&C: Normal Divide and Conquer algorithm (`DBDSDC` in LAPACK).
3. QR: QR algorithm with shifts (`DBDSQR` in LAPACK).

We set *thresh* of dDC as $thresh \equiv 1.0 \times 10^{-9}$ in all experiments. dDC solves the secular equation using `DLASD4` in LAPACK. We use a Linux (Fedora Core 3) PC with Opteron 2.4GHz (Cache L1D: 64KB, L1I: 64KB and L2: 1024KB) and 2GB Memory. We have chosen the following two bidiagonal matrices.

1. **Isolated:** All diagonal elements are 2.001 and all subdiagonal elements are 2.0. All singular values are well separated from each other.
2. **Clustered:** Glued Kimura Matrix. The 16th and 17th singular values of the $17 \times 17$ bidiagonal matrix $K$ whose diagonal elements are $(9, ..., 1, ..., 9)$ and subdiagonal elements are all unity are duplicated with double precision like the Wilkinson matrix for symmetric tridiagonal eigenvalue problem [13]. The following "glued" version of $K$ with a small number $\delta$ should contain some strong clusters.

$$\begin{pmatrix} K & \delta & & \\ & K & \ddots & \\ & & \ddots & \delta \\ & & & K \end{pmatrix}. \tag{3.1}$$

The tail $K$ is to be truncated to appropriate dimension and $\delta$ is set as $1.0 \times 10^{-10}$.

**3.1. Speed.** The Table 3.1 shows the shortest run time out of 10 executions to compute SVD by each algorithm. The size of the matrices are $n = 1,000$, $n = 2,000$ and $n = 3,000$. For Isolated matrices, D&C shows around $O(n^3)$ run time due to a small number of deflations in D&C although D&C is faster than QR all each cases. dDC is as fast as I–SVD and they are much faster than the others.

TABLE 3.2

*Evaluations of* $||V^TV - I||_{\text{abs}}$ *for orthogonality of right singular vectors,* $||U^TU - I||_{\text{abs}}$ *for orthogonality of left singular vectors and* $||(B - U\Sigma V^T)||_{\text{abs}}$ *for accuracy of SVD for* $n = 1,000$ *bidiagonal matrices of each type.*

| Algorithm | Matrix type | $\|\|V^TV - I\|\|_{\text{abs}}$ | $\|\|U^TU - I\|\|_{\text{abs}}$ | $\|\|(B - U\Sigma V^T)\|\|_{\text{abs}}$ |
|---|---|---|---|---|
| dDC | Isolated | 3.7e-10 | 3.6e-10 | 4.2e-09 |
| I–SVD | Isolated | 3.7e-10 | 3.5e-10 | 3.8e-09 |
| D&C | Isolated | 5.1e-11 | 5.1e-11 | 3.0e-11 |
| QR | Isolated | 1.3e-10 | 1.3e-10 | 1.1e-10 |
| dDC | Clustered | 6.1e-10 | 6.1e-10 | 1.1e-08 |
| D&C | Clustered | 1.3e-12 | 1.4e-12 | 1.7e-11 |
| QR | Clustered | 3.2e-11 | 3.2e-11 | 3.1e-10 |

For Clustered matrices, dDC slows down compared to the results in Isolated matrices because 17 clusters are found in each Clustered matrix and the reorthogonalization is required. In contrast, a large number of deflations occurs in D&C and it becomes the fastest algorithm. Again, they are much faster than QR in this case.

**3.2. Accuracy.** Here, we test the accuracy of dDC compared to the other algorithms. To evaluate error over all elements, we use the norm $||A||_{\text{abs}} = \Sigma_{i,j}|a_{ij}|$. The criteria are $||V^TV - I||_{\text{abs}}$ and $||U^TU - I||_{\text{abs}}$ for orthogonality of right and left singular vectors, respectively, and $||(B - U\Sigma V^T)||_{\text{abs}}$ for accuracy of SVD. Table 3.2 shows the results for $n = 1,000$ bidiagonal matrices. For Isolated matrices, we observe that D&C shows the highest accuracy and QR is the second for all criteria. dDC performs as good as I–SVD.

For Clustered matrices, the same tendency of results for Isolated matrices is observed. dDC acquires singular vectors having a proper orthogonality by using the reorthogonalization.

**Conclusion.** For bidiagonal SVD, double Divide and Conquer (dDC) is $O(n^2)$ *flops* algorithm for matrices whose singular values are isolated. When singular values are tightly clustered, dDC alternatively uses the inverse iteration with the modified Gram–Schmidt for each cluster. In such case, time complexity becomes $O(n^2 + nk^2)$ *flops*, where $k$ is the size of the largest cluster. For any type of matrices, dDC needs only $O(n)$ working memory, in contrast, D&C requires $O(n^2)$. A hybrid implementation of dDC and D&C can be effective because their speed is complementary to each other depending on the existence of strong clusters.

REFERENCES

[1] E. Anderson, Z. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide Second Edition.* SIAM, Philadelphia, 1995.

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

[3] J. Demmel. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[4] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):873–912, 1990.

[5] I. Dhillon and B. Parlett. Orthogonal eigenvectors and relative gaps. *SIAM J. Matrix Anal. Appl.*, 25(3):858–899, 2004.

[6] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numeri. Anal.*, 2(2):205–224, 1965.

[7] M. Gu and S.C. Eisenstat. A stable and efficient algorithm for the rank–1 modification of the symmetric eigenproblem. *SIAM J. Matrix Anal. Appl.*, 15(4):1266–1276, 1994.

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

[9] 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.

[10] M. Iwasaki and Y. Nakamura. Accurate computation of singular values in terms of the shifted integrable schemes. *Japan J. Indust. Appl. Math.*, 23:239–259, 2006.

[11] M. Iwasaki, S. Sakano, and Y. Nakamura. Accurate twisted factorization of real symmetric tridiagonal matrices and its application to singular value decomposition. *Trans. Japan Soc. Indust. Appl. Math.*, 15(3):461–481, 2005.

[12] 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.

[13] M. Katayama, K. Kimura, M. Takata, H. Tsuboi, M. Iwasaki, and Y. Nakamura. Improvement of computation of left singular vectors by the I–SVD algorithm for singular value decomposition. *Trans. Japan Soc. Indust. Appl. Math.*, 18(3):389–407, 2008.

[14] T. Konda, M. Takata, M. Iwasaki, and Y. Nakamura. A new singular value decomposition algorithm suited to parallelization and preliminary results. In *Proceedings of the IASTED International Conference on Advances in Computer Science and Technology*, pages 79–84, 2006.

[15] T. Konda, H. Tsuboi, M. Takata, M. Iwasaki, and Y. Nakamura. Parallelism of double divide and conquer algorithm for singular value decomposition. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pages 393–398, 2007.

[16] R. Li. Solving secular equations stably and efficiently. Technical Report UCB//CSD–94–851, Computer Science Division, Berkeley, Univ. of California, 1993.

[17] Y. Nakamura, M. Iwasaki, K. Kimura, and M. Takata. Recent developments of the mdLVs algorithm for singular values and the I-SVD algorithm for singular value decomposition. *Fast Algorithms in Computational Fluids: Theory and Applications, RIMS Kyokuroku, Kyoto Univ.*, 1606:87–104, 2008.

[18] B. Parlett and I. Dhillon. Fernando's solution to wilkinson's problem: An application of double factorization. *Lin. Alg. Appl.*, 267:247–279, 1997.

[19] G. Peters and J. H. Wilkinson. *The calculation of specified eigenvectors by inverse iteration, contribution II/18, Handbook for Automatic Computation*, volume 2. Springer Verlag, Berlin, 1971.

[20] M. Takata, M. Iwasaki, K. Kimura, and Y. Nakamura. An evaluation of singular value computation by the discrete Lotka–Volterra system. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 2, pages 410–416, 2005.

[21] M. Takata, T. Konda, K. Kimura, and Y. Nakamura. Verification of dlvv transformation for singular vector computation with high accuracy. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 881–887, 2006.