Implementation and performance evaluation of new inverse iteration algorithm with Householder transformation in terms of the compact WY representation

Hiroyuki ISHIGAMI[1], **Kinji KIMURA**[1], **and Yoshimasa NAKAMURA**[1]
[1]Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

Abstract—*A new inverse iteration algorithm that can be used to compute all the eigenvectors of a real symmetric tridiagonal matrix on parallel computers is developed. In the classical inverse iteration algorithm, the modified Gram-Schmidt orthogonalization is used, and this causes a bottleneck in parallel computing. In this paper, the use of the compact WY representation is proposed in the orthogonalization process of the inverse iteration algorithm with the Householder transformation. This change results in drastically reduced synchronization cost in parallel computing. The new algorithm is evaluated on a 32-core parallel computer, and it is shown that the algorithm is up to 7.46 times faster than the classical algorithm in computing all the eigenvectors of matrices with several thousand dimensions.*

Keywords: compact WY representation, Householder transformation, inverse iteration, eigenvalue decomposition

1. Introduction

The eigenvalue decomposition of a symmetric matrix, i.e., a decomposition into a product of matrices consisting of eigenvectors and eigenvalues, is one of the most important operations in linear algebra. It is used in vibrational analysis, image processing, data searches, etc.

Owing to recent improvements in the performance of computers equipped with multicore processors, we have had more opportunities to perform calculations on parallel computers. As a result, there has been an increase in the demand for an eigenvalue decomposition algorithm that can be effectively parallelized.

Such an eigenvalue decomposition algorithm involves a process of transforming a real symmetric matrix into a real symmetric tridiagonal matrix as a preconditioning step. Therefore the problem of eigenvalue decomposition can be reduced to that of a symmetric tridiagonal matrix. Several eigenvalue decomposition algorithms for a real symmetric tridiagonal matrix have been proposed. They are classified into two types. The first type of algorithm computes simultaneously all the eigenvalues and the eigenvectors. Algorithms of this type includes the QR algorithm [5] and the divide-and-conquer algorithm [8]. The second type of algorithm computes all or some eigenvalues and all or some eigenvectors. Algorithms for computing eigenvalues includes the root-free QR algorithm [7] and the bisection algorithm [5]. Algorithms for computing eigenvectors includes the MR^3 algorithm [3] and the inverse iteration algorithm [10]. LAPACK (Linear Algebra PACKage) [9], a software library for numerical linear algebra, has codes that integrate all the above-mentioned algorithms.

The inverse iteration algorithm is an algorithm for computing eigenvectors independently associated with mutually distinct eigenvalues. However, when we use the inverse iteration algorithm, we must reorthogonalize the eigenvectors if some eigenvalues are very close to each other. Adding this reorthogonalization algorithm increases the computational cost. Moreover, for this reorthogonalization, we have generally used the MGS (modified Gram-Schmidt) algorithm. However, this algorithm is sequential and inefficient for parallel computing. As a result, we are unable to maximize the performance of parallel computers. Hereinafter, we will refer to the inverse iteration algorithm with the MGS algorithm as the classical inverse iteration algorithm.

We can also orthogonalize vectors by using the Householder transformation [12], and we call this orthogonalization precess the Householder orthogonalization algorithm. While the MGS algorithm is unstable in the sense that the orthogonality of the resulting vectors depends on the condition number of the symmetric tridiagonal matrix [13], the Householder algorithm is stable because its orthogonality does not depend on the condition number. The Householder algorithm is also sequential and ineffective for parallel computing, and its computational cost are higher than those of the MGS algorithm.

In 1989, the Householder orthogonalization algorithm in terms of the compact WY representation was proposed in [11]. By adopting this Householder orthogonalization, stability and effective parallelization can be achieved. Hereafter, we refer to this algorithm as the compact WY orthogonalization. In 2010, Yamamoto demonstrated the fact [13]: When this algorithm is used in the Arnoldi process, the computation time for parallel computation is less than that when the MGS algorithm is used, and the orthogonality of the eigenvectors generated using this algorithm is better than that of the eigenvectors generated using the MGS algorithm.

In this paper, we consider an implementation of the compact WY orthogonalization to the inverse iteration al-

gorithm and we evaluate its performance. Thereafter, we present a new inverse iteration algorithm for computing the eigenvectors of a real symmetric tridiagonal matrix.

The contents of this paper are as follows. In Sec.2, we explain the classical inverse iteration algorithm and describe its defect. In Sec.3, we explain the Householder orthogonalization and the compact WY orthogonalization. In Sec.4, we present a new inverse iteration algorithm, namely, the compact WY inverse iteration algorithm, whose orthogonalization process is performed by compact WY orthogonalization instead of MGS, and we explain its properties. In Sec.5, we discuss numerical experiments on parallel computers and their results. In the experiments, we compute eigenvectors of symmetric tridiagonal matrices with several thousand dimensions by using the classical algorithm and the new algorithm. It is shown that the new algorithm is up to 7.46 times faster than the classical algorithm. Section 6 presents our conclusions.

2. Classical inverse iteration algorithm and its defect

2.1 Classical inverse iteration

We consider the problem of computing eigenvectors of a real symmetric tridiagonal matrix $T \in \mathbb{R}^{n\times n}$. Let $\lambda_j \in \mathbb{R}$ be eigenvalues of T such that $\lambda_1 < \lambda_2 < \cdots < \lambda_n$. Let $\boldsymbol{v}_j \in \mathbb{R}^n$ be the eigenvector associated with λ_j. When $\tilde{\lambda_j}$, an approximate value of λ_j, and a starting vector $\boldsymbol{v}_j^{(0)}$ are given, we can compute an eigenvectors of T. To this end, we solve the following equation iteratively:

$$\left(T - \tilde{\lambda}_j I\right) \boldsymbol{v}_j^{(k)} = \boldsymbol{v}_j^{(k-1)}. \tag{1}$$

Here I is the n-dimensional identity matrix. If the eigenvalues of T are mutually well-separated, the solution of $\boldsymbol{v}_j^{(k)}$, Eq.(1) generically converges to the eigenvector associated with λ_j as k goes to ∞. The above iteration method is the inverse iteration method. The computational cost of this method is of order mn when we compute m eigenvectors, and it is less than that of other methods for eigenvalue decomposition. In the implementation, we have to normalize the vectors $\boldsymbol{v}_j^{(k)}$ to avoid overflow.

When some of all the eigenvalues are close together or there are clusters of eigenvalues, we have to reorthogonalize all the eigenvectors associated with such eigenvalues because they need to be orthogonal to each other. If we apply the MGS orthogonalization, the computational cost is of order m^2n. Therefore, when we calculate eigenvectors of the matrix T that has many clustered eigenvalues, the total computational cost increases significantly. In general, when we implement the inverse iteration method on computers, we use the MGS orthogonalization with the Peters-Wilkinson method [10] as the standard orthogonalization process. The MGS with the Peters-Wilkinson method is also available on DSTEIN, the LAPACK code of the inverse iteration algorithm for computing eigenvectors of a real symmetric tridiagonal matrix. In the Peters-Wilkinson method, when the distance between the close eigenvalues is less than $10^{-3}\|T\|$, we regard them as members of the same cluster of eigenvalues, and we orthogonalize all of the eigenvectors associated with these eigenvalues.

Figure 1 shows the inverse iteration algorithm based on the MGS with the Peters-Wilkinson method outlined above. We call this the classical inverse iteration method.

```
1: for j = 1 to n do
2:     Generate v_j^(0) from random numbers.
3:     k = 0
4:     repeat
5:         k ← k + 1.
6:         Normalize v_j^(k−1).
7:         (1) : Compute v_j^(k) by using v_j^(k−1).
8:         if |λ̃_j − λ̃_{j−1}| ≤ 10^−3 ||T||, then
9:             for i = j_1 to j − 1 do
10:                v_j^(k) ← v_j^(k) − ⟨v_j^(k), v_i⟩ v_i
11:            end for
12:        else
13:            j_1 = j
14:        end if
15:    until some condition is met.
16:    Normalize v_j^(k) to v_j.
17: end for
```

Fig. 1: Classical inverse iteration algorithm. j_1 means the index j of the first eigenvalue of cluster.

2.2 The defect of the classical inverse iteration algorithm

The inverse iteration is a prominent method for computing eigenvectors, because we can compute eigenvectors independently and this process is easily parallelized. When we use the classical inverse iteration on parallel computers, we can parallelize it even if some clusters exist.

Let us consider the Peters-Wilkinson method in the classical inverse iteration. When the dimension of T is greater than 1000, most of the eigenvalues are regarded as being in the same cluster [3].

In this case, we have to parallelize the inverse iteration with respect to not the cluster but the loop described from lines 1 to 17 in Figure 1. This loop includes the iteration based on Eq.(1) and the orthogonalization of the eigenvectors. This orthogonalization process becomes a bottleneck of the classical inverse iteration with respect to the computational time. The MGS algorithm is mainly based on a BLAS level-1 operation such as the inner product operation and the AXPY operation, and it is a sequential algorithm. Because of this, when we compute all the eigenvectors in parallel computers, the number of synchronizations is of order m^2. Therefore, the MGS algorithm is ineffective on parallel computing.

In conclusion, the classical inverse iteration is an ineffective algorithm for parallel computing because the MGS algorithm is used in its orthogonalization process

3. Other orthogonalization algorithms

3.1 Householder orthogonalization

The Householder orthogonalization, based on the Householder matrices, is one of the alternative orthogonalization methods. When some vectors $\boldsymbol{v}$, $\boldsymbol{d} \in \mathbb{R}^n$ satisfy $\|\boldsymbol{v}\|_2 = \|\boldsymbol{d}\|_2$, there exists the symmetric matrix H satisfying $HH^\top = H^\top H = I$, $H\boldsymbol{v} = \boldsymbol{d}$ defined by

$$H = I - t\boldsymbol{y}\boldsymbol{y}^\top, \ \ \boldsymbol{y} = \boldsymbol{v} - \boldsymbol{d}, \ \ t = \frac{2}{\|\boldsymbol{y}\|_2^2}. \tag{2}$$

The transformation by H is called the Householder transformation. We can orthogonalize some vectors by using the Householder transformations. The algorithm of the Householder transformations is shown in Figure 2. The vector $\boldsymbol{y}_j$ is the vector in which the elements from 1 to $(j-1)$ are the same as the elements of $\boldsymbol{v}'_j$ and the elements from $(j+1)$ to n are zero. The vector $\boldsymbol{e}_j$ is the jth vector of an n-dimensional identity matrix. In this paper, we call this algorithm the Householder orthogonalization.

1: **for** $j = 1$ to m **do**
2: Generate $\boldsymbol{v}_j$ from $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_{j-1}$.
3: $\boldsymbol{v}'_j = \left(I - t_{j-1}\boldsymbol{y}_{j-1}\boldsymbol{y}_{j-1}^\top\right) \cdots \left(I - t_2\boldsymbol{y}_2\boldsymbol{y}_2^\top\right)\left(I - t_1\boldsymbol{y}_1\boldsymbol{y}_1^\top\right)\boldsymbol{v}_j$.
4: Compute $\boldsymbol{y}_j$ and t_j by using $\boldsymbol{v}'_j$.
5: $\boldsymbol{q}_j = \left(I - t_1\boldsymbol{y}_1\boldsymbol{y}_1^\top\right)\left(I - t_2\boldsymbol{y}_2\boldsymbol{y}_2^\top\right) \cdots \left(I - t_j\boldsymbol{y}_j\boldsymbol{y}_j^\top\right)\boldsymbol{e}_j$.
6: **end for**

Fig. 2: Householder orthogonalization algorithm.

The orthogonality of the vectors $\boldsymbol{q}_j$ generated by the Householder orthogonalization does not depend on the condition number of the matrix T. Therefore, the Householder orthogonalization is more stable than the MGS. On the other hand, being similar to the MGS, it is a sequential algorithm that is mainly based on a BLAS level-1 operation. Its computational cost is higher than that of the MGS. Thus the Householder orthogonalization algorithm is an ineffective algorithm in parallel computing.

3.2 Compact WY orthogonalization

In 2010, Yamamoto presented the Householder orthogonalization in the Arnoldi process in terms of the compact WY representation [13]. This study suggests that the Householder orthogonalization becomes capable of computation with a BLAS level 2 operation in terms of the compact WY representation [11]. Yamamoto also showed that the computation time for orthogonalization on parallel computers has decreased with the use of the Householder orthogonalization algorithm in terms of the compact WY representation, compared to this computational time in the case of the MGS algorithm [13]. Although Yamamoto mainly shows the new representation of orthogonalization in [13], in this paper, we show the implementation of this orthogonalization to the inverse iteration algorithm, and we evaluate its performance.

Now, we consider the Householder orthogonalization in Figure 2 and we introduce the compact WY representation. First, we define $Y_1 = \boldsymbol{y}_1 \in \mathbb{R}^{n\times j}$ and $T_1 = t_1 \in \mathbb{R}^{1\times 1}$. Next, we define matrices $Y_j \in \mathbb{R}^{n\times j}$ and upper triangular matrices $T_j \in \mathbb{R}^{j\times j}$ recursively as follows:

$$Y_j = \begin{bmatrix} Y_{j-1} & \boldsymbol{y}_j \end{bmatrix}, T_j = \begin{bmatrix} T_{j-1} & -t_j T_{j-1} Y_{j-1}^\top \boldsymbol{y}_j \\ \boldsymbol{0} & t_j \end{bmatrix}. \tag{3}$$

In this case, the following equation holds

$$H_1 H_2 \cdots H_j = I - Y_j T_j Y_j^\top. \tag{4}$$

As shown by Eq.(4), we can rewrite the product of the Householder matrices $H_1 H_2 \cdots H_j$ in a simple block matrix form. Here $I - Y_j T_j Y_j^\top$ is called the compact WY representation of the product of the Householder matrices. Figure 3 shows the orthogonalization algorithm. Hereinafter, we refer to this orthogonalization algorithm as the compact WY orthogonalization.

1: **for** $j = 1$ to m **do**
2: Generate $\boldsymbol{v}_j$ from $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_{j-1}$.
3: $\boldsymbol{v}'_j = \left(I - Y_{j-1}T_{j-1}^\top Y_{j-1}^\top\right)\boldsymbol{v}_j$.
4: Compute $\boldsymbol{y}_j$ and t_j by using $\boldsymbol{v}'_j$.
5: (3) : Update Y_j and T_j by using t_j, $\boldsymbol{y}_j$, T_{j-1} and Y_{j-1}.
6: $\boldsymbol{q}_j = \left(I - Y_j T_j Y_j^\top\right)\boldsymbol{e}_j$.
7: **end for**

Fig. 3: Householder orthogonalization algorithm in terms of the compact WY representation.

3.3 Comparison of the orthogonalization algorithms

The compact WY orthogonalization has a stable orthogonality arising from the Householder transformations, and its mathematical calculation is mainly performed by BLAS level-2 operations such as the product of a matrix and a vector and a rank-1 update operation. As a result, this orthogonalization has more stable and sophisticated orthogonality, and it is more effective for parallel computing than the MGS. Table 1 displays the differences in performance of the three orthogonalization methods, considered in the above sections. In this table, "House" denotes the Householder orthogonalization and "cWY" denotes the compact WY orthogonalization. *Computation* denotes the order of the computational cost. *Synchronization* denotes the order of the number of synchronizations. *Orthogonality* denotes the norm $\|Q^\top Q - I\|$, where $Q = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n]$. ϵ denotes the machine epsilon and κ denotes the condition number of a matrix. These are the results obtained from [1] and [13].

Table 1: Comparison of the orthogonalization methods [1] [13].

methods	*Computation*	*Synchronization*	*Orthogonality*
MGS	$O(2m^2n)$	$O(m^2)$	$O(\epsilon\kappa(A))$
House	$O(4m^2n)$	$O(m^2)$	$O(\epsilon)$
cWY	$O(4m^2n)$	$O(m)$	$O(\epsilon)$

4. Inverse iteration method with the compact WY orthogonalization

In this section, we present a new inverse iteration algorithm. This new algorithm is described in Figure 4 and is based on DSTEIN, a LAPACK code of the classical inverse iteration. We change the orthogonalization process from the MGS to the Householder transformation in terms of the compact WY representation. In other words, we rewrite the MGS algorithm (from line 4 to 15 in Figure 1) to the compact WY orthogonalization algorithm shown in Figure 3.

Next, we explain an application of the compact WY orthogonalization to the classical inverse iteration. For the DSTEIN algorithm, we need not know the index j_c which denotes the j_c-th eigenvalue of the cluster in computing the j_c-th eigenvector. However, we must know the index for the compact WY orthogonalization when we compute and update T_j, Y_j. To overcome the above difficulty, we introduce a variable j_c on line 9, and we can recognize it.

This introduction of j_c enables us to execute the intended program. However, we do not get accurate results because the compact WY orthogonalization algorithm includes many equations with a comparatively large number of elements such as $Y_{j_c}T_{j_c}^{\top}Y_{j_c}^{\top}$ and $Y_{j_c}T_{j_c}Y_{j_c}^{\top}$ and they may cause overflow. To overcome this difficulty, we have to normalize $\boldsymbol{v}_j^{(k)}$ on line 6, and this normalization excludes overflow.

Finally, to reduce the computational cost, we transform parts of the equations. There are some examples in Figure 4 for $j_c = 1$, i.e., $j = 2$ in Figure 3.

In the original DSTEIN algorithm, we need not know that λ_{j_1} $(j_1 = j - 1)$ is the first eigenvalue of the cluster. However, we must compute $\boldsymbol{y}_1$ and t_1. Therefore, at the starting point of the computation of the eigenvector associated with the second eigenvalue, we compute $\boldsymbol{y}_1$ and t_1. At this time, because T_1 is a 1×1 matrix, i.e., a scalar, we can omit the computation of some of Eq.(3) and only compute them. In addition, because we normalize $\boldsymbol{v}_j^{(k-1)}$ on line 6 so that $\boldsymbol{v}_j^{(k-1)} = \boldsymbol{q}_1$, we need not compute $\boldsymbol{y}_1$ again. As shown in lines 15 and 17, to save the computation step that is required when using BLAS, we change the formula from the matrix-vector operations to the vector operations. In addition, we implemented another formula because of the benefit of using BLAS computations to reduce the computational cost in line 23.

1: **for** $j = 1$ to n **do**
2: Generate $\boldsymbol{v}_j^{(0)}$ from random numbers.
3: $k = 0$
4: **repeat**
5: $k \leftarrow k + 1$.
6: Normalize $\boldsymbol{v}_j^{(k-1)}$.
7: (1) : Compute $\boldsymbol{v}_j^{(k)}$ by using $\boldsymbol{v}_j^{(k-1)}$.
8: **if** $|\tilde{\lambda_j} - \tilde{\lambda}_{j-1}| \leq 10^{-3}\|T\|$, **then**
9: $j_c \leftarrow j - j_1$.
10: **if** $j_c = 1$ and $k = 1$, **then**
11: Compute $Y_1 = \boldsymbol{y}_1$ and $T_1 = t_1$ by using $\boldsymbol{v}_{j_1-1}$.
12: **end if**
13: Normalize $\boldsymbol{v}_j^{(k)}$.
14: **if** $j_c = 1$, **then**
15: $\boldsymbol{v}_2' \leftarrow \boldsymbol{v}_2 - t_1\langle \boldsymbol{y}_1, \boldsymbol{v}_j^{(k)}\rangle \boldsymbol{y}_1$.
16: Compute $\boldsymbol{y}_2$ and update Y_2 by using $\boldsymbol{v}_2'$.
17: Compute t_2 and $T_{1,2} = -t_2 t_1 \langle \boldsymbol{y}_1, \boldsymbol{y}_2\rangle$ and update T_2.
18: **else**
19: $\boldsymbol{v}_{j_c+1}' = \left(I - Y_{j_c}T_{j_c}^{\top}Y_{j_c}^{\top}\right)\boldsymbol{v}_j^{(k)}$.
20: Compute $\boldsymbol{y}_{j_c+1}$ and t_{j_c+1} by using $\boldsymbol{v}_{j_c+1}'$.
21: (3) : Update Y_{j_c+1} and T_{j_c+1} by using t_{j_c+1}, $\boldsymbol{y}_{j_c+1}$, T_{j_c} and Y_{j_c}.
22: **end if**
23: $\boldsymbol{v}_j^{(k)} \leftarrow \left(I - Y_{j_c+1}T_{j_c+1}Y_{j_c+1}^{\top}\right)\boldsymbol{e}_{j_c+1}$.
24: **else**
25: $j_1 \leftarrow j$.
26: **end if**
27: **until** some condition is met.
28: Normalize $\boldsymbol{v}_j^{(k)}$ to $\boldsymbol{v}_j$.
29: **end for**

Fig. 4: Algorithm of the compact WY inverse iteration.

5. Numerical experiments

In this section, we describe some numerical experiments performed using DSTEIN and DSTEIN-cWY in parallel computers, and we compare the computation time. DSTEIN is implemented in the classical inverse iteration algorithm, and DSTEIN-cWY is implemented in the new inverse iteration algorithm presented in the previous section.

5.1 Contents of the numerical experiments

In this subsection, we report computations of all the eigenvectors associated with eigenvalues of some matrices by using DSTEIN and DSTEIN-cWY on parallel computers, and we compare the calculation time. In these experiments, we compute the approximate eigenvalues by using LAPACK's program DSTEBZ, which is capable of computing them by using the bisection method. We record the calculation time for DSTEIN and DSTEIN-cWY using TIME, which is the internal function of Fortran and returns an integer number of times.

In the experiments, we use two computers equipped with multicore CPUs, and we implement those algorithms by using GotoBLAS2 [6], which is implemented to parallelize BLAS operations by assigning them to each CPU core. Table 2 shows the specifications of two computers.

All the matrices in the experiments are the glued-Wilkinson matrices $W_g^{\dagger}$, which are real symmetric and have

Table 2: The specification of Computer 1 and 2

	Computer 1	Computer 2
CPU	AMD Opteron 2.0GHz 32cores(8cores×4)	Intel Xeon 2.93GHz 8cores(4cores×2)
Memory	16GB	32GB
Compiler	Gfortran-4.4.5	Gfortran-4.4.5
LAPACK	LAPACK-3.3.0	LAPACK-3.3.0
BLAS	GotoBLAS2-1.13	GotoBLAS2-1.13

dimensions on the order of thousands. More precisely, $W_g^\dagger$ consists of the block matrix $W_{21}^\dagger \in \mathbb{R}^{21\times 21}$ and the scalar parameter $\delta \in \mathbb{R}^{1\times 1}$ and is defined as follow:

$$W_g^\dagger = \left[\begin{array}{c|c|cc|c} W_{21}^\dagger & \delta & & & \\ \hline \delta & W_{21}^\dagger & \delta & & \\ \hline & \delta & \ddots & \ddots & \\ & & \ddots & \ddots & \delta \\ \hline & & & \delta & W_{21}^\dagger \end{array}\right], \tag{5}$$

where $W_{21}^\dagger$ is defined by

$$W_{21}^\dagger = \begin{bmatrix} 10 & 1 & & & & & \\ 1 & 9 & 1 & & & & \\ & 1 & \ddots & \ddots & & & \\ & & \ddots & 0 & \ddots & & \\ & & & \ddots & \ddots & 1 & \\ & & & & 1 & 10 \end{bmatrix}, \tag{6}$$

and δ satisfies $0 < \delta < 1$ and is also the semi-diagonal element of $W_g^\dagger$. Since $W_g^\dagger$ is real symmetric tridiagonal and its semi-diagonal elements are nonzero, all the eigenvalues of $W_g^\dagger$ are distinct and real, and they are divided into 11 clusters of close eigenvalues. When δ is small, the distance between the minimum and maximum eigenvalues in any cluster is small. In our experiments, we set $\delta = 10^{-4}$.

Computing eigenvalues and eigenvectors of the glued-Wilkinson matrix is one of the benchmark problems of eigenvalue decomposition. For example, in [2] and [4], the glued-Wilkinson matrix was used to evaluate the performance of the algorithm.

5.2 Results of the experiments

Table 3 shows the results of the experiments on Computer 1 that were mentioned in the previous section, and Table 4 shows the results of the experiments on Computer 2. In addition, Figure 5 illustrates the results in Tables 3 and 4 through graphs. The dotted line corresponds to Table 3 and the straight line to Table 4.

From Table 3 and 4, we see that, on both Computers 1 and 2, all the eigenvectors of the glued-Wilkinson matrix $W_g^\dagger$ with dimensions of the order of several thousand are computed in parallel.

It is noted that DSTEIN-cWY is faster than DSTEIN. We see that the change from MGS to the compact WY orthogonalization on the DSTEIN code in parallel computing results in a significant reduction in computation time. We introduce a barometer $t/t_{\rm cwy}$ of the reduction effect by using the program DSTEIN-cWY which depends on n, the dimension of $W_g^\dagger$. On Computer 1, the maximum value of $t/t_{\rm cwy}$ is 7.46 for $n = 7,350$ and $t/t_{\rm cwy} = 6.71$ for $n = 10,500$. On Computer 2, the maximum value of $t/t_{\rm cwy}$ is 3.16 for $n = 4,200$ and $t/t_{\rm cwy} = 2.03$ for $n = 10,500$. Considering these facts, even if the dimension of $W_g^\dagger$ is larger than that in these examples, we cannot expect that the computation time can be further shortened by using DSTEIN-cWY. In the following section, we discuss this computation time phenomenon.

5.3 Discussion on numerical experiments

It is shown that DSTEIN-cWY is faster than DSTEIN for any dimension n of the glued-Wilkinson matrix both on Computers 1 and 2. As mentioned earlier, according to the theoretical background in section 3.3, this result shows that the compact WY orthogonalization is an effective algorithm in parallel computing.

The cause of this is related to the time required for floating-point arithmetic and for synchronization in parallel computing. The floating-point computation time increases with increasing n because the elements of the computation increase. In comparison, the synchronization cost does not change significantly even if n becomes larger. Therefore, in parallel computing, DSTEIN, which contains MGS (for which the number of synchronizations is large), creates a huge bottleneck for the synchronization cost when n is small. This bottleneck gradually becomes less when n is larger. However, DSTEIN-cWY has a smaller bottleneck for the synchronization cost because the compact WY orthogonalization requires less synchronization, and the floating-point computation time increases to a value greater than that of DSTEIN. This reduction effect is seen in Table 3 and 4.

6. Conclusions

In this study, we present a new inverse iteration algorithm for computing all the eigenvectors of a real symmetric tridiagonal matrix. The new algorithm is equipped with the compact WY algorithm in the orthogonalization process. We have performed numerical experiments for computing eigenvectors of certain real symmetric tridiagonal matrices that have many clusters with several thousand dimensions by using two types of inverse iteration algorithms on parallel computers. The results show that the compact WY inverse iteration is more efficient than the classical one owing to the reduction in computation time.

The main reason for this outcome is the parallelization efficiency with respect to computation time. The parallelization efficiency of the compact WY orthogonalization is

Table 3: DSTEIN and DSTEIN-cWY on Computer 1. Here, n is the dimension of the glued-Wilkinson matrix, t and $t_{\rm cwy}$ are computation time (sec.) by DSTEIN and DSTEIN-cWY on Computer 1, respectively.

n	1050	2010	3150	4200	5250	6300	7350	8400	9450	10500
t	2	9	25	55	106	178	276	400	560	758
$t_{\rm cwy}$	1	2	5	10	16	25	37	57	81	113
$t/t_{\rm cwy}$	2.00	4.50	5.00	5.50	6.63	7.12	7.46	7.02	6.91	6.71

Table 4: DSTEIN and DSTEIN-cWY on Computer 2. Here, n is the dimension of the glued-Wilkinson matrix, t and $t_{\rm cwy}$ are computation time (sec.) by DSTEIN and DSTEIN-cWY on Computer 2, respectively.

n	1050	2010	3150	4200	5250	6300	7350	8400	9450	10500
t	1	3	8	19	37	67	109	159	225	309
$t_{\rm cwy}$	1	1	3	6	13	25	45	73	107	152
$t/t_{\rm cwy}$	1.00	3.00	2.67	3.16	2.84	2.68	2.42	2.17	2.10	2.03

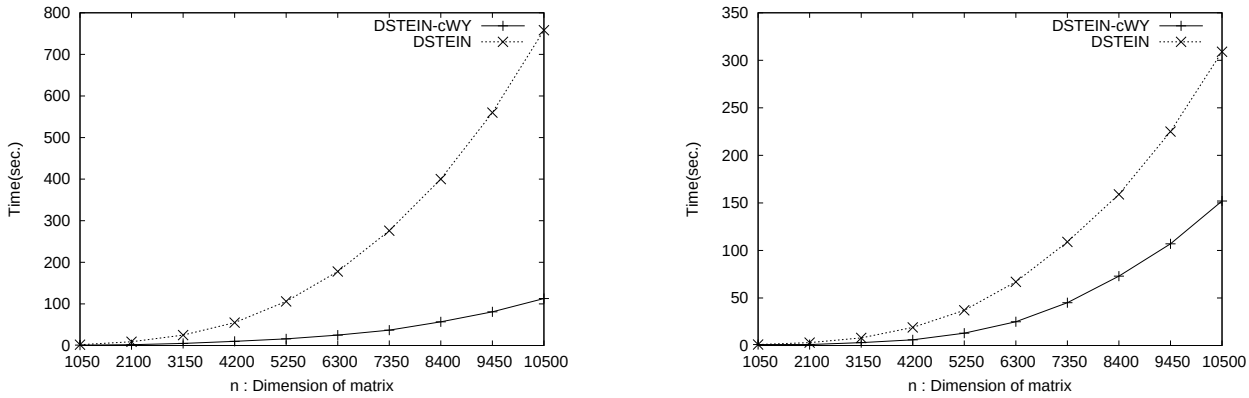

Fig. 5: Dimension n of the glued-Wilkinson matrix and the computation time by DSTEIN and DSTEIN-cWY. the above graph corresponds to Computer1 and the below Computer 2, respectively.

greater than that of the MGS orthogonalization where the classical inverse iteration is used. As the number of cores of the CPU increases, the parallelization efficiency increases.

In future studies, we will try to apply the new inverse iteration algorithms to other types of matrix eigenvector problem, such as eigenvectors of a real symmetric banded matrix, or singular vectors of a bidiagonal matrix.

Acknowledgements.

The authors thank Professor Yusaku Yamamoto of Kobe University for providing several helpful suggestions.

References

[1] J. W. Demmel, L. Grigori, M. Hoemmen and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, LAPACK Working Notes, No.204, 2008.

[2] J. W. Demmel, O. A. Marques, B. N. Parlett, and C. Vömel, *Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers*, SIAM J. Sci. Comput., Vol. 30, No. 3, pp. 1508-1526, 2008.

[3] I. S. Dhillon, *A new O(n^2) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*, Ph.D. thesis, Computer Science Division, University of California, Berkeley, California, available as UC Berkeley Technical Report UCB//CSD-97-971, 1997.

[4] I. S. Dhillon, B. N. Parlett, and C. Vömel, *Glued matrices and the MRRR algorithm*, SIAM J. Sci. Comput., Vol. 27, No. 2, pp. 496-510, 2005.

[5] G. Golub and C. van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, 1996.

[6] GotoBLAS2, http://www.tacc.utexas.edu/tacc-projects/gotoblas2/.

[7] M. Gu and S. C. Eisenstat, *A stable algorithm for the rank-1 modification of the symmetric eigenproblem*, Computer Science Department Report YALEU/DCS/RR-916, Yale University, New Haven, CT, 1992.

[8] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Mat. Anal. Appl., Vol. 16, pp. 172-191, 1995.

[9] LAPACK, http://www.netlib.org/lapack/.

[10] G. Peters and J. Wilkinson, *The calculation of specified eigenvectors by inverse iteration*, contribution II/18, in Linear Algebra, Handbook for Automatic Computation, Vol. II, Springer-Verlag, Berlin, pp. 418-439, 1971.

[11] R. Schreiber and C. van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Stat. Comput., Vol. 10, No. 1, pp. 53-57, 1988.

[12] H. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., Vol. 9, No. 1, pp. 152-163, 1988.

[13] Y. Yamamoto, *Parallelization of orthogonalization in Arnoldi process based on the compact WY representation*, Proceedings of the Annual Conference of JSIAM, pp. 39-40, 2010 (in Japanese).