Performance Evaluation of a Combination of the Parallel Bisection Method and the Block Inverse Iteration Method with Reorthogonalization for Eigenvalue Problems on MIC processor

Sho Araki[1], **Hiroyuki Ishigami**[2], **Masayuki Osawa**[1], **Kinji Kimura**[1] **and Yoshimasa Nakamura**[1]
[1]Graduate School of Informatics, Kyoto University, Kyoto, Japan
[2]Yahoo Japan Corporation, Tokyo, Japan

Abstract— *We discuss the implementation, performance tuning, and evaluation of an eigensolver of real symmetric tridiagonal matrices using the bisection method and the block inverse iteration method with reorthogonalization on Intel Xeon Phi (Xeon Phi) many integrated core (MIC) processor. We develop an OpenMP thread parallel program for the eigensolver for Xeon Phi and experimentally determine the optimal block size parameter for both the MIC and CPU environments. Moreover, we perform experiments for evaluating the performance of the algorithm with the optimal block size. The eigensolver exhibits higher computation speed and accuracy in the MIC environment than the MRRR algorithm, which is known as the conventional high-speed eigensolver.*

Keywords: eigenpair problem; bisection method; block inverse iteration method with reorthogonalization; parallel computing; MIC processor;

1. Introduction

This study is mainly concerned with the standard eigenvalue problem for an $n \times n$ dense symmetric matrix A, as follows

$$Av_i = \lambda_i v_i, i = 1, \ldots, n,$$

where $\{\lambda_i\}_{i=1}^n (\lambda_1 \leq \cdots \leq \lambda_n)$ is the sequence of real eigenvalues of A. We refer to the pair of eigenvalue and eigenvector as "eigenpair" below.

There are many applications of the eigenvalue problem, since problems in computational science are often reduced to primal linear programming. Sometimes, only a small number of eigenpairs are required. Target matrices of eigenpair problems have become increasingly large, and the solvers need to be accelerated by means of parallel computing. Intel Xeon Phi is a parallel processor exhibiting higher watt performance than ordinary CPU (Central Processing Units), and was very similar to GPU (Graphics Processing Units) until the previous "Knights Corner" generation. The largest difference between MIC (Many Integrated Core) architecture and GPU is the remarkably high compatibility of the former with CPU architecture, which is capable of compiling existing source code of C and Fortran languages without major modifications. In addition, "Knights Landing" generation processors, released in 2016, are mounted directly on CPU sockets. Therefore, the bandwidth limitation of PCI-Express is no longer an issue. Moreover, the higher compatibility of the updated Xeon Phi with ordinary CPU widens the range of application of MIC architecture in numerical calculations. The solver of eigenpair problems considered in this paper is an example.

The computation of the eigenpairs of a real symmetric matrix A generally employs the transformation of A to a symmetric tridiagonal or band matrix. Then, the eigenpairs of the tridiagonal or band matrix are computed. The eigenvalues of the transformed tridiagonal or band matrix are equal to the eigenvalues of the given matrix A, and the desired eigenvectors are obtained by the reverse transformation of the previous tridiagonal or band transformation. Many types of highly efficient parallel computing methods have been proposed for pre-processing and post-processing. Subsequently, one may concentrate on the eigenvalue computation part.

In this paper, we consider the bisection method [1] for obtaining the set (or a subset) of the eigenvalues of a pre-processed real symmetric tridiagonal matrix. It should be noted that there are several well-known methods for obtaining the eigenvectors of a real symmetric tridiagonal matrix, such as reverse iteration, MRRR (Multiple Relatively Robust Representation) [2], QR, and Divide and Conquer methods. MRRR can be used to compute all or a part of the eigenpairs of a given matrix. By contrast, the QR and Divide and Conquer methods can be used to compute all eigenpairs. The advantage of the QR method is accuracy in terms of absolute error measurement, whereas the advantage of the Divide and Conquer method is high computation speed in parallel environments. Even though the bisection method is suitable for the partial eigenvalue problem as well, we evaluate its performance in obtaining all eigenvalues, in order to make a comparison with the QR and Divide and Conquer algorithms. For parallel computing, Intel Math Kernel Library [10] (MKL) provides the routines for the methods.

The paper is organized as follows. The second section provides a brief presentation of the specifications of the "Knights Landing" generation Xeon Phi processor. In the third section, we present the implementation for MIC architecture of the parallel eigensolver using a combination of the bisection method [1] and the block inverse iteration method with reorthogonalization [3]–[6]. In the fourth section, we present the results of performance tuning by block size and the benchmarks for computation speed and accuracy. Moreover, a comparison with other methods for Xeon Phi is presented. The fifth section concludes this study.

ISBN: 1-60132-466-9, CSREA Press ©

2. Xeon Phi Processor

Herein, the specifications of the Xeon Phi processor are presented.

Xeon Phi "Knights Corner", manufactured by Intel Corporation, is an accelerator board based on MIC (Many Integrated Core) architecture, connected to the host computer through a PCI Express slot. The co-processor competes with general purpose GPU (GPGPU) boards, such as NVIDIA Tesla and AMD FirePro series. The main advantage of the co-processor is the high compatibility with Intel CPU of x86 architecture, which is capable of running existing programs written for x86 CPU without major modification of the source code. In particular, unlike GPU, the MIC co-processor core can execute conditional instructions, such as **if** and **switch** statements of C language code, independently of the other cores. In addition, the latest architecture of Xeon Phi, i.e., "Knights Landings", is no longer a co-processor, but rather a processor that is mounted directly on the CPU socket. Even though the previous architectures of Xeon Phi co-processor needed to be accessed via the host operating system, the latest "Knights Landing" architecture is capable of self-booting and can control the operating system natively. "Knights Landing" Xeon Phi has 64 to 72 cores, each executing up to four threads of SMT (Simultaneous Multi-Threading). In addition, the latest Xeon Phi processor supports Advanced Vector Extension (AVX) of 512 bit extended instructions.

Xeon Phi accelerates parallel computing by means of a large number of execution threads, despite that each core has lower clock frequency than CPU, as in the case of GPU. Furthermore, in contrast with GPU and the previous co-processors, there are no limitations with regard to independent execution of conditional instructions and the bandwidth of PCI-Express. This enables the processor to run the bisection method faster and is the main motivation for evaluating the performance of the eigensolver using a combination of the bisection method and the block inverse iteration method with reorthogonalization in MIC environment. The implementation will be discussed in detail in the following section.

3. Implementation Details of the Target Eigensolver

The computation of the eigenpairs of a real symmetric matrix is generally performed through the transformation of the target matrix A to a symmetric band matrix, and subsequent computation of the eigenpairs of the band matrix. The transformed symmetric band matrix, often a tridiagonal matrix, has the same eigenvalues as the original matrix. Then, the eigenvectors of the original matrix are obtained by the reverse transformation of the previous band transformation. In this section, we briefly present this procedure.

3.1 Implementation of the Bisection Method

Herein, we discuss the implementation of the bisection method for real symmetric tridiagonal matrices. Let T be an $n \times n$ tridiagonal matrix, $d_i (i = 1, \cdots, n)$ the sequence of diagonal entries, and $e_i (i = 1, \cdots, n-1)$ the sequence of subdiagonal entries. Moreover, the eigenvalues of T are assumed to be real numbers. We note that the sequences d_i, e_i, and μ_i are aligned as array type of double precision floating point numbers in the memory. For simplicity, at the beginning of the computation, if the off-diagonal component is smaller than a specified threshold, we divide the matrix by assuming that this component is 0. This operation is called splitting.

Algorithm 1 Main routine of thread parallel bisection method

```
1:  k_b = 1, k_e = 1
2:  repeat
3:      !$omp parallel do private(r_t, c_t)
4:      do k = k_b to k_e
5:          c_t = 0
6:          do j = 1 to n
7:              r_j = d_j − e_{j−1}^2 / r_{j−1} − μ_k  (e_0 = 0)
8:              if r_t ≤ p_min then
9:                  r_t = min(r_t, −p_min), c_t = c_t + 1
10:         c_k = c_t
11:     Manage tasks & Check convergence ▷ Update k_b, k_e, μ_k
12: until k_b > k_e
13: return λ = μ_k
```

The bisection method is an algorithm for computing the eigenvalues of a real symmetric matrix using binary search. It is proposed in [1], and its implementation for ordinary CPU is provided by the DSTEBZ routine of LAPACK (Linear Algebra PACKage) [8]. DSTEBZ for obtaining all eigenvalues may be briefly described as follows.

1) Calculate the range $[\mu_i, \mu_r]$ containing all eigenvalues of T.
2) Calculate the number of eigenvalues smaller than μ_i and μ_r.
3) Obtain all eigenvalues using binary search.

Implementations of (2) and (3) are provided by DLAEBZ, which is a subroutine of DSTEBZ.

We adopt the implementation introduced in [6] as a thread parallel bisection method suitable for shared memory systems such as MIC environments. Algorithm 1 is a modification of DLAEBZ.

The do loop that begins at the third line is parallelized with privatization of the variables c_t and r_t for each thread. The task management shown in line 7 should be exclusive. However, it should be of significantly smaller complexity than the calculation of the do loop. Therefore, the task managements are executed serially in the proposed method. We note that the other routines of DSTEBZ, except for algorithm 1, are also executed serially using the implementation of the original DSTEBZ routines.

Even though the algorithm contains synchronizations whose number is equal to that of repeat-until iterations, it hardly affects computation time due to the absence of exclusive processing between threads in repeat-until iterations.

3.2 Implementation of the Inverse Iteration Method

Herein, we briefly present the block inverse iteration method with reorthogonalization [3]–[6] for the eigenvector problem. Moreover, Parallel implementation of each method for shared memory-systems is considered.

For an $n \times n$ real symmetric tridiagonal matrix T, let $\lambda_i \in \mathbb{R}(\lambda_1 < \cdots < \lambda_n)$ be its eigenvalues and $\boldsymbol{q}_i \in \mathbb{R}$ be the eigenvectors corresponding to λ_i. If $\tilde{\lambda}_i$ is the approximate value of λ_i, and the initial vector $\boldsymbol{v}_i^{(0)}$ is randomly (uniformly) generated, then the vector $\boldsymbol{v}_i^{(j)}$ converges to the eigenvector $\boldsymbol{q}_i$ as $j \to \infty$ in the following linear iterative equation

$$(T - \tilde{\lambda}_i I)\boldsymbol{v}_i^{(j)} = \boldsymbol{v}_i^{(j-1)}, j = 1, 2, \ldots, \quad (1)$$

where I is the $n \times n$ identity matrix. The inverse iteration method [3]–[5] is based on the above procedure. The complexity of this method for obtaining $m(< n)$ eigenvectors is $O(mn)$. Practically, the vector $\boldsymbol{v}_i^{(j)}$ must be normalized at each iteration to avoid overflow and underflow. The obtained eigenvectors are orthogonal if the eigenvalues of T are sufficiently separated. By contrast, it is known that the eigenvectors may fail to be orthogonal if the eigenvalues of T are clustered. In this case, it is proposed that eigenvectors corresponding to clustered eigenvalues should be reorthogonalized.

We adopt the block inverse iteration method with reorthogonalization proposed in [6] as the implementation of the inverse iteration method for MIC environment. This is a modification of the simultaneous inverse iteration method. The modified algorithm aims at reducing memory consumption by a division of the m_c eigenvectors (corresponding to a certain eigenvalue) into blocks of size r. For simplicity, we assume that the block size r divides m_c.

These eigenvectors are regarded as a group of blocks of size r and are computed individually. In the following, it is shown how the eigenvectors $\boldsymbol{q}_{(j-1)r+1}, \ldots, \boldsymbol{q}_{jr}$ can be obtained after the computation of $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{(j-1)r}$, where $j = 1, 2, \ldots, m_c/r$.

We first generate an $n \times r$ random matrix $V_{j,r}^{(0)}$ and obtain an orthonormal matrix $Q_{j,r}^{(0)}$ by QR decomposition $V_{j,r}^{(0)} := Q_{j,r}^{(0)} R_{j,r}^{(0)}$. Then, the eigenvectors $\boldsymbol{q}_{(j-1)r+1}, \ldots, \boldsymbol{q}_{jr}$ are obtained in three steps from the initial matrix $Q_{j,r}^{(0)}$, as follows.

1) Solve the r simultaneous equations

$$(T - \tilde{\lambda}_k I)\boldsymbol{v}_k^{(i)} = \boldsymbol{q}_k^{i-1}, k = (j-1)r + 1, \ldots, jr. \quad (2)$$

2) Reorthogonalize $V_{j,r}^{(i)}$ to be orthogonal to the vectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{(j-1)r}$.
3) Apply QR decomposition to the $n \times r$ matrix consisting of the vectors obtained at step 2 and define the matrix Q as $Q_{j,r}^{(i)}$.

The most obvious difference with the simultaneous inverse iteration method is that the number of simultaneous equations solved at step 1 is r. There is no data synchronization in the parallel computation in step 1.

Algorithm 2 Block inverse iteration method with reorthogonalization

1: **do** $i = 1$ to m_c/r
2: $j = 0$
3: Generate $V_{i,r}^{(0)} = [\boldsymbol{v}_{(i-1)r+1}^{(0)} \cdots \boldsymbol{v}_{ir}^{(0)}]$
4: $V_{i,r}^{(0)} = Q_{i,r}^{(0)} R_{i,r}^{(0)}$ ▷ QR decomposition
5: **do** $k = (i-1)r + 1$ to ir
6: $T - \tilde{\lambda_k} I = P_k L_k U_k$ ▷ LU decomposition with partial pivot selection
7: **repeat**
8: $j = j + 1$
9: **do** $k = (i-1)r + 1$ to ir
10: Solve $P_k L_k U_k \boldsymbol{v}_k^{(j)} = \boldsymbol{q}_k^{(j-1)}$
11: $V_{i,r}^{(j)} = V_{i,r}^{(j)} - Q_{(i-1),r} Q_{(i-1),r}^{\intercal} V_{i,r}$
12: $V_{i,r}^{(j)} = \bar{Q}_{i,r}^{(j)} R_{i,r}^{(j)}$ ▷ QR decomposition
13: $\bar{Q}_{i,r}^{(j)} = \bar{Q}_{i,r}^{(j)} - Q_{(i-1),r} Q_{(i-1),r}^{\intercal} \bar{Q}_{i,r}^{(j)}$
14: $\bar{Q}_{i,r}^{(j)} = Q_{i,r}^{(j)} R_{i,r}^{(j)}$ ▷ QR decomposition
15: **until** converge
16: $Q_{i,r} = [Q_{(i-1),r} Q_{i,r}^{(j)}] (Q_r = [Q_{1,r}^{(j)}])$
17: **return**

Table 1: Specification of the experimental environment (CPU)

CPU	Intel Xeon E5-2695 v4 x2 (2.10GHz, 18 cores x2)
RAM	DDR4-2400 128GB
Compiler	icc 16.0.4, ifort 16.0.4
Options	-O3 -ipo -xCORE-AVX2 -fp-model precise -qopenmp -mkl
Software	Intel Math Kernel Library 11.3.4

The block inverse iteration method with reorthogonalization obtains the eigenvectors of a given matrix by iteration of the above three steps. If the block parameter is $r = m_c$, the algorithm is equivalent to the simultaneous inverse iteration method, whereas if $r = 1$, the algorithm is equivalent to the inverse iteration method.

Algorithm 2 shows pseudo-code for the block inverse iteration method using the BCGS2 reorthogonalization method. The operation of step 2 appears in lines 11 and 13, and the operation of step 3 appears in lines 12 and 14.

The dominant part of the block inverse iteration method could be implemented, with efficient execution of matrix multiplications, on SSE and AVX enabled processors.

4. Performance Evaluation

In this section, we present the results of the numerical experiments that were conducted to evaluate the performance of the eigensolver using the parallel bisection method (PBi) and the block inverse iteration method with reorthogonalization (BIR), mentioned in section 3. The results in CPU and MIC environments are shown in Tables 1 and 2, respectively.

Test matrices for the evaluation are $n \times n$ symmetric tridiagonal matrices T_1 and T_2, where T_1 is a random matrix with uniformly distributed random numbers $d_i, e_i \in [0, 1]$, and

Table 2: Specification of the experimental environment (MIC)

CPU	Intel Xeon Phi 7250 (1.4GHz, 68 cores)
RAM	DDR4-2133 96GB + MCDRAM 16GB (Cache mode)
Compiler	icc 16.0.4, ifort 16.0.4
Options	-O3 -ipo -xMIC-AVX512 -fp-model precise -qopenmp -mkl
Software	Intel Math Kernel Library 11.3.4

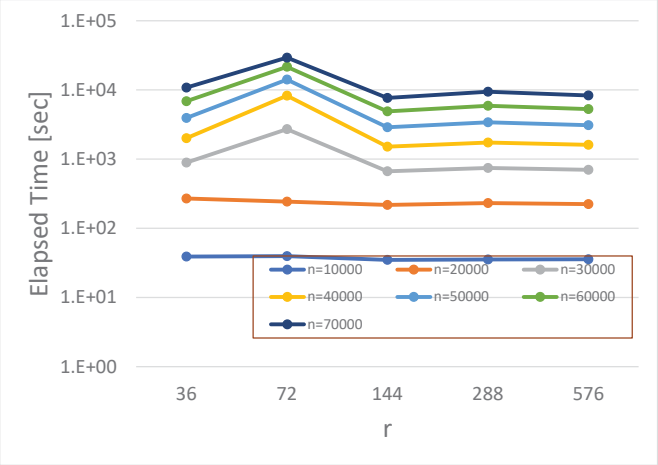

Fig. 1: Computation time for each block size (CPU, T_1)

T_2 has uniform diagonal entries $d_i = 2(i = 1, \ldots, n)$ and subdiagonal entries $e_i = 1(i = 1, \ldots, n-1)$.

4.1 Determining the Optimal Block Size for the Eigensolver

We first determine the optimal block size for the BIR algorithm. To this end, we use the following search method.

1) Set the initial block size r to the number of processor cores of each experimental environment.
2) Measure the computation time for obtaining all eigenvalues and eigenvectors of T_1 and T_2.
3) If the computation time is longer than that of previous two trials, stop searching and let the block size giving minimum computation time be optimal.
4) Otherwise, set the block size $r := 2 \times r$ and continue searching.

Figures 1, 2, 3, and 4 show the computation time for T_1 and T_2 using the block inverse iteration method with reorthogonalization for different block sizes r on CPU and MIC environments for the matrices T_1 and T_2 of size n. It should be noted that all graphs are logarithmic.

We optimize the parameter for the BIR algorithm, since it was more than 100 times slower than the PBi algorithm. In addition, if we use hyper-threading technology, the PBi algorithm becomes faster, whereas the BIR algorithm becomes slower. Thus, hyper-threading technology is not adopted.

These graphs show that block size $r = 144$ for CPU environment and $r = 2176$ for MIC yield the best performance.

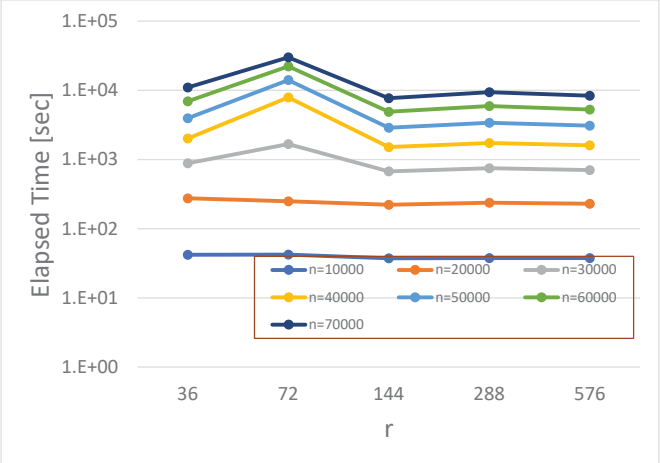

Fig. 2: Computation time for each block size (CPU, T_2)

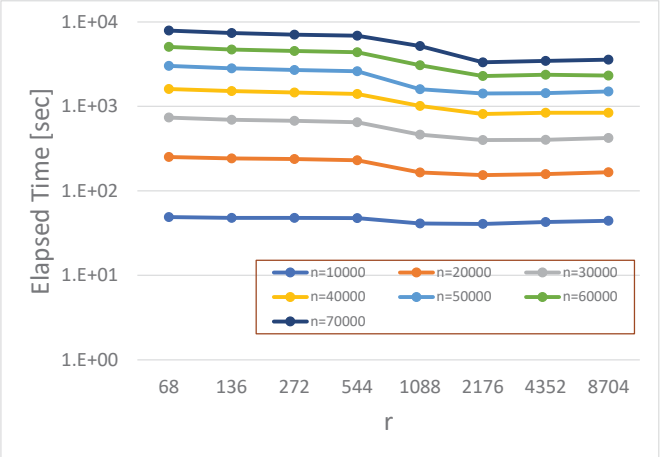

Fig. 3: Computation time for each block size (MIC, T_1)

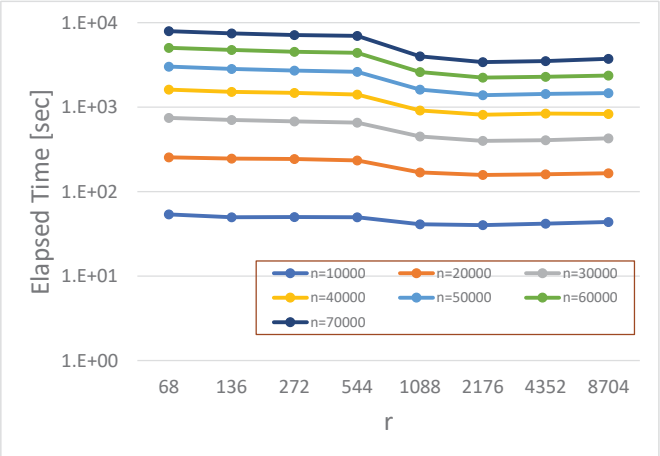

Fig. 4: Computation time for each block size (MIC, T_2)

4.2 Comparison with Other Algorithms

We compare the execution time for obtaining all eigenvalues and eigenvectors of the matrices T_1 and T_2 using the PBi+BIR algorithm with the corresponding time for the MRRR al-

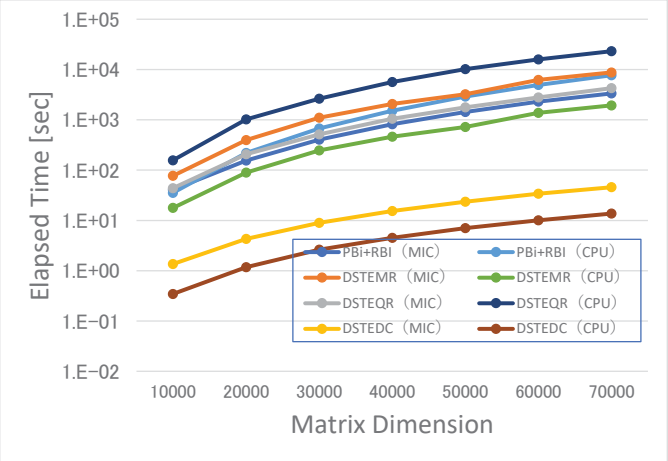

Fig. 5: Computation time for T_1

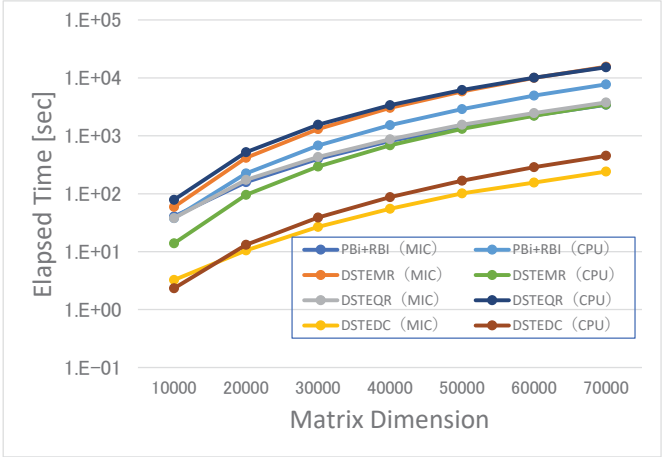

Fig. 6: Computation time for T_2

Table 3: Computation time for T_1

	10000	30000	50000	70000
PBi+BIR (CPU)	35.4	671	2.90+3E	7.69E+03
PBi+BIR (MIC)	40.9	403	1.43+3E	3.35E+03
DSTEMR (CPU)	17.7	248	717	1.94E+03
DSTEMR (MIC)	77.0	1.10E+03	3.22E+03	8.76E+03
DSTEQR (CPU)	157	2.64E+03	1.01E+04	2.31E+04
DSTEQR (MIC)	43.8	515	1.77E+03	4.26E+04
DSTEDC (CPU)	0.345	2.63	7.03	13.7
DSTEDC (MIC)	1.36	8.91	23.6	45.6

Table 4: Computation time for T_2

	10000	30000	50000	70000
PBi+BIR (CPU)	37.8	677	2.89E+03	7.71E+03
PBi+BIR (MIC)	40.4	401	1.39E+03	3.43E+03
DSTEMR (CPU)	13.9	295	1.31E+03	3.50E+03
DSTEMR (MIC)	59.1	1.30E+03	5.85E+03	1.56E+04
DSTEQR (CPU)	78.4	1.55E+03	6.15E+03	1.52E+04
DSTEQR (MIC)	38.0	431	1.55E+03	3.76E+04
DSTEDC (CPU)	2.32	38.7	167	450
DSTEDC (MIC)	3.24	26.7	101	241

gorithm. For further comparison, we add the QR [9] and Divide and Conquer algorithms [11], which are well-known methods. Because QR algorithm [9] and Divide and Conquer algorithm [11] are not able to compute partial eigenvalue and eigenvectors of target matrix. The implementations of these three algorithms are provided in Intel Math Kernel Library (MKL) [10]. In these experiments, we use DSTEMR, DSTEQR, and DSTEDC LAPACK routines provided by Intel MKL as the parallel implementations of the MRRR, QR, and Divide and Conquer algorithms, respectively. We note that the number of threads in all numerical experiments is set to be the number of processor cores.

Figure 5 shows the execution time for obtaining all eigenvalues and eigenvectors using each algorithm for $T_1 (n = 10000, \ldots, 70000)$. It should be noted that the graph is semilogarithmic. The Divide and Conquer algorithm is the fastest. However, it cannot be adopted for partial eigenvalue and eigenvectors of the target matrix. PBi+BIR achieves higher performance than MRRR in MIC environment, as the graph shows.

The results for matrix T_2 are shown in Figure 6. PBi+BIR in MIC environment is faster than MRRR in CPU environment for $n \geq 60000$. Unlike the other algorithms, Pbi+BIR has smaller computation time gain in MIC environment than in CPU environment, both absolutely and asymptotically. The asymptotic behavior suggests that the PBi+BIR algorithm is suitable for MIC environment.

Moreover, evaluations for accuracy are performed. Figures 7 and 8 show the orthogonality $\|Q^{\mathsf{T}}Q - I\|_{\mathrm{F}}$ of egeinvectors

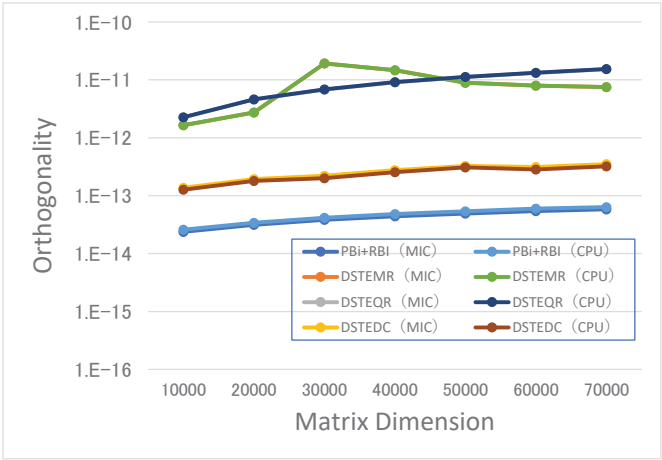

Fig. 7: Orthogonality of obtained eigenvectors for T_1

obtained by each eigensolver for T_1 and T_2, respectively (with $n = 10000, \ldots, 70000$). n denotes the dimension of the target matrix, $D = \mathrm{diag}\{\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n\}$, and $Q = [\boldsymbol{q}_1 \cdots \boldsymbol{q}_n]$. The graphs are semilogarithmic. The lines of PBi+BIR (MIC), DSTEMR(MIC), DSTEQR(MIC) and DSTEDC (MIC) are hidden behind the corresponding lines in CPU environment, since the results are nearly equal. The orthogonality among the eigenvectors obtained by the PBi+BIR algorithm is significantly smaller than that obtained by the other algorithms.

Figures 9 and 10 show the results of the evaluation of decomposition accuracy, namely, the residuals $\|TQ - QD\|_{\mathrm{F}}$ of the decomposed eigenvalues and eigenvectors for T_1 and T_2,

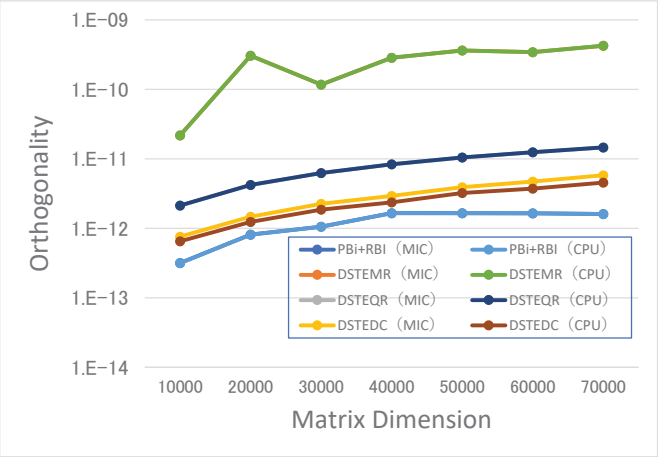

Fig. 8: Orthogonality of obtained eigenvectors for T_2

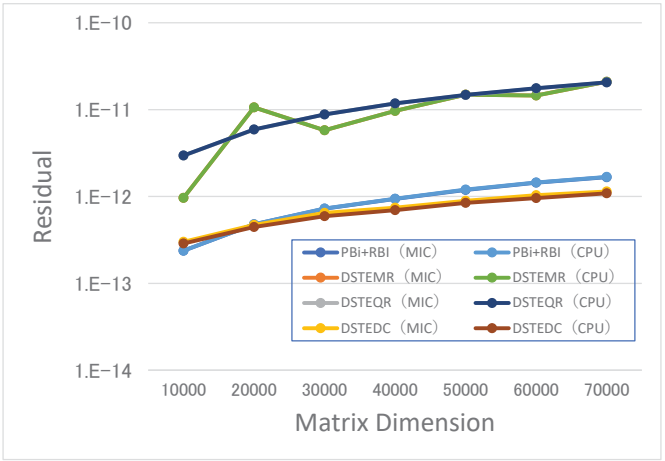

Fig. 10: Residuals of decomposition for T_2

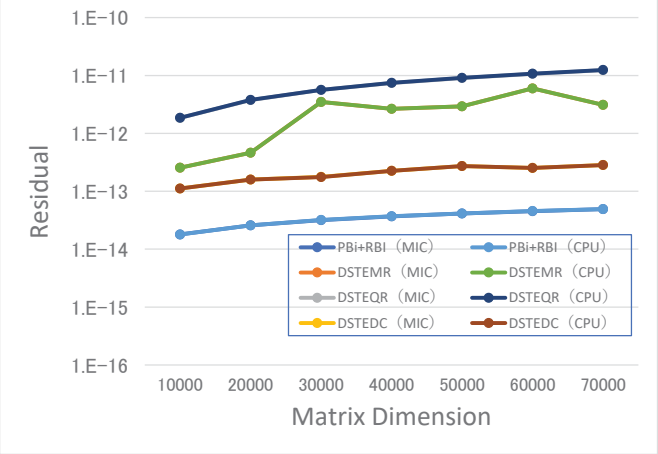

Fig. 9: Residuals of decomposition for T_1

respectively. PBi+BIR achieves higher accuracy than MRRR in terms of decomposition. In conclusion, in terms of both speed and accuracy, the PBi+BIR algorithm is superior to the MRRR algorithm in MIC environment. The high accuracy of PBi+BIR may be attributed to the high relative accuracy of the PBi algorithm.

5. Conclusion

We evaluated the performance of an eigensolver using a combination of the parallel bisection method and the block inverse iteration method with reorthogonalization (PBi+BIRalgorithm) for computing all eigenpairs of a real symmetric tridiagonal matrix. In particular, we compared the performance of PBi+BIR with that of MRRR, which can be adopted for obtaining a subset of eigenpairs. From the preliminary experiments, we determined the optimal block size for the block inverse iteration method with reorthogonalization. We measured the computation time of PBi+BIR, for each block of size $r = 2^m \times C$, where C is the number of processor cores, and let the block size yielding the minimum computation time be the optimal.

For T_1, PBi+BIR with the optimal block size parameter obtains the eigenpairs faster than MRRR in MIC environment. This is true for T_2 as well, with the condition that size of the target matrix is $n \geq 60000$. In addition, PBi+BIR is the only algorithm computing eigenpairs faster in MIC environment than in CPU environment. It is conceivable the AVX512 instruction set contributes to the superior performance of PBi+BIR in MIC environment, as this algorithm involves a large number of matrix multiplications. This suggests that PBi+BIR is suitable for MIC environment. Moreover, in a comparison of orthogonality of eigenvectors and residuals of eigenpairs, PBi+BIR achieves higher accuracy than MRRR, and there is no trade-off between computation speed and accuracy in MIC environment.

There are many applications of eigensolvers for real symmetric matrices such as kernel principal component analysis, which frequently appears in the industrial field. Fast and accurate eigensolvers are in great demand in this field, and PBi+BIR in MIC environment may be the eigensolver of choice. In future work, we would perform a more detailed examination of the relation between computation time and block size, and establish a method for auto-tuning of the algorithm in MIC environment.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 17H02858.

References

[1] J. Wilkinson, "Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection", *Numer. Math.*, vol. 4, no. 1, pp. 362 367, 1962.

[2] I. S. Dhillon, B. N. Parlett, and C. Vömel, "The design and implementation of the MRRR algorithm", *ACM Trans. Math. Softw.*, vol. 32, no. 4, pp. 533–560, 2006.

[3] G. H. Golub and C. F. van Loan, *Matrix Computations*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[4] J. W. Demmel, *Applied Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, 1997.

[5] B. N. Parlett, *The Symmetric Eigenvalue Problem*. Philadelphia, PA, USA: SIAM, 1998.

[6] H. Ishigami, K. Kimura, Y. Nakamura, "A New Parallel Symmetric Tridiagonal Eigensolver Based on Bisection and Inverse Iteration Algorithms for Shared-memory Multi-core Processors", *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp. 216–213, 2015.

[7] L. Blackford, J. Choi, A. Cleary, E. DAzevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK Users Guide*. Philadelphia, PA, USA: SIAM, 1997.

[8] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. W. Demmel, J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users Guide (Third ed.)*. Philadelphia, PA, USA: SIAM, 1999.

[9] W. Kahan, "Accurate eigenvalues of a symmetric tridiagonal matrix", *Technical Report*, Computer Science Dept. Stanford University, no. CS41, 1966.

[10] Intel Math Kernel Library, "Available electronically at https://software.intel.com/en-us/intel-mkl/."

[11] M. Gu and S. C. Eisenstat, "A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem", *SIAM J. Matrix Anal. Appl.*, vol. 16, no. 1, pp. 172–191, 1995.