IPSJ Transactions on Advanced Computing Systems Vol. 1 No. 3 14–27 (Dec. 2008)

Regular Paper

A Fully Pipelined Multishift QR Algorithm for Parallel Solution of Symmetric Tridiagonal Eigenproblems

Takafumi Miyata,[†1] Yusaku Yamamoto[†1] and Shao-Liang Zhang[†1]

In this paper, we propose a fully pipelined multishift QR algorithm to compute all the eigenvalues of a symmetric tridiagonal matrix on parallel machines. Existing approaches for parallelizing the tridiagonal QR algorithm, such as the conventional multishift QR algorithm and the deferred shift QR algorithm, have suffered from either inefficiency of processor utilization or deterioration of convergence properties. In contrast, our algorithm realizes both efficient processor utilization and improved convergence properties at the same time by adopting a new shifting strategy. Numerical experiments on a shared memory parallel machine (Fujitsu PrimePower HPC2500) with 32 processors show that our algorithm is up to 1.9 times faster than the conventional multishift algorithm and up to 1.7 times faster than the deferred shift algorithm.

1. Introduction

In numerical linear algebra and its applications, computing the eigenvalues of a real symmetric matrix is one of the most important problems. Recently, eigenvalues of matrices of the order of more than 100,000 need to be computed in the analysis of protein structures [8),12)]. To solve such large problems, parallel computing is one of the most promising approaches. In particular, the use of shared memory parallel machines has become increasingly popular with the advent of multicore processors. To attain high performance on such machines, we need an efficient parallel algorithm that can fully utilize all the processors, while requiring small parallelization cost, that is, a small number of inter-processor synchronizations [19)].

In the standard procedure to solve a dense symmetric eigenproblem, we first reduce the matrix to tridiagonal form by the Householder algorithm and then compute the eigenvalues of the tridiagonal matrix. There are several approaches for the second step, such as the bisection algorithm [20)], the single-shift QR algorithm [7),15)], and the divide & conquer algorithm [4),10)]. Among them, the single-shift QR algorithm has a long history and is widely used as an efficient and reliable algorithm. When the matrix order is n, the first step needs about $4n^3/3$ floating point operations, while the second step needs about $O(n^2)$ operations. However, the first step can be parallelized efficiently using a large number of processors [3),6),13)]. As a result, the relative computational time of the second step increases with the number of processors. We therefore focus on the second step. The purpose of this paper is to speed up the single-shift QR algorithm for the tridiagonal matrix by parallel computing.

The single-shift QR algorithm computes the eigenvalues by applying a series of orthogonal transformations to the input tridiagonal matrix. It introduces an origin shift to speed up the convergence. As a generalization of this, Bai et al. proposed the multishift QR (M-QR) algorithm [2)], which introduces m shifts and performs m steps of the single-shift QR algorithm at once. The operations in one step of the single-shift QR algorithm, known as bulge-chasing, is inherently sequential and is difficult to parallelize. On the other hand, in the M-QR algorithm, m bulge chasing operations can be done in a pipelined fashion and we can parallelize the algorithm by allocating one processor to each bulge.

To accelerate the convergence of the single-shift QR algorithm, we need to set the shift as close to an eigenvalue as possible. To this end, we update the shift after each QR step because the shift computed from a newer iteration gives better approximation to the eigenvalue. In the M-QR algorithm we update m shifts at the end of each multishift QR step, that is, after m bulge-chasings have finished. However, in the M-QR algorithm, the m bulge-chasing operations are introduced one by one, so m iterations don't finish all at once. Consequently, the processor that finishes the bulge-chasing first idles until the last processor finishes the work. Thus, the M-QR algorithm cannot make all processors busy.

To fully utilize the processors, it is necessary for a processor that has finished the bulge-chasing to start the next iteration without waiting for the completion of other processors' work. This is possible if we use older shifts, for example,

†1 Department of Computational Science & Engineering, Graduate School of Engineering, Nagoya University

shifts computed at the end of the multishift QR step that is two steps before the current one. This is known as the deferred shift QR (D-QR) algorithm [16] and allows us to make processors busy all the time. However, this algorithm shows slower convergence than the M-QR algorithm because the shifts in this algorithm are computed from an older iteration and therefore do not approximate the eigenvalues so well as the shifts in the M-QR algorithm.

In this paper, we propose a fully pipelined multishift QR (FPM-QR) algorithm to improve the convergence property of the the D-QR algorithm while keeping the level of processor utilization. To attain this, we reorganize the algorithm so that all the processors, not only the m-th processor as in the existing algorithms, update a shift after each bulge-chasing and start the next step with the new shift as soon as possible. In this way, we can make all the processors fully operative. In addition, the FPM-QR algorithm is expected to show better convergence than the D-QR algorithm since the shifts used in the FPM-QR algorithm contain newer information. Although the work for shift computation increases by m times, we can hide the increase by rebalancing the load among the processors. A brief summary of the FPM-QR algorithm is given in a survey 22), although without any numerical or performance results.

This paper is structured as follows: in Section 2, we give a brief explanation about the single-shift QR algorithm, the M-QR algorithm, and the D-QR algorithm. Section 3 gives the details of our new algorithm: the FPM-QR algorithm. Experimental results on a shared memory parallel machine are presented in Section 4. Finally, Section 5 gives some concluding remarks.

2. The Tridiagonal QR Algorithm and Its Parallelization: Existing Algorithms

We treat the following standard eigenvalue problem:

$$T\boldsymbol{x} = \lambda\boldsymbol{x}, \tag{1}$$

where T is an $n \times n$ real symmetric tridiagonal matrix. Our aim is to get all eigenvalues $\{\lambda_i\}_{i=1}^n$ of T quickly. In this section, we give a brief review of eigenvalue computation by the single-shift QR algorithm and its parallelization.

2.1 The Single-shift QR Algorithm

Given a real tridiagonal matrix T, the single-shift QR algorithm produces a series of matrices that is orthogonally similar to T using the orthogonal matrix Q given by the QR decomposition of the matrix. The algorithm is shown in Algorithm 1. Iterations of this algorithm make T converge to a diagonal matrix under suitable assumptions [9]. By introducing the origin shift s_i, which is an approximation of an eigenvalue of T, the convergence can be accelerated. Let *Shift* (m, T_{i+1}) be the set of the eigenvalues of the $m \times m$ trailing submatrix of T_{i+1}. The Wilkinson shift is the one which is closer to the (n, n) element of T_{i+1} in *Shift* $(2, T_{i+1})$. With this shift, the single-shift QR algorithm has the property of global convergence and the asymptotic convergence rate is usually cubic [21].

Algorithm 1 The single-shift QR algorithm

$T_1 = T$
Computation of *Shift* $(2, T_1)$
$s_1 \leftarrow$ the shift which is closer to the (n, n) element of T_1
for $i = 1, 2, \ldots$ **do**
  $T_i - s_i I \rightarrow Q_i R_i$
  $T_{i+1} \leftarrow R_i Q_i + s_i I \ \left(= Q_i^T T_i Q_i\right)$
  if any subdiagonal element e is close to zero **then** ▷ Convergence check
    set $e = 0$ to obtain $T_{i+1} = \begin{pmatrix} T_A & 0 \\ 0 & T_B \end{pmatrix}$
    apply the single-shift QR algorithm to T_B and $T_{i+1} \leftarrow T_A$
  end if
  Computation of *Shift* $(2, T_{i+1})$
  $s_{i+1} \leftarrow$ the shift which is closer to the (n, n) element of T_{i+1} ▷ Update of shift
end for

In a practical implementation, the effect of the shift is introduced implicitly [9], and one step of the single-shift QR algorithm is composed of $n - 1$ times of similarity transformations. This operation is called bulge-chasing. A bulge is set at the upper left corner of the matrix at the first transformation and successive transformations chase the bulge down by one row at a time until the tridiagonal form is finally recovered. The $k + 1$ th transformation cannot be done until the k-th one finishes, so the bulge-chasing is a sequential procedure. After the i-th

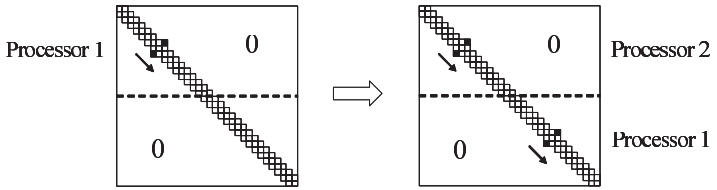

Fig. 1 Pipelined bulge-chasing in the M-QR algorithm ($m = 2$, $M = 2$).

QR step, the shift is updated to speed up the convergence. The new shift s_{i+1} is used in the next $i+1$ th step, so this algorithm cannot start the step $i+1$ until the new shift is prepared at the end of the step i. Such a sequential procedure makes it difficult to parallelize the single-shift QR algorithm.

2.2 The M-QR Algorithm

2.2.1 The Algorithm and Its Parallelization

We can generalize the single-shift QR algorithm to perform m steps at once using m shifts. This is called the M-QR algorithm [2] and is shown in Algorithm 2. Here, $T_{i,j}$ denotes the tridiagonal matrix in the i-th multishift QR step right before the j-th shift is applied, $s_{i,j}$ is the j-th shift in the i-th multishift step. The m shifts introduced in the step $i+1$ are the m eigenvalues of the $m \times m$ trailing submatrix of $T_{i,m+1}$, that is $\{s_{i+1,j}\}_{j=1}^{m} = \mathit{Shift}\ (m, T_{i,m+1})$. For the computation of $\mathit{Shift}\ (m, T_{i,m+1})$, the single-shift QR algorithm can be used.

As the iteration proceeds, the subdiagonal elements become smaller, and finally one of them can be regarded as zero. At this point, this element is set to zero and the matrix is split into two submatrices T_A and T_B. If the size of T_B is less than or equal to m, the eigenvalues of T_B are computed with the single-shift QR algorithm and the M-QR algorithm proceeds with T_A as a new input matrix. This deflation procedure is used also by the D-QR and FPM-QR algorithms to be explained later.

The multishift scheme enables us to parallelize the original algorithm by chasing more than one bulge at once. Let the number of processors equal that of the shifts and assume that each processor chases one bulge. Then we can parallelize one step of the M-QR algorithm as follows. Assume that the matrix is divided into M regions horizontally. Here, $M \geq m$. The simple case of $m = 2$ and $M = 2$ is shown in **Fig. 1**. The first bulge is introduced at the top left corner and chased by processor 1. When it passes through the first region, the second bulge is introduced and chased by processor 2. Each chasing of bulge is independent and can be executed in a pipelined fashion [17], but the bulges have to be kept at least three rows apart to avoid conflict. By making sure that there is only one bulge in one region, no conflict occurs. This can be guaranteed by inserting an inter-processor synchronization when a bulge passes the boundary of the regions.

In this paper, we assume the number of processors equals that of the shifts. Namely, one processor moves one bulge.

Algorithm 2 The M-QR algorithm

$T_{1,1} = T$
Computation of $\mathit{Shift}\ (m, T_{1,1})$
for $j = 1, 2, \ldots, m$ **do**
  $s_{1,j} \leftarrow$ the j-th smallest shift
end for
for $i = 1, 2, \ldots$ **do**
  for $j = 1, 2, \ldots, m$ **do**
    $T_{i,j} - s_{i,j} I \rightarrow Q_{i,j} R_{i,j}$
    $T_{i,j+1} \leftarrow R_{i,j} Q_{i,j} + s_{i,j} I \ \ (= Q_{i,j}^T T_{i,j} Q_{i,j})$
    if any subdiagonal element e is close to zero **then** ▷ Convergence check
      set $e = 0$ to obtain $T_{i,j+1} = \begin{pmatrix} T_A & 0 \\ 0 & T_B \end{pmatrix}$
      if the order of T_B is less than or equal to m **then**
        apply the single-shift QR algorithm to T_B and $T_{i,j+1} \leftarrow T_A$
      end if
    end if
  end for
  Computation of $\mathit{Shift}\ (m, T_{i,m+1})$
  for $j = 1, 2, \ldots, m$ **do** ▷ Update of m shifts
    $s_{i+1,j} \leftarrow$ the j-th smallest shift
  end for
  $T_{i+1,1} = T_{i,m+1}$
end for

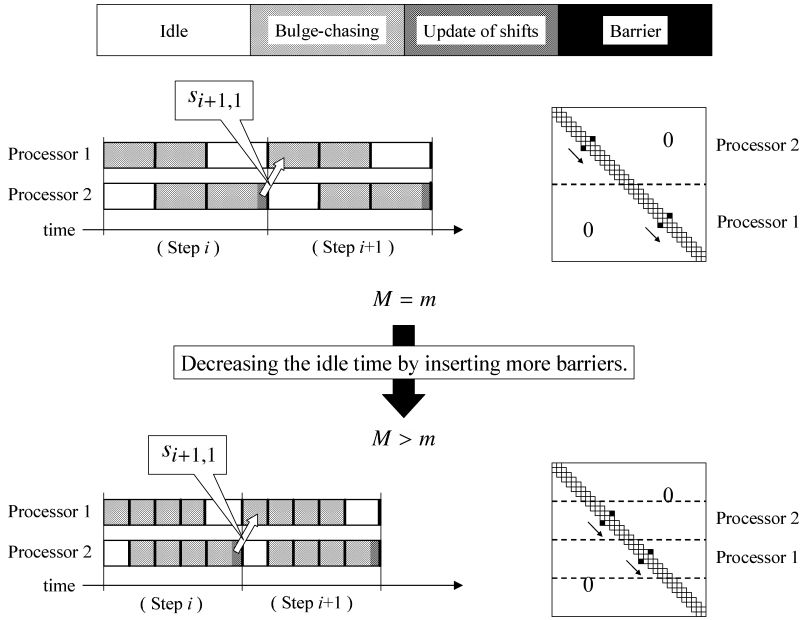

Fig. 2 Computational sequence of the M-QR algorithm ($m = 2$). Upper: $M = 2$. Lower: $M = 4$.

2.2.2 The Trade-off between Processor Idle Time and Synchronization Cost

The processor j which has completed one sweep of the j-th bulge becomes idle until other $m - j$ bulge-chasing is finished since only then can the shifts used in the next step be computed. In addition, the processor j is forced to wait to start the next step until the foregoing $j - 1$ bulges pass through the first region of the matrix. These idle times occur at every step in the M-QR algorithm.

The idle time of processors can be decreased by shortening the distance between the bulges. To attain this, we need to divide the matrix into more and more regions, see **Fig. 2**. But this also increases the number of synchronizations, so there is a trade-off between the efficiency of processor utilization and the cost of synchronization. Under computational environments with low or no cost of synchronization, such as the vector processors or SIMD machines, the idle time of processors can be minimized by choosing the largest possible division number [14]. So the M-QR algorithm with this choice of parameter can achieve the optimal performance on vector processors or SIMD machines. But this is not the case for most environments which have considerable synchronization costs.

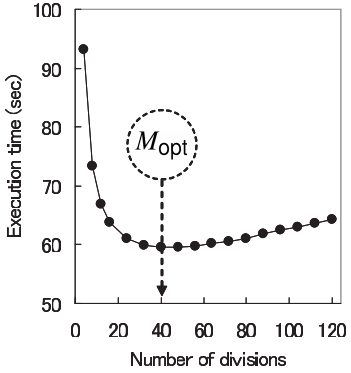

Fig. 3 Effect of the division number of the matrix on the execution time of the M-QR algorithm ($m = 4$). Details of the testing matrix (Type 1, $n = 50000$) and the computational environment are described in Section 4.

2.2.3 The Optimal Division Number

To maximize the performance of the M-QR algorithm, it is necessary to choose the division number appropriately for each computational environment. By modeling the execution time of the M-QR algorithm, we have derived the optimal division number M_{opt} as follows:

$$M_{\mathrm{opt}} = \sqrt{\frac{n\,(m-1)}{4}\left(\frac{t_{\mathrm{bulge}}}{t_{\mathrm{sync}}}\right)}, \tag{2}$$

where n is the order of the matrix, m is the number of shifts, t_{bulge} is the time of moving a bulge down by one row, and t_{sync} is the time for one synchronization. The last two parameters are dependent on the computational environment. The derivation of Eq. (2) is given in Section A.1. Eq. (2) shows that an environment with low cost of synchronization relative to the computational cost allows us to divide the matrix into more regions than standard m areas.

Figure 3 shows an experimental result that illustrates the effect of the divi-

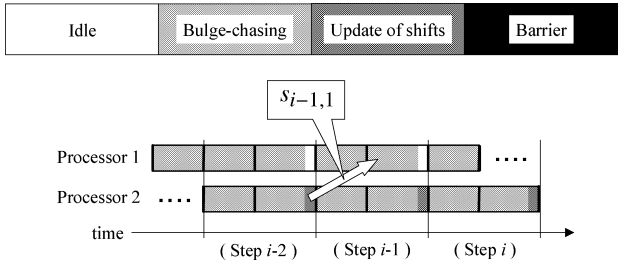

Fig. 4 Computational sequence of the D-QR algorithm ($m = 2$).

sion number M on the parallel performance. The optimal division number $M_{\rm opt}$ predicted by Eq. (2) is 40 in this situation. As the number of regions increases, the execution time decreases rapidly due to the reduction in the idle time, but dividing into too many regions increases the execution time due to large synchronization costs. As shown in the graph, the optimal division number obtained from our model actually achieves near-optimal performance. However, note that the performance in Fig. 3 is a compromise between reducing the idle time and reducing the synchronization cost. If we can eliminate the idle time, it may be possible to further improve the performance. We will pursue this possibility in the following sections.

2.3 The D-QR Algorithm

The D-QR algorithm, or the M-QR algorithm with deferred shifts [16], has an advantage over the basic M-QR algorithm in terms of efficiency of the use of processors. This algorithm is given by Algorithm 3. In the M-QR algorithm, the shifts $\{s_{i,j}\}_{j=1}^{m}$ used in the i-th step are updated after finishing all bulge-chasings at the step $i-1$, so the idle time of processors emerges at each step. In contrast, the step i of the D-QR algorithm uses shifts $\{s_{i-1,j}\}_{j=1}^{m}$, which are updated after all bulge-chasings at the step $i-2$ have finished. This algorithm therefore makes it possible to start the step i before finishing all the sweeps of bulges at the step $i-1$, because the shifts are already available, see **Fig. 4**.

It has been observed that the use of older shifts in the D-QR algorithm increases the number of iterations before convergence. This is because these shifts do not contain the effects of the $i-1$ th multishift QR step and are therefore not such a good approximation to the eigenvalues as the shifts used in the M-QR algorithm. The local convergence rate of the D-QR algorithm is shown to be quadratic [16], while that of the M-QR algorithm is shown to be cubic [18]. It may be possible to improve the performance of the parallel QR algorithm if we modify the D-QR algorithm to use the shifts that contain new information as much as possible, while keeping the pipeline fully operative.

Algorithm 3 The D-QR algorithm

$T_{1,1} = T$
Computation of *Shift* $(m, T_{1,1})$
for $j = 1, 2, \ldots, m$ **do**
  $s_{0,j} = s_{1,j} \leftarrow$ the j-th smallest shift
end for
for $i = 1, 2, \ldots$ **do**
  for $j = 1, 2, \ldots, m$ **do**
    $T_{i,j} - \boxed{s_{i-1,j}}\, I \rightarrow Q_{i,j} R_{i,j}$
    $T_{i,j+1} \leftarrow R_{i,j} Q_{i,j} + \boxed{s_{i-1,j}}\, I \ \left(= Q_{i,j}^T\, T_{i,j}\, Q_{i,j}\right)$
    if any subdiagonal element e is close to zero **then** ▷ Convergence check
      set $e = 0$ to obtain $T_{i,j+1} = \begin{pmatrix} T_A & 0 \\ 0 & T_B \end{pmatrix}$
      if the order of T_B is less than or equal to m **then**
        apply the single-shift QR algorithm to T_B and $T_{i,j+1} \leftarrow T_A$
      end if
    end if
  end for
  Computation of *Shift* $(m, T_{i,m+1})$
  for $j = 1, 2, \ldots, m$ **do** ▷ Update of m shifts
    $s_{i+1,j} \leftarrow$ the j-th smallest shift
  end for
  $T_{i+1,1} = T_{i,m+1}$
end for

The optimal division number of the matrix for the D-QR algorithm is equal to the number of shifts ($M_{\rm opt} = m$) because the idle time of the processors arises

only at the first multishift QR step when m bulges are introduced step by step, and this time is negligible. So there is no benefit in dividing the matrix into more than standard m regions.

3. The New FPM-QR Algorithm

3.1 The Algorithm

We propose the FPM-QR algorithm to improve the convergence property of the D-QR algorithm while keeping the efficiency of using processors. This algorithm is shown in Algorithm 4.

Algorithm 4 The FPM-QR algorithm

$T_{1,1} = T$
Computation of *Shift* $(m, T_{1,1})$
for $j = 1, 2, \ldots, m$ **do**
  $s_{1,j} \leftarrow$ the j-th smallest shift
end for
for $i = 1, 2, \ldots$ **do**
  for $j = 1, 2, \ldots, m$ **do**
    $T_{i,j} - s_{i,j} I \rightarrow Q_{i,j} R_{i,j}$
    $T_{i,j+1} \leftarrow R_{i,j} Q_{i,j} + s_{i,j} I \ \left(= Q_{i,j}^T T_{i,j} Q_{i,j}\right)$
    if any subdiagonal element e is close to zero **then** ▷ Convergence check
      set $e = 0$ to obtain $T_{i,j+1} = \begin{pmatrix} T_A & 0 \\ 0 & T_B \end{pmatrix}$
      if the order of T_B is less than or equal to m **then**
        apply the single-shift QR algorithm to T_B and $T_{i,j+1} \leftarrow T_A$
      end if
    end if
    Computation of *Shift* $(m, T_{i,j+1})$
    $s_{i+1,j} \leftarrow$ the j-th smallest shift ▷ Update of only one shift
  end for
  $T_{i+1,1} = T_{i,m+1}$
end for

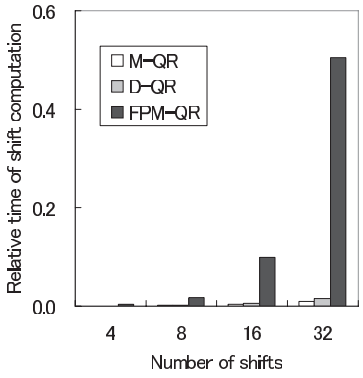

Fig. 5 Relative time of shift computation (the ratio of shift computation time to total execution time) of the M-QR, D-QR, and FPM-QR algorithm. Details of the testing matrix (Type 1, $n = 50000$) and the computational environment is described in Section 4.

In contrast to the M-QR and D-QR algorithms, which update the shifts after m sweeps of bulges have finished, our algorithm updates the shifts after each sweep of a bulge. To keep the pipeline fully operative and to introduce a new shift as soon as possible, a processor finishing a bulge sweep updates a shift without waiting for other bulges' arrival. Then, using the shift, the processor starts the next step.

In the FPM-QR algorithm, the processor j uses $T_{i,j+1}$ to compute the new shift $s_{i+1,j}$, and chooses the j-th smallest shift within *Shift* $(m, T_{i,j+1})$ as $s_{i+1,j}$. In contrast, the M-QR algorithm uses $T_{i,m+1}$, while the D-QR algorithm uses $T_{i-1,m+1}$. Thus the FPM-QR algorithm uses better shifts than the D-QR algorithm. The M-QR algorithm uses even better shifts, but the FPM-QR algorithm has an advantage in terms of the efficiency of running processors.

3.2 Hiding the Cost of Shift Update

In the FPM-QR algorithm, each processor updates a shift after finishing its own bulge-chasing. This means that the computation of shifts is needed m times per each multishift QR step, while the existing algorithms need this only once. As a result, as shown in **Fig. 5**, the shift computation time of the FPM-QR algorithm with 32 shifts accounts for about 50% of the total execution time while that of the existing algorithms is less than 10%. However, this overhead can be reduced

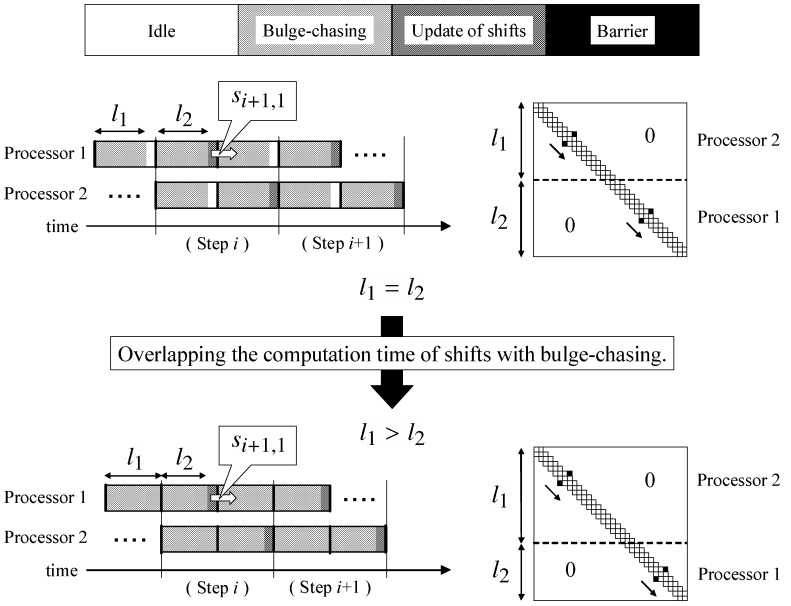

Fig. 6 Computational sequence of the FPM-QR algorithm ($m = 2$). Upper: the processor idle time emerges if the region sizes are equal. Lower: all the processors become fully operative after load-rebalancing.

as follows.

In the FPM-QR algorithm, the processor working in the undermost region updates a shift after finishing bulge-chasing, while other processors do only bulge-chasing work. As a result, the former processor has an extra task and other processors are forced to wait until the processor completes shift computation. To reduce this overhead, we reduce the size of the undermost region and make the load-balance equal among the processors.

We show the computational sequence of the FPM-QR algorithm in **Fig. 6**. If the sizes of the regions are equal (the upper figure), the processor 2 idles while the processor 1 updates a shift and the processor 1 idles while the processor 2 updates a shift. Thus the time of shift computation is two times per each multishift QR step. After the load-rebalance (the lower figure), the shift computation by one processor can be overlapped with the bulge-chasing by other processors. In this

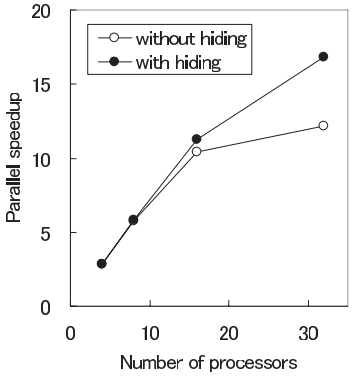

Fig. 7 Parallel speedup of the FPM-QR algorithm: T_1/T_m, where T_1 and T_m are the execution times with single processor and m processors, respectively. Details of the testing matrix (Type 1, $n = 50000$) and the computational environment are described in Section 4.

way, the effective shift computation is one time per each multishift QR step like existing algorithms. In general, to hide $m-1$ times of shift computation, the region size is assigned as follows. Let the time of computing m shifts be $t_{\mathrm{shift},m}$, the time of moving a bulge down by one row be t_{bulge}, and $t_{\mathrm{shift},m} = \Delta \cdot t_{\mathrm{bulge}}$, which means that the computation of shifts is equivalent to bulge sweeps by Δ rows. The number of rows in the i-th region from the top, l_i, is given as follows: $l_1 = l_2 = \cdots = l_{m-1} = (n + \Delta)/m$ and for the undermost region $l_m = (n - (m-1) \cdot \Delta))/m$.

In a practical implementation, the row length Δ is determined before running the program from the measured values of t_{bulge} and $t_{\mathrm{shift},m}$ on the target computational environment. To measure these values, we use the Type 1 matrix described in Section 4. The measured values of t_{bulge} and $t_{\mathrm{shift},m}$ for the computational environment used in Section 4 are shown in Table 3 in Section A.1.

We show the effect of this modification for a matrix of the order of $n = 50,000$ in **Fig. 7**. The effect is not prominent when the number of shifts m is small. However, when m is increased to 16 and 32, the modified FPM-QR algorithm achieves 10% and 40% greater parallel speedup than the original algorithm, respectively. This is because the ratio of the shift computation time to the total computation time increases as more shifts are used, as shown in Fig. 5. Note that

to take full advantage of the hiding of shift computation, we need the matrix size to be greater than $m\Delta$ for most of the computation time. Considering the effect of deflation, this condition amounts to $n \geq cm\Delta$, where c is a small constant, say, 2. However, even if this condition is not satisfied, we can hide part of the shift computation and obtain some performance gains. In fact, in the above example, when $m = 32$, we have $\Delta = t_{\text{bulge}}/t_{\text{shift},m} = 1310$ from Table 3. Thus $m\Delta = 32 \times 1310 = 41920$ in this case and $n = 50000$ is not sufficiently large. Still, we succeeded in obtaining considerable performance gains as shown above.

Like the D-QR algorithm, the FPM-QR algorithm allows us to fully utilize the processors. This means that the optimal division number for the FPM-QR algorithm equals the number of shifts; $M_{\text{opt}} = m$.

A simplified pseudocode of the FPM-QR algorithm is shown in Section A.2.

4. Numerical Results

In this section, we compare the proposed algorithm with the existing algorithms by numerical experiments in terms of (1) accuracy of the computed eigenvalues, (2) execution time, (3) parallel speedup, and (4) convergence property. We computed all eigenvalues $\{\lambda_i\}_{i=1}^n$ of eight kinds of eigenproblems:

Type 1 A tridiagonal matrix whose diagonal elements are all equal to a, subdiagonal elements are all equal to b. The exact eigenvalues are $\lambda_i = a + 2b\cos(i\pi/(n+1))$ $(i = 1, \ldots, n)$. (We set $a = 2$ and $b = -1$.)

Type 2 A symmetric matrix whose elements are random numbers in (-0.5, 0.5).

Type 3 A coefficient matrix of the Laplace equation in two dimensions.

Type 4 A symmetric matrix "PARSEC/CO" obtained from University of Florida Sparse Matrix Collection [5].

Type 5 A tridiagonal matrix with eigenvalues $\lambda_i = \sinh(10i/n)$ $(i = 1, \ldots, n)$.

Type 6 A tridiagonal matrix with eigenvalues $\lambda_i = \sinh(-5 + 10i/n)$ $(i = 1, \ldots, n)$.

Type 7 A tridiagonal matrix with eigenvalues $\lambda_i = \sinh(-10 + 10i/n)$ $(i = 1, \ldots, n)$.

Type 8 A tridiagonal matrix with eigenvalues $\lambda_i = \tanh(-5 + 10i/n)$ $(i = 1, \ldots, n)$.

The tridiagonal matrices of Types 5-8 were generated to have the specified eigenvalues by DLATMS in LAPACK [1]. The symmetric matrices of Type 2-4 are transformed to tridiagonal matrices by Householder transformations. The order of the matrices is $n = 50000$ and 200000 for Type 1, 2, and 5-8, $n = 99856$ and 200704 for Type 3, and $n = 221119$ for Type 4.

Our computational environment is Fujitsu PRIMEPOWER HPC2500 (CPU: SPARC 64V 8 Gflops × 32 processors, Memory: 512 GB). We wrote three codes with C and OpenMP [11] for shared memory parallel machines: (1) The M-QR algorithm (with M_{opt}, see Section 2.2.3), (2) The D-QR algorithm, (3) The FPM-QR algorithm, and these codes were compiled by Fujitsu C Compiler with options `-Kfast_GP2=2 -KOMP`. To check the convergence, we compare the subdiagonal element e_k $(1 \leq k \leq n-1)$ with the neighbor diagonal elements and set it to zero when $|e_k| \leq \varepsilon\,(|d_k| + |d_{k+1}|)$ $(1 \leq k \leq n-1)$, where ε is the round-off unit. We can check the convergence every time we finish one of the bulge sweeps. We therefore check the convergence m times in one multishift QR step with m shifts to catch the opportunity of deflation as soon as possible. Such a vigilant deflation strategy [17] is used in all the three algorithms.

4.1 Accuracy of Computed Eigenvalues

We evaluate the accuracy of eigenvalues computed by the three eigensolvers. The metric is the error relative to the spectral radius: $\max_i \left|\lambda_i - \hat{\lambda}_i\right| \,/\, \max_j \left|\hat{\lambda}_j\right|$, where λ_i is the eigenvalue computed by the multishift algorithm (M-QR, D-QR, or FPM-QR) and $\hat{\lambda}_i$ is the exact eigenvalue (for the Type 1, 5-8 matrices) or eigenvalue computed by DSTEQR in LAPACK (for other matrices).

We obtained similar results for all the problems, so we show the results for random matrices (Type2) in **Table 1**. From Table 1, we can observe that every algorithm shows similar accuracy within the relative error of 10^{-11}.

4.2 Convergence Property

We define the relative iteration number as a metric for the convergence property: $i_{\text{avg},m} \,/\, i_{\text{avg},1}$, where $i_{\text{avg},m}$ and $i_{\text{avg},1}$ are the weighted average iteration number (see Eq. (4) in Section A.1) to get one eigenvalue by one of the multishift algorithms (M-QR, D-QR, or FPM-QR) with m shifts and the single-shift QR algorithm, respectively. The smaller value of this quantity means that the algorithm shows faster convergence.

Table 1 Accuracy of the computed eigenvalues: $\max_i \left|\lambda_i - \hat{\lambda_i}\right| \ / \ \max_j \left|\hat{\lambda_j}\right|$.

Type 2 ($n = 50000$)

m	M-QR	D-QR	FPM-QR
4	8.8E-13	1.1E-12	1.2E-12
8	9.0E-13	1.3E-12	1.3E-12
16	1.0E-12	1.2E-12	9.5E-13
32	1.1E-12	1.3E-12	8.4E-13

Type 2 ($n = 200000$)

m	M-QR	D-QR	FPM-QR
4	3.8E-12	4.7E-12	4.3E-12
8	4.0E-12	5.6E-12	5.1E-12
16	5.0E-12	5.5E-12	4.4E-12
32	4.5E-12	5.7E-12	3.9E-12

We obtained similar results for the matrices of Type 2, 3, 4, and 6, and similar results for the matrices of Type 5, 7, and 8. So we show the results for the matrices of Type 2, 5, and 1 in **Fig. 8**. As shown in Fig. 8, the relative iteration number of the FPM-QR algorithm is about 10% less than that of the D-QR algorithm for the Type 2 matrix. For the Type 5 matrix, the relative iteration number of the FPM-QR algorithm is about 20% less, and for the Type 1 matrix, more than 20% less. Thus we can say that the FPM-QR algorithm improves the convergence property of the D-QR algorithm for all the eight problems. However, the degree of improvement is dependent on the problem. The improvement is greater for matrices of Type 1, 5, 7, and 8.

To investigate the reason for this, we plotted the distribution of eigenvalues of eight kinds of eigenproblems in **Fig. 9**. As shown in Fig. 9, the smaller eigenvalues of the Type 5 matrix form a cluster. For the Type 7 matrix, the larger eigenvalues form a cluster. For the Type 1 and 8 matrices, both smaller and larger eigenvalues form a cluster. On the other hand, for the Type 2, 3, 4, and 6 matrices, there is no cluster at either end of the spectrum. From these observations, it can be said that the FPM-QR algorithm tends to show better convergence when either the top or the bottom eigenvalues are clustered. The theoretical explanation of this convergence behavior, as well as why the FPM-QR algorithm exhibits better convergence than the M-QR algorithm for the Type 1 matrix, is the subject of further research.

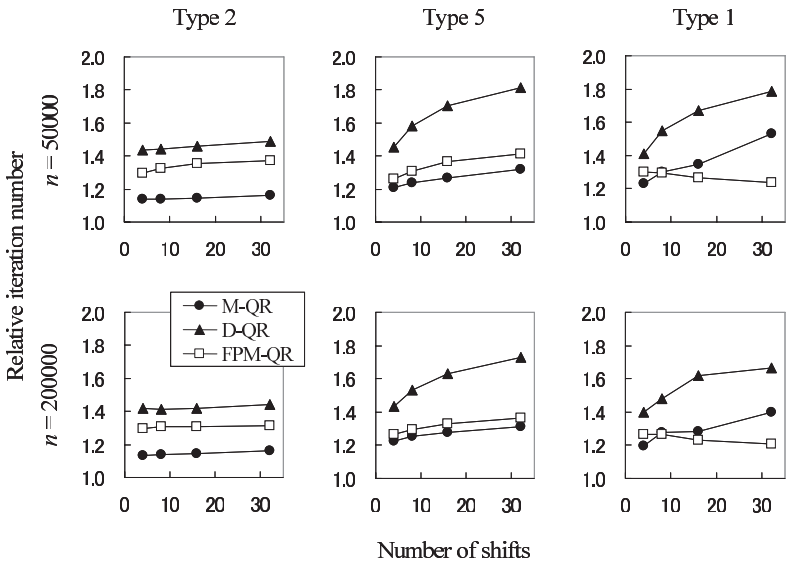

Fig. 8 Relative iteration number of the M-QR, D-QR, and FPM-QR algorithm.

4.3 Performance Results

We show the execution time and the parallel performance of the three eigensolvers. As a metric to measure the parallel performance, we used parallel speedup T_1/T_m, where T_m and T_1 are the execution times of the multishift algorithm with m processors and of the single-shift QR algorithm with one processor, respectively.

Like the results of the convergence property, we obtained similar results for the matrices of Type 2, 3, 4, and 6, and similar results for the matrices of Type 5, 7, and 8. So we show the execution time and the parallel speedup for the matrices of Type 2, 5, and 1 in **Table 2** and **Fig. 10**.

As shown in Fig. 10, the FPM-QR algorithm shows parallel speedup similar to that of the existing algorithms for the Type 2 matrix. For the Type 5 matrix, the FPM-QR algorithm shows about 1.2 times higher parallel speedup over the D-QR algorithm, which shows the second greatest parallel speedup. For the Type 1 matrix, the FPM-QR algorithm shows more than 1.4 times higher parallel

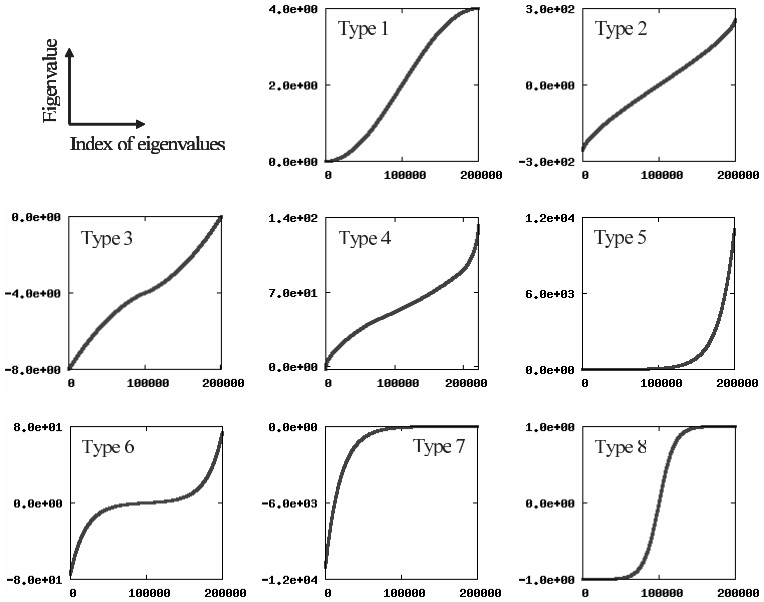

Fig. 9 Distribution of eigenvalues of eight kinds of eigenproblems. The eigenvalues are indexed in ascending order: $\lambda_1 < \lambda_2 < \cdots < \lambda_n$.

speedup.

- Comparison of the FPM-QR algorithm with the M-QR algorithm
  As shown in subsection 4.2, the average iteration number of the FPM-QR algorithm tends to be larger than that of the M-QR algorithm. But the FPM-QR algorithm shows greater parallel speedup than the M-QR algorithm because the FPM-QR algorithm allows us to use processors more efficiently. From Fig. 10, the FPM-QR algorithm with 32 shifts attains about 1.9 times greater parallel speedup for the Type 1 matrix ($n = 50000$) compared with the M-QR algorithm.
- Comparison of the FPM-QR algorithm with the D-QR algorithm
  The FPM-QR algorithm showed better convergence than the D-QR algorithm while keeping all the processors fully operative, so the FPM-QR algorithm shows greater parallel speedup than the D-QR algorithm for the sufficiently

Table 2 Execution time (sec.) of the M-QR, D-QR, and FPM-QR algorithm.

$n = 50000$

	Type 2			Type 5			Type 1		
m	M-QR	D-QR	FPM-QR	M-QR	D-QR	FPM-QR	M-QR	D-QR	FPM-QR
4	57.2	61.2	55.4	51.1	53.5	46.3	59.6	63.6	51.4
8	33.0	32.1	29.8	30.2	29.9	24.9	36.3	35.3	25.5
16	19.6	17.2	16.5	18.8	17.1	13.9	22.8	19.5	13.1
32	13.3	10.4	11.0	13.3	10.9	9.9	16.9	12.6	8.8

$n = 200000$

	Type 2			Type 5			Type 1		
m	M-QR	D-QR	FPM-QR	M-QR	D-QR	FPM-QR	M-QR	D-QR	FPM-QR
4	815.8	916.0	836.8	727.7	792.6	701.0	853.1	982.8	774.1
8	439.1	468.5	442.2	401.4	427.1	361.6	484.1	515.4	378.3
16	242.3	241.3	229.4	229.4	231.5	189.2	275.1	280.1	184.4
32	144.3	129.4	123.1	138.9	129.3	104.1	169.5	159.7	96.3

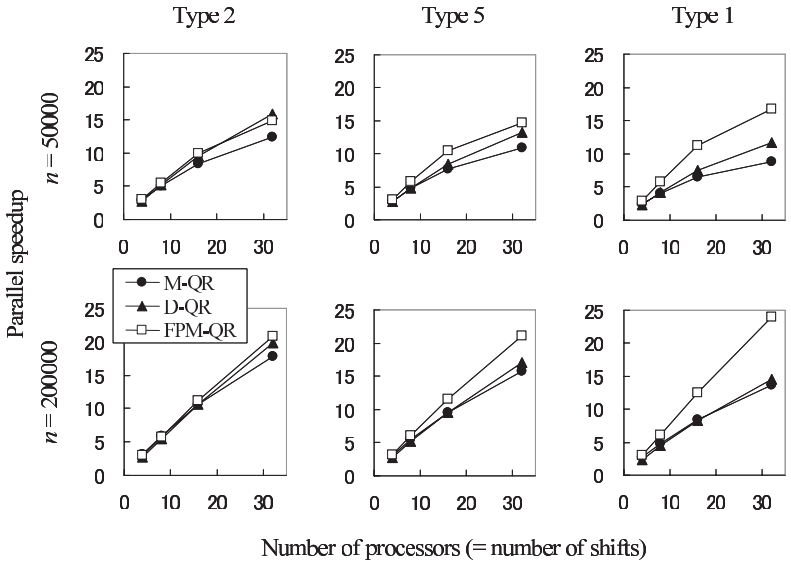

Fig. 10 Parallel speedup of the M-QR, D-QR, and FPM-QR algorithm.

large matrices ($n = 200000$). From Fig. 10, the FPM-QR algorithm with 32 shifts attains about 1.7 times greater parallel speedup for the Type 1 matrix ($n = 200000$) compared with the D-QR algorithm. For the small matrix (Type 2, $n = 50000$), the FPM-QR algorithm shows a little lower parallel speedup than the D-QR algorithm due to incomplete hiding of the shift computation time (see Section 3.2).

In summary, we have confirmed that the FPM-QR algorithm shows competitive or greater performance for these eight kinds of eigenproblems.

5. Conclusion

In this paper, we proposed the fully pipelined multishift QR (FPM-QR) algorithm and compared the parallel speedup and the convergence property of our algorithm with that of the existing algorithms. To compare our algorithm with the multishift QR (M-QR) algorithm in its best condition, we derived the optimal division number for the M-QR algorithm by modeling the execution time. The results of implementation on a shared memory parallel machine (Fujitsu PRIMEPOWER HPC2500) were as follows:

(1) Although the M-QR algorithm showed the best convergence property except in one eigenproblem, the FPM-QR algorithm showed greater total performance because of the efficiency of running processors. The attained speedup of the FPM-QR algorithm compared with the M-QR algorithm was up to 1.9.

(2) The FPM-QR algorithm can improve the convergence property of the deferred shift QR (D-QR) algorithm while keeping all the processors fully operative. The attained speedup of the FPM-QR algorithm compared with the D-QR algorithm was up to 1.7.

In summary, the FPM-QR algorithm showed greater performance, or was at least competitive for eight kinds of eigenproblems.

Our future work includes the analysis of the convergence behavior of the FPM-QR algorithm to know why the FPM-QR algorithm shows better convergence than the M-QR algorithm for some problems. And the asymptotic convergence rate of the FPM-QR algorithm is to be analyzed. We also plan to implement the FPM-QR algorithm on distributed memory parallel machines.

Acknowledgments This work was supported by a Grant-in-Aid for 21st Century COE "Frontiers of Computational Science".

References

1) Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S. and Sorensen, D.: *LAPACK User's Guide*, SIAM (1992).
2) Bai, Z. and Demmel, J.: On a block implementation of Hessenberg QR iteration, *Intl. J. of High Speed Comput.*, Vol.1, pp.97–112 (1989).
3) Chang, H., Utku, S., Sakama, M. and Rapp, D.: A parallel Householder tridiagonalization using scattered square decomposition, *Parallel Computing*, Vol.6, No.3, pp.297–311 (1988).
4) Cuppen, J.J.M.: A divide and conquer method for the symmetric eigenproblem, *Numer. Math.*, Vol.36, pp.177–195 (1981).
5) Davis, T.: University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/.
6) Dongarra, J.J. and van de Geijn, R.A.: Reduction to condensed form for the eigenvalue problem on distributed architectures, *Parallel Computing*, Vol.18, No.9, pp.973–982 (1992).
7) Francis, J.G.F.: The QR transformation, parts I and II, *Computer Journal*, Vol.4, pp.265–271, 332–345 (1961-62).
8) Fukuzawa, K., Kitaura, K., Nakata, K., Kaminuma, T. and Nakano, T.: Fragment molecular orbital study of the binding energy of ligands to the estrogen recepter, *Pure and Applied Chemistry*, Vol.75, pp.2405–2410 (2003).
9) Golub, G.H. and van Loan, C.F.: *Matrix Computations*, 3rd ed., Johns Hopkins University Press (1996).
10) Gu, M. and Eisenstat, S.C.: A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem, *SIAM J. Matrix Anal. Appl.*, Vol.16, pp.172–191 (1995).
11) http://openmp.org/wp/
12) Inadomi, Y., Nakano, T., Kitaura, K. and Nagashima, U.: Definition of molecular orbitals in fragment molecular orbital method, *Chemical Physics Letters*, Vol.364, pp.139–143 (2002).
13) Katagiri, T. and Kanada, Y.: An efficient implementation of parallel eigenvalue computation for massively parallel processing, *Parallel Computing*, Vol.27, No.14, pp.1831–1845 (2001).
14) Kaufman, L.: A parallel QR algorithm for the symmetric tridiagonal eigenvalue problem, *J. Parallel and Distrib. Comput.*, Vol.3, pp.429–434 (1994).
15) Kublanovskaya, V.N.: On some algorithms for the solution of the complete eigenvalue problem, *U.S.S.R. Comput. Math. and Math. Phys.*, Vol.3, pp.637–657 (1961).
16) Van de Geijn, R.A.: Deferred shifting schemes for parallel QR methods, *SIAM J.*

Matrix Anal. Appl., Vol.14, pp.180–194 (1993).
17) Watkins, D.S.: Shifting strategies for the parallel QR algorithm, *SIAM J. Sci. Comput.*, Vol.15, pp.953–958 (1994).
18) Watkins, D.S. and Elsner, L.: Convergence of algorithms of decomposition type for the eigenvalue problem, *Lin. Alg. Appl.*, Vol.143, pp.19–47 (1991).
19) Wilkinson, B. and Allen, M.: *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall (1999).
20) Wilkinson, J.H.: *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford (1965).
21) Wilkinson, J.H.: Global convergence of tridiagonal QR algorithm with origin shifts, *Lin. Alg. Appl.*, Vol.1, pp.409–420 (1968).
22) Yamamoto, Y.: Recent developments in algorithms for solving dense eigenproblems (II) — multishift QR algorithms, *Trans. JSIAM*, Vol.16, No.4, pp.507–534 (2006) (in Japanese).

Appendix

A.1 The Optimal Division Number for the M-QR Algorithm

In this section, we derive the optimal division number of the existing M-QR algorithm to achieve the shortest execution time. To this end, we model the execution time $T_{\rm model}$ of the M-QR algorithm as follows:

$$T_{\rm model} = T_{\rm bulge} + T_{\rm idle} + T_{\rm shift} + T_{\rm sync}, \tag{3}$$

where $T_{\rm bulge}$ is the total time of bulge-chasing, $T_{\rm idle}$ is the total time of processor idling, $T_{\rm shift}$ is the total time of shift computation, and $T_{\rm sync}$ is the total time of processor synchronization. Thus our model ignores memory access conflicts and cash effects.

The input parameters of this model are as follows:

- n: the order of the matrix,
- m: the number of shifts,
- $i_{{\rm avg},m}$: the average iteration number to get one eigenvalue,
- $t_{\rm bulge}$: the time of moving a bulge down by one row,
- $t_{{\rm shift},m}$: the time of computing m shifts,
- $t_{\rm sync}$: the time for one synchronization.

Among these, $i_{{\rm avg},m}$ is dependent on the input matrix, and $t_{\rm bulge}$, $t_{{\rm shift},m}$, $t_{\rm sync}$ are dependent on the computational environments. We show the measured values of $t_{\rm bulge}$, $t_{{\rm shift},m}$, and $t_{\rm sync}$ in **Table 3**. These quantities are experimentally determined only once for each computational environment. Then, $t_{\rm bulge}$ and $t_{\rm sync}$ can be used to determine the optimal division number of the M-QR algorithm, see Eq. (2). On the other hand, $t_{\rm bulge}$ and $t_{{\rm shift},m}$ can be used to determine the load rebalance in the FPM-QR algorithm to hide the cost of shift update, see Section 3.2.

Table 3 Experimental results of the parameters. Details of the testing matrix (Type 1, $n = 20000$) and the computational environment is described in Section 4.

m	$t_{\rm bulge}$	$t_{{\rm shift},m}$	$t_{\rm sync}$
4	8.4E-08	2.1E-06	1.9E-06
8	8.4E-08	7.9E-06	3.7E-06
16	8.4E-08	2.9E-05	4.9E-06
32	8.4E-08	1.1E-04	8.4E-06

The other parameter, $i_{{\rm avg},m}$, is the average iteration number to get one eigenvalue weighted by the length of bulge-chasing. Let $L_{i,j}$ denote the length of bulge-chasing in the i-th multishift QR step with the j-th shift. Also, let $N_{\rm step}$ denote the number of multishift QR steps needed to compute all the eigenvalues. Then the total length of bulge-chasing is $\sum_{i=1}^{N_{\rm step}} \sum_{j=1}^{m} L_{i,j}$. On the other hand, if the single-shift QR algorithm is used and each eigenvalue is computed in only one iteration, the total length of bulge-chasing is $N_L = (n-1)+(n-2)+\cdots+1 = n(n-1)/2$. Using these quantities, $i_{{\rm avg},m}$ is defined as

$$i_{{\rm avg},m} = \frac{1}{N_L} \sum_{i=1}^{N_{\rm step}} \sum_{j=1}^{m} L_{i,j}. \tag{4}$$

To get all eigenvalues, the (single-shift) QR steps of about $i_{{\rm avg},m}(n-1)$ times are needed. One multishift QR step corresponds to m single-shift QR steps, so the total multishift QR steps, $N_{\rm step}$, is approximately

$$N_{\rm step} = \frac{i_{{\rm avg},m}(n-1)}{m}. \tag{5}$$

In the following, we focus on the processor m and derive the expressions for $T_{\rm bulge}$, $T_{\rm idle}$, $T_{\rm shift}$ and $T_{\rm sync}$.

- The total time of bulge-chasing
  As shown in Fig. 2, the time of bulge-chasing in the i-th multishift QR step is $L_{i,m} t_{\rm bulge} \simeq (\sum_{j=1}^{m} L_{i,j}\, t_{\rm bulge})/m$. Hence, the total time of bulge-chasing

can be written as follows:

$$T_{\text{bulge}} = \frac{1}{m} \sum_{i=1}^{N_{\text{step}}} \sum_{j=1}^{m} L_{i,j}\, t_{\text{bulge}} = \frac{n}{2} N_{\text{step}} t_{\text{bulge}}. \tag{6}$$

- The total time of processor idling
  In one multishift QR step of the M-QR algorithm with m shifts and $M\,(\geq m)$ division of a matrix, the processor idle time is $(m-1)/M$ times of the bulge-chasing time. As a result, the total time of processor idling can be written as follows:

$$T_{\text{idle}} = \frac{m-1}{M}\, T_{\text{bulge}}. \tag{7}$$

  By increasing M, T_{idle} can be decreased. But, the cost of processor synchronization increases at the same time. As a result, there is a trade-off between the processor idle time and the synchronization cost. The optimal division number is derived later.
- The total time of shift computation
  The M-QR algorithm updates the shifts at the end of each multishift QR step, so the total time of shift computation can be written as follows:

$$T_{\text{shift}} = N_{\text{step}}\, t_{\text{shift},m}. \tag{8}$$

- The total time of processor synchronization
  In each multishift QR step, synchronization is necessary whenever a bulge passes through the boundary of the matrix regions. So there are $M+m-1$ synchronization points and two synchronization (one for bulge arrival and another for bulge departure) is needed at each synchronization point. Hence, the total time of processor synchronization can be written as follows:

$$T_{\text{sync}} = 2N_{\text{step}}(M+m-1)\, t_{\text{sync}}. \tag{9}$$

The execution time of the M-QR algorithm can be written as the sum of the four terms as follows:

$$T_{\text{model}} = \frac{i_{\text{avg},m}(n-1)}{m} \cdot \left\{ \frac{n}{2} \left(1 + \frac{m-1}{M} \right) t_{\text{bulge}} \;+\; t_{\text{shift},m} \;+\; 2(M+m-1)\, t_{\text{sync}} \right\}. \tag{10}$$

The optimal division number M_{opt} for the M-QR algorithm, which minimizes the execution time, is obtained from $\partial T_{\text{model}}/\partial M = 0$, see Eq. (2).

A.2 Pseudocode of the FPM-QR Algorithm

We show a simplified pseudocode of the FPM-QR algorithm with OpenMP [11] for shared memory parallel machines. In this code, *Bulge-set*(s) means setting a bulge at the upper left corner of the matrix by the first similarity transformation with shift s. *Bulge-chasing*(i, j) means moving a bulge from i-th row to j-th row.

In this algorithm, the processor myrank (myrank $= 0, \ldots, m-1$) waits in an idle state until the first myrank bulges go through the first region. Then it sets a bulge at the upper left corner of the matrix and starts chasing the bulge. When the bulge passes through the boundary of the regions, inter-processor synchronization is inserted. After the bulge reaches the bottom right corner, the processor updates a shift and start the next bulge-chasing immediately. When a processor in the undermost region detects a sufficiently small subdiagonal element, it separates a small matrix, computes its eigenvalues with a single-shift QR algorithm, and updates the value of N. The update of N is done within a block between two barriers for inter-processor synchronization. Using this new value of N, all the processors determine the next region of bulge-chasing.

```
1:  N ← n                                          ▷ n: order of the input matrix
2:  Δ ← t_shift,m / t_bulge                        ▷ See Table 3
3:  #pragma omp parallel private(myrank, s_myrank, i, j, start, end)
4:  Begin
5:      myrank ← omp_get_thread_num()
6:      Initialization of shift s_myrank
7:      for i = 0, 1, ..., myrank − 1 do           ▷ Idle time only at the first step
8:          #pragma omp barrier
9:          #pragma omp barrier
10:     end for
11:     for i = 1, 2, ..., do
12:         Bulge-set(s_myrank)
13:         for j = 1, 2, ..., m − 1 do
14:             start ← 1 + (j − 1)(N + Δ)/m
15:             end ← j(N + Δ)/m
```

```
16:            #pragma omp barrier
17:            Bulge-chasing(start, end)
18:            #pragma omp barrier
19:        end for
20:        start ← 1 + (m − 1)(N + Δ)/m
21:        end ← N
22:        #pragma omp barrier
23:        Bulge-chasing(start, end)
24:        if any subdiagonal element converges to zero then      ▷ Convergence
   check
25:            Update of N & Solution of small eigenproblem
26:        end if
27:        Update of shift s_myrank
28:        #pragma omp barrier
29:    end for
30: End
```

(Received April 4, 2008)
(Accepted August 20, 2008)

Takafumi Miyata received his master's degree in Engineering from Nagoya University in 2007. His research interests include numerical algorithm for eigenvalue problems and parallel computing.

Yusaku Yamamoto received his master's degree in Material Physics from the University of Tokyo in 1992. He worked in Central Research Laboratory, Hitachi, Ltd. from 1992. He became a visiting scholar at business school of Columbia University in 2001. He became an assistant professor at Nagoya University in 2003. He received his Ph.D. in Engineering from Nagoya University in 2003. He became a lecturer at Nagoya University in 2004, associate professor at Nagoya University in 2006. His research interests include numerical algorithm for large-scale matrix computation and financial engineering.

Shao-Liang Zhang received his master's degree in Computational Mathematics from Jilin University in 1983. He received his Ph.D. in Engineering from University of Tsukuba in 1990. He became a research at Institute of Computational Fluid Dynamics in 1990. He became an assistant professor at Nagoya University in 1993, a lecturer at University of Tsukuba in 1995, an associate professor at the University of Tokyo in 1998, and a professor at Nagoya University in 2005. His research interests include numerical iterative algorithm for large-scale matrix computation and parallel computing.