East Asian Journal on Applied Mathematics
doi: 10.4208/eajam.300510.250311a

Vol. **1**, No. 2, pp. 187-196
May 2011

Optimization of the Multishift QR Algorithm with Coprocessors for Non-Hermitian Eigenvalue Problems

Takafumi Miyata[*,1], Yusaku Yamamoto[2], Takashi Uneyama[3], Yoshimasa Nakamura[4] and Shao-Liang Zhang[1]

[1] *Graduate School of Engineering, Nagoya University, Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan.*
[2] *Graduate School of System Informatics, Kobe University, 1-1 Rokkodai-cho, Nada-ku, Kobe 657-8501, Japan.*
[3] *Institute for Chemical Research, Kyoto University, Gokasho, Uji 611-0011, Japan.*
[4] *Graduate School of Informatics, Kyoto University, 36-1 Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan.*

Received 30 May 2010; Accepted (in revised version) 25 March 2011

Available online 7 April 2011

Abstract. The multishift QR algorithm is efficient for computing all the eigenvalues of a dense, large-scale, non-Hermitian matrix. The major part of this algorithm can be performed by matrix-matrix multiplications and is therefore suitable for modern processors with hierarchical memory. A variant of this algorithm was recently proposed which can execute more computational parts by matrix-matrix multiplications. The algorithm is especially appropriate for recent coprocessors which contain many processor-elements such as the CSX600. However, the performance of the algorithm highly depends on the setting of parameters such as the numbers of shifts and divisions in the algorithm. Optimal settings are different depending on the matrix size and computational environments. In this paper, we construct a performance model to predict a setting of parameters which minimizes the execution time of the algorithm. Experimental results with the CSX600 coprocessor show that our model can be used to find the optimal setting.

AMS subject classifications: 65F15, 65Y10, 65Y20

Key words: Eigenvalues, multishift QR algorithm, bulge-chasing, performance modeling.

1. Introduction

We consider computing all the eigenvalues of the standard eigenvalue problem

$$A\boldsymbol{x} = \lambda \boldsymbol{x}, \ \ \boldsymbol{x} \neq \boldsymbol{0} \tag{1.1}$$

[*]Corresponding author. *Email addresses:* miyata@na.cse.nagoya-u.ac.jp (T. Miyata), yamamoto@cs.kobe-u.ac.jp (Y. Yamamoto), uneyama@scl.kyoto-u.ac.jp (T. Uneyama), ynaka@amp.i.kyoto-u.ac.jp (Y. Nakamura), zhang@na.cse.nagoya-u.ac.jp (S.-L. Zhang)

where A is an $n \times n$ complex non-Hermitian matrix. We suppose that the matrix A is dense and large-scale. Examples of non-Hermitian eigenvalue problems arising in scientific computing and many other applications are given in [1].

In the standard procedure to solve the problem in Eq. (1.1), the non-Hermitian matrix A is first reduced to a Hessenberg form by the Householder algorithm [10]. After that, the Hessenberg matrix is transformed to a Schur form by the QR algorithm [8, 9, 13]. These two steps consist of unitary transformations, i.e., the eigenvalues λ of the matrix A are not changed by these steps and are finally given by the diagonal elements of the Schur form. Among these steps, the second step by the QR algorithm requires more computational work. Hence it is necessary to speed up the QR algorithm. The promising approach is a block implementation of the QR algorithm which is referred to as the multishift QR algorithm [2, 3]. The algorithm can perform most of the computational work by matrix-matrix multiplications and is therefore suitable for modern processors with hierarchical memory.

Recently, a recursive implementation of the multishift QR algorithm was proposed [18] to perform more computational parts by matrix-matrix multiplications. The algorithm is especially appropriate for recent coprocessors which have many processor-elements, e.g., GRAPE-DR [11] and CSX600 [4]. The performance of the recursive algorithm highly depends on the setting of parameters in the algorithm. Optimal settings are different depending on the matrix size n and computational environments.

In this paper, we optimize the recursive multishift QR algorithm for coprocessors to attain the minimal execution time of the algorithm for a given matrix size n and a computational environment. We adopt the hierarchical modeling approach [5–7] and construct a performance model of the algorithm. For a given set of parameters, our model provides the prediction of the execution time of the algorithm. By using the model, we can predict the execution time for several sets of parameters and choose the optimal one among them before running the algorithm.

Note that the hierarchical modeling approach has long been exploited to optimize the multishift QR algorithm [14, 17]. Here we consider optimizing the recursive version of the multishift QR algorithm. This approach enables us to fully exploit the potential of recent coprocessors.

This paper is organized as follows. The multishift QR algorithm and its recursive version are described in Section 2. We give performance modeling of the recursive algorithm in Section 3. Experimental results with the CSX600 coprocessor are presented in Section 4. Section 5 gives some concluding remarks and future work. Throughout this paper, the conjugate transpose is denoted by $(\cdot)^*$. The $n \times n$ identity matrix is denoted by I.

2. The Multishift QR Algorithm and Its Recursive Implementation

In this section, we review some of the standard facts on the multishift QR algorithm [3] and its variant to fully utilize coprocessors [18]. We focus on computing all the eigenvalues of a complex Hessenberg matrix.

2.1. The multishift QR algorithm

The convergence of the QR algorithm can be accelerated by shifts which are approximate eigenvalues. The double-shift QR algorithm [10], which uses two shifts, is well known and is widely used in many applications for long years. As a block version of this, Bai et al. [2] proposed the multishift QR algorithm, which uses m shifts, to perform $m/2$ steps of the double-shift QR algorithm at once (m is even). A major part of this algorithm can be executed by matrix-matrix multiplications and is therefore suitable for modern processors with hierarchical memory. However, this algorithm suffers from slower convergence due to numerical errors [15]. To overcome this difficulty, Braman et al. [3] modified the multishift QR algorithm while keeping the good property of the original algorithm. In this paper, we focus on the modified algorithm and refer to it as the multishift QR algorithm.

Let A_0 be an $n \times n$ complex Hessenberg matrix and A_ℓ be the matrix after the ℓ-th QR iteration. In the multishift QR algorithm, to obtain $A_{\ell+m}$ from A_ℓ, m shifts $\sigma_1, \sigma_2, \ldots, \sigma_m$ are first computed. The shifts are the m eigenvalues of the $m \times m$ trailing principal submatrix of A_ℓ. After that, $A_{\ell+m}$ is computed in the following step

$$\begin{cases} (A_\ell - \sigma_2 I)(A_\ell - \sigma_1 I) \rightarrow Q_\ell R_\ell, \\ A_{\ell+2} \leftarrow Q_\ell{}^* A_\ell Q_\ell, \\ \vdots \\ (A_{\ell+m-2} - \sigma_m I)(A_{\ell+m-2} - \sigma_{m-1} I) \rightarrow Q_{\ell+m-2} R_{\ell+m-2}, \\ A_{\ell+m} \leftarrow Q_{\ell+m-2}^* A_{\ell+m-2} Q_{\ell+m-2} \end{cases} \tag{2.1}$$

where $Q_{\ell+i}$ is the unitary matrix and $R_{\ell+i}$ is the upper triangular matrix, produced by the QR factorization of $(A_{\ell+i} - \sigma_{i+2} I)(A_{\ell+i} - \sigma_{i+1} I)$ $(i = 0, 2, \ldots, m-2)$. When the number of shifts $m = 2$ in Eq. (2.1), the above step is equivalent to the one step of the double-shift QR algorithm [10].

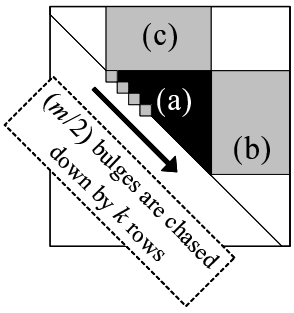

Figure 1: The operation pattern of the multishift QR algorithm. The blocks (a), (b) and (c) in this figure are modified at Steps (a), (b) and (c) in bulge-chasing, respectively. Steps (b) and (c) can be executed by matrix-matrix multiplications.

In a practical implementation of the double-shift QR algorithm, the effect of shifts is implicitly introduced. The operation pattern of one step of the algorithm is so called bulge-chasing, which moves a bulge from the upper left corner to the bottom of a matrix [10]. In a similar way, the multishift QR algorithm, using m shifts, performs $m/2$ sets of bulge-chasing operations at once. In the multishift QR algorithm, $m/2$ bulges are chased down by k rows at a time, where k is some positive integer. This is done in the following three steps [3].

Step (a): Chase each of the $m/2$ bulges by k rows sequentially within the diagonal block (a) in Fig. 1. At the same time, accumulate the Householder transformations, used to move bulges at this step, into a unitary matrix U (the size of U is $(3m/2+k)$).

Step (b): Multiply the off-diagonal block (b) in Fig. 1 by U from left.

Step (c): Multiply the off-diagonal block (c) in Fig. 1 by U^* from right.

By repeating these steps, all the bulges are moved from the upper left corner to the bottom of the matrix. Among these steps, matrix-matrix multiplications at Steps (b) and (c) are dominant. This is the reason why the multishift QR algorithm can exploit the potential of modern processors.

In the multishift QR algorithm, there are two parameters m and k, which affect the performance of the algorithm. It is shown that $k = 3m/2$ is optimal to minimize the computational work [3]. We adopt this choice of k in the following. In this case, the size of the unitary matrix U is $3m$ and all that left is to optimize m. For this parameter, the performance model was constructed to predict the effect of m on the performance [17]. By experimental results, it was shown that the model can be used to find nearly optimal value of m to minimize the computational time of the algorithm for a given matrix and a computational environment.

2.2. The recursive multishift QR algorithm for coprocessors

As shown in Section 2.1, most of the computation in the multishift QR algorithm is performed by matrix-matrix multiplications, which are Steps (b) and (c) in Fig. 1. It is well known that a matrix-matrix multiplication with $m \times m$ matrices can perform $O(m^3)$ computations while requiring $O(m^2)$ data-transfer, i.e., this operation can reuse the transfered data. In view of this, we can say that the multishift QR algorithm is suitable not only for modern processors but also for recent coprocessors such as the CSX600 [4]. This is because, to use coprocessors, we need to transfer some data between memory and coprocessors, see Fig. 2. The speed of data-transfer is often relatively slower than that of operation by coprocessors, e.g., the ClearSpeed Advance board [4] contains 2 chips of the CSX600 coprocessor (48 GFlops at most) and is connected to a computer by PCI-X Bus (1.0 GB / sec.). In [18], the CSX600 coprocessor was used for matrix-matrix multiplications in the multishift QR algorithm, which is computationally dominant part. Note that, for other

computational parts, the coprocessor was not used because it is difficult to compensate for the data-transfer cost.

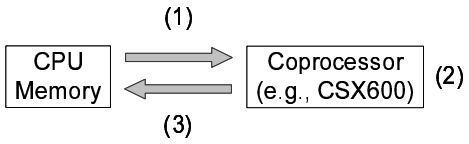

Figure 2: How to use a coprocessor. (1) Transfer data from memory to a coprocessor (2) Compute matrix-matrix multiplications (3) Transfer computational results to memory.

The effect of the CSX600 coprocessor on the multishift QR algorithm ($k = 3m/2$) is shown in Fig. 3. Comparing the left result with the middle result in Fig. 3, we can see that the coprocessor accelerates matrix-matrix multiplications at Steps (b) and (c) in bulge-chasing operations. Comparing the middle result with the right result in Fig. 3, we can also see the effect of the parameter m. For larger value of m, the execution time of matrix-matrix multiplications is decreased. However, the time for other computations is increased. These reasons are described as follows. For larger value of m, as shown in Section 2.1, matrix-matrix multiplications at Steps (b) and (c) are performed with larger-sizes of matrices. This will give more chance to reuse the data transfered from memory to the coprocessor and can exploit the potential of the coprocessor. However, as m increases, the computational work at Step (a) increases. This part can not be executed by matrix-matrix multiplications and thus become a bottleneck.

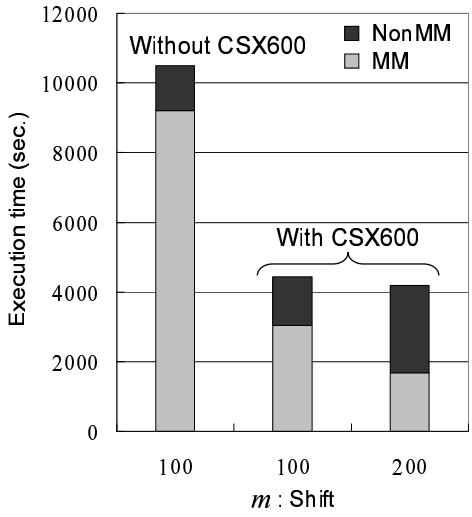

Figure 3: The execution time of the multishift QR algorithm for all the eigenvalues of a complex Hessenberg matrix ($n = 6000$). "MM" shows matrix-matrix multiplications at Steps (b) and (c) in bulge-chasing operations. "NonMM" shows other operations such as Step (a) in bulge-chasing operations. Details of the computational environment are described in Section 4.

To overcome this difficulty, the computational work at Step (a) was reformulated. As described above, the operation by matrix-matrix multiplications plays a key role in high performance computing. One way to accelerate the operation at Step (a) is to transform it to matrix-matrix multiplications. This can be done as follows. Let q be a common divisor of m and k. Then, the bulge-chasing operation at Step (a) is divided into q sets and each set is composed of the following three steps.

Step (a'): Chase each of the $m/2q$ bulges by k/q rows sequentially within the diagonal block (a') in Fig. 4. At the same time, accumulate the Householder transformations, used to move bulges at this step, into a unitary matrix U' (the size of U' is $(3m/2q + k/q)$).

Step (b'): Multiply the small off-diagonal block (b') in Fig. 4 by U' from left.

Step (c'): Multiply the small off-diagonal block (c') in Fig. 4 by U'^* from right.

These three steps are exactly the same as that in the original algorithm except the sizes of matrices in matrix-matrix multiplications, see Fig. 4. In a similar fashion, Step (a') can be recursively divided into another three steps. As a result, more parts can be performed by matrix-matrix multiplications, which can be executed faster. Such a recursive implementation of the multishift QR algorithm was proposed and showed up to 1.4 times speedup than the original algorithm [18].

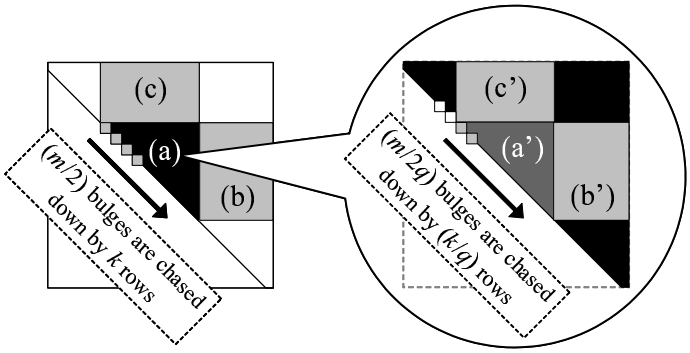

Figure 4: The operation pattern of the recursive multishift QR algorithm. The operation at Step (a), which updates the block (a) in this figure, is divided into q sets. Each set is composed of Steps (a'), (b') and (c') updating the blocks (a'), (b') and (c') in this figure, respectively. Steps (b'), (c'), (b) and (c) can be executed by matrix-matrix multiplications.

In addition to m and k, two more parameters are introduced in the recursive multishift QR algorithm, which are q and the level of recursion (Fig. 4 shows the level one). The execution time of the algorithm depends on the setting of these parameters, e.g., the performance of the CSX600 coprocessor for matrix-matrix multiplications depends on the parameter m, see Fig. 3. It is desirable to find an optimal setting of the parameters which minimizes the execution time of the algorithm before running the algorithm. Note that the optimal setting depends on a given size n of a Hessenberg matrix and a given computational environment. We will pursue this possibility in the following section.

3. Performance Modeling of the Recursive Multishift QR Algorithm

In this section, we construct a performance model of the recursive multishift QR algorithm for coprocessors. The model provides the prediction of the execution time of the algorithm for input parameters. Before running the algorithm, we check the predicted time for several sets of parameters and predict an optimal setting of parameters to minimize the computational time. Note that the construction of the model is needed only once for a given computational environment and it can be used for many sizes of matrices. We optimize two parameters in the recursive multishift QR algorithm. One is m (the number of shifts), the other is q (the number of divisions). For the other parameters, k (the row of moving bulges) is set to be $k = 3m/2$ [3], and the level of recursion is fixed to one [18] (see Fig. 4).

Our approach is based on the hierarchical modeling approach [5–7]. The approach has been used for optimizing several algorithms and turned out to be a promising one for practical use [6,7,16,17]. In this approach, we first construct the models for basic routines which compose the whole algorithm. These models provide the prediction of the execution time of each routine for input parameters, e.g., the model of matrix-matrix multiplications provides us the predicted value of its execution time for given sizes of two matrices. After that, all the predicted time of basic routines is summed up and the total time is given as the predicted time of the whole algorithm.

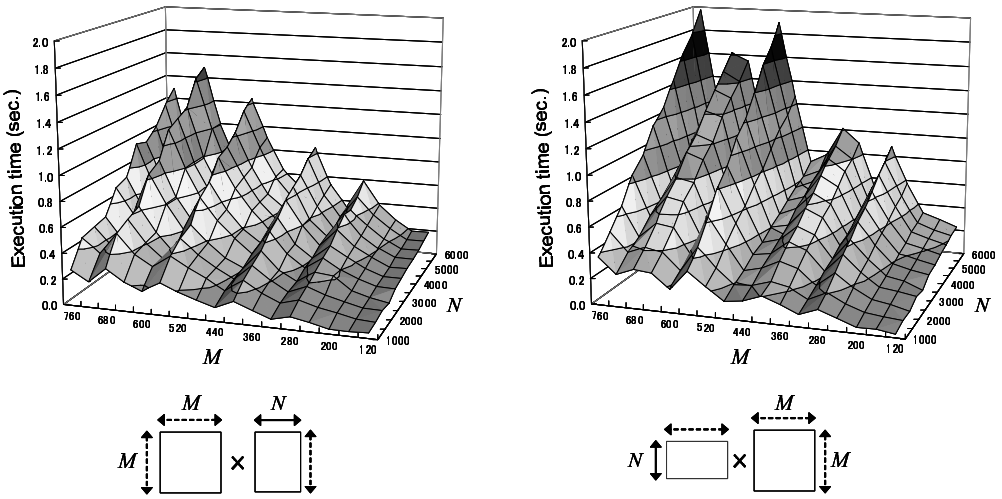

Figure 5: The execution time of matrix-matrix multiplications on the CSX600 coprocessor. The left figure shows the time for multiplying a $M \times N$ real matrix by a $M \times M$ real matrix from left. The right figure shows the time for multiplying a $N \times M$ real matrix by a $M \times M$ real matrix from right. The execution time roughly changes especially with respect to M even though the computational work is proportional to M^2N in both cases.

In [17], the performance model of the multishift QR algorithm was constructed. We modify this model for the recursive multishift QR algorithm. To be more precise, we modify the modeling of two basic routines in the original algorithm. One is the routine for

bulge-chasing operations at Step (a') because the parameter q is added in the recursive algorithm. The other is the routine for matrix-matrix multiplications, which is allocated to a coprocessor such as the CSX600 in the recursive algorithm. In [17], the execution time of this routine is approximated by polynomial interpolation. However, this approach is not promising for the CSX600 coprocessor because the execution time of this routine roughly changes, see Fig. 5.

From the above remarks, we use bilinear interpolation for these basic routines. Let α and β be the parameters of a routine, and denote by $f(\alpha,\beta)$ the execution time of the routine as a function of the parameters. We first measure function values on several grid points in (α,β) plane. After that, to predict a function value between grid points, we use linear interpolation for each grid point. To be more precise, by using measured values $f(\alpha_i,\beta_i)$, $f(\alpha_{i+1},\beta_i)$, $f(\alpha_i,\beta_{i+1})$, $f(\alpha_{i+1},\beta_{i+1})$, we approximate the function value $f(\tilde{\alpha},\tilde{\beta})$ $(\alpha_i \le \tilde{\alpha} \le \alpha_{i+1},\ \beta_i \le \tilde{\beta} \le \beta_{i+1})$ as follows

$$\begin{cases} f(\tilde{\alpha},\tilde{\beta}) = (1-\gamma_2)\left((1-\gamma_1)f(\alpha_i,\beta_i)+\gamma_1 f(\alpha_{i+1},\beta_i)\right) \\ \qquad\qquad + \gamma_2\left((1-\gamma_1)f(\alpha_i,\beta_{i+1})+\gamma_1 f(\alpha_{i+1},\beta_{i+1})\right), \\ \qquad \gamma_1 = \dfrac{\tilde{\alpha}-\alpha_i}{\alpha_{i+1}-\alpha_i}, \quad \gamma_2 = \dfrac{\tilde{\beta}-\beta_i}{\beta_{i+1}-\beta_i}. \end{cases} \tag{3.1}$$

In Eq. (3.1), two parameters (α,β) are (m,q) for the routine of bulge-chasing operations at Step (a'). For the routine of matrix-matrix multiplications, two parameters are the sizes of a square matrix and a rectangular matrix, used at Steps (b'), (c'), (b) and (c) in bulge-chasing operations.

The performance models of other routines are constructed in the same way as in [17], e.g., the execution time of the routine for computing m shifts is approximated as a cubic function in m. After that, the performance model of the whole algorithm is constructed by combining the performance models of all the basic routines. At that time, we assume that $m \times m$ trailing principal submatrix becomes isolated after four iterations of the algorithm. Such an assumption is based on the observation [12] and is adopted in [17].

4. Experimental Results

In this section, we evaluate the performance model constructed in Section 3. Our computational environment consists of Xeon (3.2 GHz) and Memory (8.0 GB). We have carried out the following experiments by Fortran 77 with double precision arithmetic. Our codes were compiled by Intel Fortran Compiler (ver. 9.1) with Intel Math Kernel Library (ver. 8.1). For matrix-matrix multiplications, we used the ClearSpeed Advance Board with two CSX600 chips and CSXL Library (ver. 2.51) [4]. Test matrices are complex Hessenberg matrices whose elements are given by random numbers in real and imaginary parts. The sizes of the matrices are $n = 3000, 6000$. We compare the predicted execution time by our model with the actual execution time of the recursive multishift QR algorithm. The results for two matrices are similar, so we show the results for $n = 6000$ in Fig. 6.

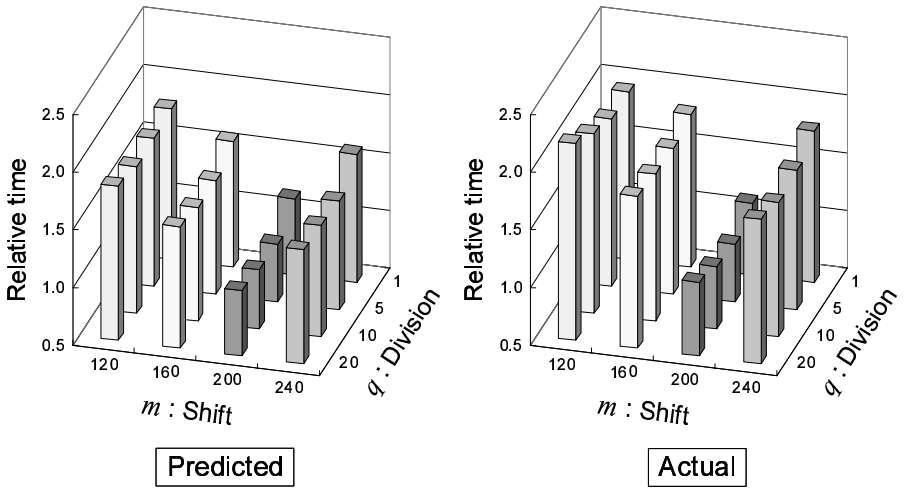

Figure 6: Effect of the two parameters m and q on the execution time of the recursive multishift QR algorithm. The (left / right) figure shows the relative (predicted / actual) time which is normalized by the minimal (predicted / actual) time.

As shown in Fig. 6, the actual execution time of the recursive algorithm changes twice at most by the choice of parameters. Such a difference is due to two changes. One is a change of the coprocessor's performance by the parameter m, the other is a change of operation by introducing matrix-matrix multiplications at Step (a) with the parameter q, see Section 2.2. The dominant part of the algorithm is matrix-matrix multiplications. Availability of the model for the algorithm therefore depends on the model for matrix-matrix multiplications on the CSX600 coprocessor and it does not reproduce the actual time so well for some parts. This is due to the rough change of the computational time as shown in Fig. 5. As a result, the trend of the predicted time of the algorithm is different from the actual time when $m = 120$. However, viewed as a whole, our model reproduces the behavior of the actual time and, thus, can be used for the prediction of the optimal setting of the parameters. Note that we need only a few seconds for the prediction whereas the execution time of the algorithm is thousands seconds per one setting.

5. Conclusion

In this paper, we constructed the performance model of the recursive multishift QR algorithm for coprocessors. Our model is based on the hierarchical modeling approach and is composed of the models of the basic routines, e.g., the bulge-chasing operation and matrix-matrix multiplications. We adopted the CSX600 coprocessor for the recursive algorithm and computed all the eigenvalues of a complex Hessenberg matrix. Experimental results showed that our model reproduced the actual execution time of the recursive algorithm well. Thus, it can be used to find the optimal setting of the parameters which minimizes the execution time of the algorithm.

Our future plan includes the optimization of other parameters, such as the number of rows k in bulge-chasing operation and the level of the recursion. We also plan to implement the recursive multishift QR algorithm on GPU.

Acknowledgment

This work is supported by "The Hori Sciences & Arts Foundation".

References

[1] Z. Bai, D. Day, J. Demmel and J. Dongarra, *A test matrix collection for non-Hermitian eigenvalue problems*, Univ. Tennessee Comput. Sci. T. R., UT-CS-97-355 (1997).

[2] Z. Bai and J. Demmel, *On a block implementation of Hessenberg QR iteration*, Int. J. High Speed Comput., 1 (1989), 97-112.

[3] K. Braman, R. Byers and R. Mathias, *The multishift QR algorithm part I: Maintaining well-focused shifts and level 3 performance*, SIAM J. Matrix Anal. Appl., 23 (2002), 929-947.

[4] ClearSpeed, http://www.clearspeed.com/ .

[5] J. Cuenca, L.-P. García, D. Giménez, J. González and A. Vidal, *Empirical modeling of parallel linear algebra routines*, Lect. Notes Comput. Sci., 3019 (2004), 169-174.

[6] J. Cuenca, D. Giménez and J. González, *Architecture of an automatically tuned linear algebra library*, Parallel Comput., 30 (2004), 187-210.

[7] K. Dackland and B. Kågström, *An hierarchical approach for performance analysis of ScaLAPACK-based routines using the distributed linear algebra machine*, Lect. Notes Comput. Sci., 1184 (1996), 186-195.

[8] J. G. F. Francis, *The QR transformation: A unitary analogue to the LR transformation–part 1*, Comput. J., 4 (1961), 265-271.

[9] J. G. F. Francis, *The QR transformation–part 2*, Comput. J., 4 (1962), 332-345.

[10] G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, London, 1996.

[11] GRAPE-DR, http://grape-dr.adm.s.u-tokyo.ac.jp/index-en.html .

[12] D. Kressner, *Numerical Methods for General and Structured Eigenvalue Problems*, Lect. Notes Comput. Sci. Eng. 46, Springer-Verlag, Berlin, Heidelberg, 2005.

[13] V. N. Kublanovskaya, *On some algorithms for the solution of the complete eigenvalue problem*, U.S.S.R. Comput. Math. Math. Phys., 3 (1961), 637-657.

[14] T. Miyata, Y. Yamamoto and S.-L. Zhang, *Performance modeling of multishift QR algorithms for the parallel solution of symmetric tridiagonal eigenvalue problems*, Lect. Notes Comput. Sci., 6082 (2010), 401-412.

[15] D. S. Watkins, *The transmission of shifts and shift blurring in the QR algorithm*, Lin. Alg. Appl., 241-243 (1996), 877-896.

[16] Y. Yamamoto, *Performance modeling and optimal block size selection for a BLAS-3 based tridiagonalization algorithm*, Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region, 2005, 249-256.

[17] Y. Yamamoto, *Performance modeling and optimal block size selection for the small-bulge multishift QR algorithm*, Lect. Notes Comput. Sci., 4330 (2006), 451-463.

[18] Y. Yamamoto, T. Miyata and Y. Nakamura, *Accelerating the complex Hessenberg QR algorithm with the CSX600 floating-point coprocessor*, Proceedings of Parallel and Distributed Computing and Systems, 2007, 204-211.