Performance Modeling and Optimal Block Size Selection for the Small-Bulge Multishift QR Algorithm

Yusaku Yamamoto

Nagoya University, Nagoya, Aichi, 464-8603, Japan,
yamamoto@na.cse.nagoya-u.ac.jp,
WWW home page: http://www.na.cse.nagoya-u.ac.jp/~yamamoto

Abstract. The small-bulge multishift QR algorithm proposed by Braman, Byers and Mathias is one of the most efficient algorithms for computing the eigenvalues of nonsymmetric matrices on processors with hierarchical memory. However, to fully extract its potential performance, it is crucial to choose the block size m properly according to the target architecture and the matrix size n. In this paper, we construct a performance model for this algorithm. The model has a hierarchical structure that reflects the structure of the original algorithm and given n, m and the performance data of the basic components of the algorithm, such as the level-3 BLAS routines and the double implicit shift QR routine, predicts the total execution time. Experiments on SMP machines with PowerPC G5 and Opteron processors show that the variation of the execution time as a function of m predicted by the model agrees well with the measurements. Thus our model can be used to automatically select the optimal value of m for a given matrix size on a given architecture.

1 Introduction

The QR algorithm [1][2][3] is widely used as an efficient and reliable method to compute the eigenvalues of small to medium nonsymmetric matrices. However, it is not straightforward to implement the QR algorithm in a way that fully exploits the performance of modern computers such as the shared-memory parallel machines and processors with hierarchical memory. In fact, when applying the conventional double implicit shift QR algorithm to an $n \times n$ Hessenberg matrix, the parallel granularity is only $O(n)$. This is often too small to attain reasonable speedup in the presence of large inter-processor synchronization costs. In addition, the algorithm sweeps the whole matrix in each QR iteration, while performing only $O(1)$ arithmetic operations on each matrix element. This results in poor data reference locality and prevents effective use of cache memory.

Many efforts have been made to overcome these difficulties so far [4][5][6][7][8][9]. Among them, the small-bulge multishift QR algorithm proposed by Braman, Byers and Mathias [5] seems the most promising approach. Their algorithm computes m shifts at once as the eigenvalues of the $m \times m$ trailing principal submatrix, as in the conventional multishift QR algorithm [4], and performs so-called

bulge chasing [10][11] using these m shifts simultaneously. However, instead of chasing one large bulge containing the m shifts, their algorithm chases $m/2$ small bulges each containing only 2 shifts simultaneously in a pipelined fashion. This removes the defects of conventional multishift QR algorithm — numerical instability due to the use of large bulge and the resulting deterioration of convergence speed [6][12] — and recovers the excellent convergence property of the original double shift QR algorithm. Since the parallel granularity and the number of operations per iteration and per matrix element are increased to $O(m^2n)$ and $O(m)$, respectively, as a result of using m shifts, this algorithm is ideally suited for modern architectures. It has been reported that this algorithm can achieve up to three times speedup over DHSEQR, the large-bulge multishift QR algorithm in LAPACK [13], on the Origin2000 [5].

To fully extract the potential performance of the small-bulge multishift QR algorithm, it is crucial to choose the block size m properly. Larger m will provide larger parallel granularity and more chance for data reuse, but the cost of computing the shifts grows with m. The optimal value of m varies depending on the target machine, the number of processors and the matrix size n. In principle, the optimal value of m can be determined by trial and error, but this process will require huge time.

In this paper, we present a performance model for the small-bulge multishift QR algorithm. It is based on the hierarchical approach proposed by Dacklund [14] and Cuenca et al. [15][16], and given the execution time models of each component of the algorithm, such as the level-3 BLAS routines and the conventional double shift QR algorithm to compute the shifts, predicts the total execution time accurately. Using this performance model, the optimal block size for a given architecture, the number of processors and matrix size can be determined prior to execution.

There are many studies on automatic optimization of linear algebra programs. Katagiri et al. [17] propose I-LIB, an automatically tuned linear algebra library for distributed-memory parallel machines. They report the result of optimizing parameters for tridiagonalization of symmetric matrices and show that considerable performance gains can be obtained through optimization. However, to find the optimal set of parameters, I-LIB measures the execution time of the whole program repeatedly with different values of parameters. Hence the cost of tuning is very high.

The hierarchical approach to performance modeling was first proposed by Dackland [14] and Cuenca [15][16]. The basic idea of this approach is to exploit the natural hierarchy existing in linear algebra programs. In this approach, the execution time models of lower-level routines such as the BLAS are constructed first, and the total execution time is estimated by accumulating the execution times of the lower-level routines. This approach has been applied to performance modeling and optimization of LU decomposition [15], QR decomposition [14], tridiagonalization of symmetric matrices [18] and so on and has proved useful in reducing the cost of tuning without sacrificing accuracy. Our work is along this line of research.

Automatic optimization of linear algebra programs are studied both at a higher and a lower level. For example, the Self-Adapting Numerical Software (SANS) proposed by Dongarra et al. [19] aims at automatically selecting the best routine (not parameters within a routine) to solve a given problem. On the other hand, there are projects like ATLAS [20] and PhiPAC [21], which try to maximize the performance of basic routines such as BLAS by automatic tuning. These studies deal with problems that are at different levels from ours and are therefore complementary with our work.

This paper is structured as follows: in section 2, we give a brief explanation of the small-bulge multishift QR algorithm. Section 3 gives the details of our performance model. Experimental results that demonstrate the effectiveness of our model are presented in section 4. Finally, section 5 gives some concluding remarks.

2 The small-bulge multishift QR algorithm

2.1 The algorithm

Let us consider the application of the QR algorithm on an $n \times n$ Hessenberg matrix A and denote the matrix after the l-th QR iteration by A_l. In the conventional double implicit shift QR algorithm, to compute A_{l+2} from A_l, we first compute the two shifts σ_1 and σ_2 as the eigenvalues of the trailing principal submatrix of A_l, introduce a 4×4 bulge containing the information of the shifts at the top left corner of A_l and then chase the bulge along the diagonal by repeatedly applying Householder transformations until it disappears from the bottom right corner. Then the implicit Q theorem [10][11] guarantees that the resulting matrix is the one that would be obtained by applying two explicit QR steps with shifts σ_1 and σ_2 to A_l.

This idea can be naturally extended to the multishift QR algorithm [4]. In this case, we compute m shifts $\sigma_1, \sigma_2, \ldots, \sigma_m$ as the eigenvalues of the trailing principal submatrix of A_l, introduce a large $(m+2) \times (m+2)$ bulge containing their information and then perform bulge chasing. By using a large bulge, both parallel granularity and data reference locality of the bulge-chasing step can be increased. The LAPACK routine DHSEQR is based on this idea. Unfortunately, it has been shown that as the size of the bulge increases, the shift information contained in the bulge becomes extremely prone to being contaminated by numerical errors [12]. This prevents the accurate shift information to be conveyed to the bottom right corner of the matrix and thereby retards convergence [12]. Thus the value of m is usually limited to about ten, but this is too few to use cache memory efficiently.

To overcome this difficulty, Braman, Byers and Mathias propose the small-bulge mltishift QR algorithm [5]. In this algorithm, the m shifts are divided into $m/2$ sets of double shifts and in the bulge-chasing process, $m/2$ small 4×4 bulges are chased simultaneously in a pipelined fashion. Although this algorithm is mathematically equivalent to the large-bulge multishift QR algorithm, it can

avoid numerical difficulties associated with the use of large bulge and recover the excellent convergence properties of the conventional double shift QR algorithm.

The bulge chasing in the small-bulge multishift QR algorithm can be divided into three phases. In phase I, a chain of $m/2$ bulges is introduced at the top left corner of the matrix. Since bulges have to be at least three rows apart to avoid interference, the bulges occupy rows 1 through $3m/2+1$ when the phase I is completed (Fig. 1). Note that to chase the bulges in phase I, only the first $(3m/2+1)\times(3m/2+1)$ submatrix of A_l, which we denote by $A_{l,1:3m/2+1,1:3m/2+1}$ following [11], is necessary. We therefore divide the work in phase I into two steps (see Fig. 2):

(a) Chase the bulges along the diagonal by applying a sequence of Householder reflections from both sides to $A_{l,1:3m/2+1,1:3m/2+1}$. At the same time, form the product of these reflections as an $(3m/2+1)\times(3m/2+1)$ matrix U.
(b) Update the rest of rows 1 through $3m/2+1$ by multiplying $A_{l,1:3m/2+1,3m/2+2,n}$ by U from the left.

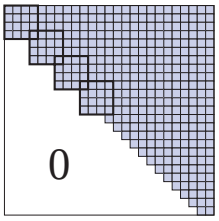

Fig. 1. The matrix after the completion of phase I ($m = 8$). Four bulges have been introduced in rows 1 through (3/2)*8+1.

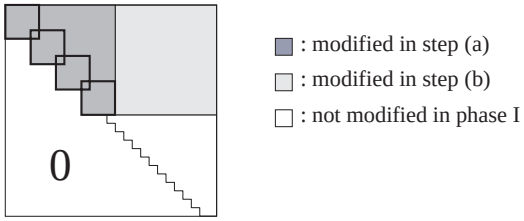

Fig. 2. Division of the work in phase I into two parts: (a) bulge-chasing in the diagonal block and (b) update of the off-diagonal block.

Of these two steps, step (b) can be done entirely with the level-3 BLAS, or matrix multiplication, and the copy operation needed to move the results back

into $A_{l,1:3m/2+1,3m/2+2,n}$. Though step (a) cannot be performed with the level-3 BLAS, the computational work needed for this step is much smaller than that for step (b) when $m \ll n$. Thus most of the work in phase I can be organized as level-3 BLAS.

In phase III, the $m/2$ bulges that lie on the last $3m/2+1$ rows of A_l are chased out of the matrix. As in phase I, the work can be divided into two steps, namely, (a) bulge-chasing within the trailing $(3m/2+1) \times (3m/2+1)$ diagonal block and accumulation of the Householder reflectors, and (b) update of the off-diagonal block of the last $3m/2+1$ columns. Again, step (b) accounts for most of the computational work and can be done with the level-3 BLAS.

In phase II, the $m/2$ bulges that lie on the first $3m/2+1$ rows of the matrix are chased to the last $3m/2+1$ rows along the diagonal. In this phase, we set some integer k and regard the operation of chasing all the bulges by k rows as one block operation (Fig. 3). Since the sequence of the bulges occupies $3m/2+1$ rows and columns, this block operation involves $3m/2+k$ rows and columns. Then we divide the work in this block operation into three steps, that is, (a) bulge-chasing within the $(3m/2+k) \times (3m/2+k)$ diagonal block and accumulation of the reflectors, (b) update of the off-diagonal block of the $3m/2+k$ rows and (c) update of the off-diagonal block of the $3m/2+k$ columns. Again, steps (b) and (c) account for most of the work and they can be done with the level-3 BLAS. Braman et al. [5] shows that $k \sim \frac{3}{2}m$ is the best to minimize the computational work of phase II.

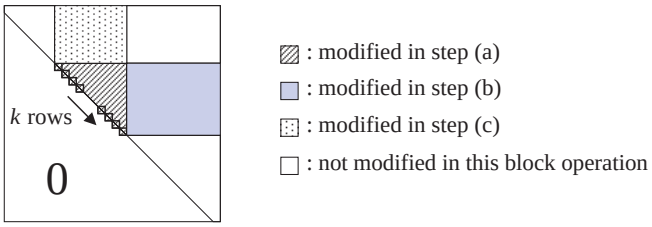

Fig. 3. Work in one block operation of phase II. The work is divided into (a) bulge-chasing in the diagonal block, (b) update of the off-diagonal rows and (c) update of the off-diagonal columns.

In addition to the work in phases I to III described above, there is work for computing the shifts. Also, when the matrix A_l becomes sufficiently small as a result of deflation, its eigenvalues are computed with the conventional double implicit shift QR algorithm. However, the computational work associated with them are much smaller than the work in phases I to III when $m \ll n$. Thus the small-bulge multishift QR algorithm can perform most of the work in the form of level-3 BLAS and exploit the potential performance of modern architectures. Numerical experiments on various test matrices show that it can attain up to three times speedup over DHSEQR on the Origin2000 [5].

2.2 Basic computational routines used in the algorithm

As is clear from the previous subsection, the small-bulge multishift QR algorithm consists of four types of basic computational routines as follows:

(A) Routines for chasing the bulges within the diagonal block. We denote the routines for phases I, II and III by BCHASE1, BCHASE2 and BCHASE3, respectively.
(B) level-3 BLAS routines for updating the off-diagonal block of the rows or columns. We can use DGEMM with TRANSA=TRANSB='N' for row updates and DGEMM with TRANSA='N' and TRANSB='T' for column updates.
(C) Two copy routines which we denote as COPY1 and COPY2. The former is used to copy the result of row updates back into the matrix, while the latter is used to copy the result of column updates back into the matrix.
(D) A Double implicit shift QR routine for computing the shifts or computing the eiegnvalues of A_l when A_l is sufficiently small. We use EISPACK routine HQR for this purpose.

In the hierarchical performance modeling to be explained in the next section, we use these routines as basic components.

2.3 The optimal block size

The attain high performance with the small-bulge multishift QR algorithm, it is critical to choose the optimal block size m. In general, as m becomes larger, the performance of the level-3 BLAS will increase because both the parallel granularity and the chance for data reuse increase. On the other hand, the cost of computing the shifts and the work of BCHASE1 to 3 relative to the total work will also grow with m. The optimal value of m is determined from these trade-offs and differs considerably depending on the architecture, the number of processors and the matrix size n. In fact, to obtain the best performance on the Origin2000, Braman et al. use $m = 60$ when $1000 \leq n \leq 1999$, $m = 116$ when $2000 \leq n \leq 2499$ and $m = 150$ when $2500 \leq n \leq 3999$ [5]. Our objective is to determine the optimal value of m for a given environment and problem size prior to execution using a performance prediction model.

3 Performance modeling

3.1 The hierarchical approach

To construct a performance model of the small-bulge multishift QR algorithm, we adopt the hierarchical modeling approach [16][15][14]. In this approach, we first construct execution time models for the basic components of the algorithm such as the level-3 BLAS routines and the double implicit shift QR routine based on the measurement of actual execution times. Then, the time consumed

by each call to these subroutines in the algorithm is estimated using the model and the input parameters. Finally, the execution time of the whole algorithm is predicted by accumulating these partial execution times. This methodology has been applied to the performance modeling of LU and QR decompositions and a BLAS-3 based tridiagonalizaion algorithm and has proved to give satisfactory results both in terms of accuracy and cost of prediction.

3.2 Modeling the performance of basic computational routines

From what we have stated in subsection 2.2, we need to construct execution time models for eight computational routines, namely, BCHASE1, BCHASE2, BCHASE3, DGEMM('N','N'), DGEMM('N','T'), COPY1, COPY2 and HQR. Here we take up the case of BCHASE2 to illustrate the process of modeling. This routine is used to chase the bulges within the diagonal block in phase II and has two input parameters m and k that determine the execution time. Our aim is to model the execution time of this routine as a function $f_{\mathrm{BCHASE2}}(m,k)$ of m and k. Note that the two-parameter model is necessary even when k is fixed. This is because the number of rows by which the bulges are chased in phase II is in general not a multiple of k and the remainder part must be processed by BCHASE2 with a smaller value of k.

To construct a model, we first measure the performance of BCHASE2 on grid points in the (m,k) plane. More specifically, we vary m from 10 to 150 with intervals of 10 and set k to one of the five values, $\frac{1}{5}m$, $\frac{2}{5}m$, $\frac{3}{5}m$, $\frac{4}{5}m$ and m. Then we approximate the execution time for a fixed value of k as a cubic function of m:

$$f_{\mathrm{BCHASE2}}(m,k) = f^{(k)}_{\mathrm{BCHASE2}}(m) = a_3^{(k)}m^3 + a_2^{(k)}m^2 + a_1^{(k)}m + a_0^{(k)}. \quad (1)$$

The coefficients $a_3^{(k)}, a_2^{(k)}, a_1^{(k)}$ and $a_0^{(k)}$ are determined by the least squares from the measured data. To obtain the value of $f_{\mathrm{BCHASE2}}(m,k)$ for k between the grid points, we compute the function values at the two adjacent grid points and use linear interpolation. It would be possible to use polynomial approximation also with respect to k, but we chose to use linear interpolation because it is more flexible and can approximate the function well even when it exhibits somewhat irregular behavior due to cache miss.

The performance modeling of other routines can be done in much the same way. In fact, COPY1 and COPY2 also have two parameters that affect the execution time. DGEMM('N','N') and DGEMM('N','T') also have only two parameters since in our algorithm, the multiplier U is a square matrix. BCHASE1, BCHASE3 and HQR have only one parameter and therefore their performance modeling is easier.

3.3 Modeling the performance of the whole algorithm

Once execution time models of the basic computational routines have been constructed, we can use them to predict the execution time of the small-bulge multi-shift QR algorithm. The conventional approach to this is to derive an analytical

expression for the total computational work executed by each routine and then calculate the time consumed by each routine using the corresponding performance models. However, it is difficult to obtain accurate results with this approach because the relation between the execution time and the computational work is usually far from linear.

We therefore take an alternative approach. We first write functions named BCHASE1_TIME, DGEMM_TIME, COPY1_TIME, HQR_TIME and so on. These functions have the same input parameters as BCHASE1, DGEMM, COPY1 and HQR, respectively, but instead of doing actual computation, estimate the execution time for given input parameters using the performance model for the corresponding routine and return it. Next we rewrite the main program of the small-bulge multishift QR algorithm so that the calls to the computational routines are replaced with the calls to the corresponding time estimation routines and the estimated execution times are accumulated. Then, by executing the rewritten program, we can obtain estimated execution time of the small-bulge multishift QR algorithm for a given value of n and m. This approach has been successfully applied to the performance modeling of a BLAS-3 based tridiagonalization algorithm. [18] reports that it can predict the execution time of this algorithm for various matrix sizes and block sizes on different architectures within errors of 5 to 10%.

Since the QR method is an iterative method, we have to address one problem that does not exist in the case of the tridiagonalization algorithm. We have to specify how many times the main loop is executed before convergence and in what manner the deflations occur. Kressner [22] observes that in average, deflation occurs after every four multishift sweeps and at each deflation, approximately $m \times m$ trailing submatrix is isolated. We confirmed this observation through our experiments on various matrices and adopt it as a model of convergence behavior in our performance estimation program.

Our performance model takes fully into account the nonlinearity between the execution time and computational work of the basic computational routines. Thus it is expected to predict the total execution time accurately, except for the variation due to variation in the number of iteration. The cost of prediction is proportional to the number of calls to the basic computational routines in the algorithm and is $O(n^2/m^2)$. This is negligible compared with the computational work needed for actual execution of the algorithm, which is $O(n^3)$. Note also that our performance model can be applied to shared-memory parallel programs without difficulty as long as parallelization is done within the basic computational routines.

4 Experimental results

4.1 Computational environments

To evaluate the effectiveness of our performance model, we performed experiments on two platforms, namely, a 2way SMP machine with 2.0GHz PowerPC G5 processors and a 4way SMP machine with 1.8GHz Opteron processors.

For the G5 machine, we used IBM XL Fortran with options -O3 -qsmp=omp -qarch=ppc970 -qtune=ppc970 and the GOTO BLAS. For the Opteron machine, we used GNU f77 compiler with option -O4 and the GOTO BLAS. The routines BCHASE1, BCHASE2, BCHASE3, COPY1 and COPY2 were written from scratch and the EISPACK routine HQR is used for computing the shifts and for computing the eigenvalues of the matrix when it becomes sufficiently small. Parallelization is done only within the GOTO BLAS.

4.2 Performance prediction

To construct the performance model, we first measured the execution times of the basic computational routines on both machines for various values of input parameters. For BCHASE1, BCHASE2, BCHASE3 and HQR, we varied m from 10 to 150 with intervals of 10. Performance measurement of BCHASE2 was done as described in subsection 3.2. For DGEMM('N','N') and DGEMM('N','T'), the size of the square multiplier matrix was varied from 10 to 300 with intervals of 10, while the number of columns in the multiplicand matrix was varied from 100 to 1000 with intervals of 100. The execution times of all the routines except for DGEMM were measured on 1 processor. The execution times of DGEMM were measured on 1 and 2 processors (in the case of PowerPC G5) or on 1 and 4 processors (in the case of Opteron). Based on these measurements, we built an execution time model for each routine on each platform following the prescription of subsection 3.2.

Next we constructed a performance model for the small-bulge multishift QR algorithm following the methodology stated in subsection 3.3 and compared the predicted execution time with the actual execution time. We varied the matrix size m from 1000 to 8000 and varied the number of shifts m from 30 to 120. The value of k was set equal to m. As test matrices, we generated random matrices whose elements follow the uniform distribution in $[-1, 1]$ and transformed them to Hessenberg by Householder transformation.

The results on the PowerPC G5 are shown in Table 1 and Fig. 4 for the 1-processor case and in Table 2 and Fig. 5 for the 2-processor case. It is clear that the model generally overestimate the actual execution time. This is because the test matrices used here require smaller number of iterations than we assumed in subsection 3.3. In fact, the average number of multishift QR sweeps needed to isolate an (approximately) $m \times m$ small submatrix was between 3 and 4. However, when we turn our attention to the relative execution time, defined as the ratio of the execution time to the shortest execution time over all m, we see that the model reproduces the behavior of the actual execution time fairly well (See Figs. 4 and 5). This is sufficient for determining the optimal value of m.

We also show the results on the 4way Opteron SMP machine in Table 3 and Fig. 6. In this case, again, the model generally overestimates the execution time, but predicts the relative execution time as a function of m fairly well.

Table 1. Actual (above) and predicted (below) execution times (in sec.) of the small-bulge multishift QR algorithm (PowerPC G5, 1CPU).

n	$m = 30$	$m = 60$	$m = 90$	$m = 120$
1000	5.84	5.27	5.12	6.75
	8.28	6.79	6.77	7.82
2000	42.21	33.75	32.83	34.47
	53.40	40.53	39.59	46.09
4000	356.85	267.65	236.20	260.97
	393.86	288.60	267.22	296.86
8000	3073.24	2496.06	2270.95	2129.30
	3075.62	2180.10	1969.56	2116.30

Table 2. Actual (above) and predicted (below) execution times (in sec.) of the small-bulge multishift QR algorithm (PowerPC G5, 2CPU).

n	$m = 30$	$m = 60$	$m = 90$	$m = 120$
1000	4.51	3.94	4.51	6.02
	7.06	5.80	5.92	7.07
2000	33.33	24.74	24.99	29.25
	42.86	31.37	31.06	37.69
4000	274.19	190.05	176.53	207.83
	307.30	208.87	191.45	220.91
8000	2448.36	1812.72	1676.86	1807.11
	2370.45	1520.44	1328.71	1463.86

4.3 Optimal block size selection

As can be seen clearly from Figs. 4 through 6, the value of m that gives the shortest execution time varies considerably depending on the matrix size and the architecture. Generally speaking, the optimal value of m increases with the matrix size. Also, the optimal value differs widely with the matrix size in the case of the 4way Opteron machine, while it is rather insensitive in the case of G5. Figs. 4 through 6 show that these tendencies are represented by our model well. In fact, our model succeeds in predicting the optimal value of m in 9 cases of the total 12 cases presented here. Even when it fails to predict the correct m, it can be seen that the execution time using predicted m differs only slightly from the shortest execution time. In addition, the time needed to predict the execution time for all value of m for given n is less than a fraction of a second. Thus we can conclude that our model can be used effectively for choosing the optimal block size in the small-bulge multishift QR algorithm.

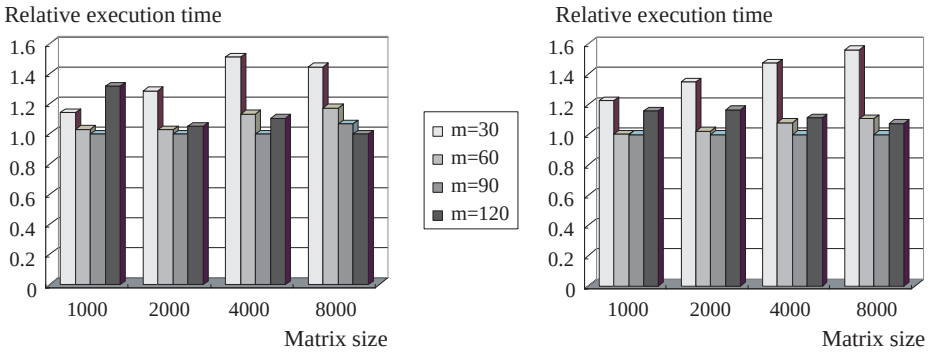

Fig. 4. Actual (left) and predicted (right) relative execution times of the small-bulge multishift QR algorithm (PowerPC G5, 1CPU).

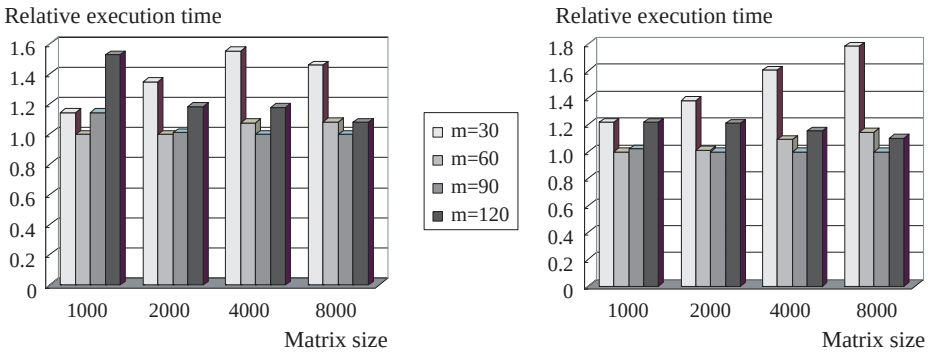

Fig. 5. Actual (left) and predicted (right) relative execution times of the small-bulge multishift QR algorithm (PowerPC G5, 2CPU).

5 Conclsion

In this paper, we construct a performance model for the small-bulge multishift QR algorithm proposed by Braman, Byers and Mathias. Our model has a hierarchical structure that naturally arises from the structure of the original algorithm and given the matrix size n, the number of simultaneous shifts m and the performance data of the basic components of the algorithm, such as the level-3 BLAS routines and the double implicit shift QR routine, predicts the total execution time. Experiments on SMP machines based on PowerPC G5 and Opteron processors show that the variation of the execution time as a function of m predicted by the model agrees well with the measurements. Thus our model can be used to automatically select the optimal value of m for a given matrix size on a given architecture.

Table 3. Actual (above) and predicted (below) execution times (in sec.) of the small-bulge multishift QR algorithm (Opteron, 4CPU).

n	$m = 30$	$m = 60$	$m = 90$	$m = 120$
1000	3.58	3.80	4.37	5.56
	4.26	4.32	5.02	6.00
2000	24.32	20.86	24.67	24.18
	27.38	24.39	26.26	29.95
4000	189.38	144.14	141.81	145.77
	190.86	158.50	157.95	168.69
8000	1522.24	1069.79	1021.83	1014.12
	1419.33	1116.15	1066.48	1089.39

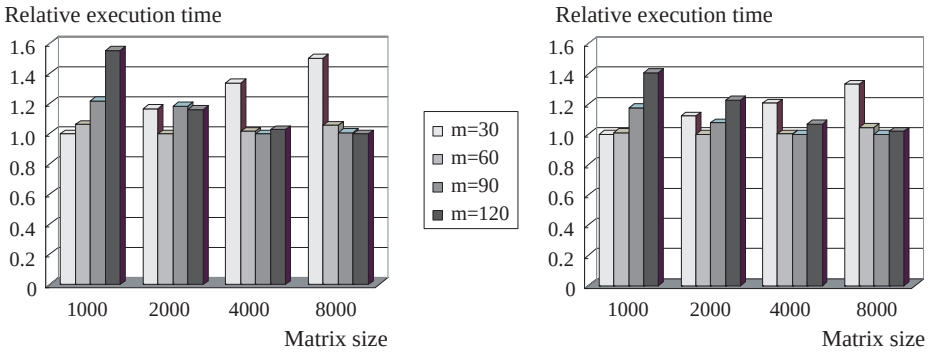

Fig. 6. Actual (left) and predicted (right) relative execution times of the small-bulge multishift QR algorithm (Opteron, 4CPU).

Future works include extension of this model to a more efficient parallel implementation of the small-bulge multishift QR algorithm, where the bulge chasing in the diagonal block is overlapped with the update of the off-diagonal blocks. In this case, the number of parameters to optimize will increase and we will need an efficient search algorithm that can find near-optimal parameters without an exhaustive search of all possible candidates. As another direction of research, we are planning to extend the modeling methodology used in this work to distributed-memory parallel programs.

Acknowledgements

I would like to thank Professor Reiji Suda, Professor Takahiro Katagiri, Professor Toshitsugu Yuba, Professor Toshiyuki Imamura, Dr. Ken Naono and other members of the Auto-Tuning Study Group for fruitful discussion. This work is partially supported by the Ministry of Education, Science, Sports and

Culture, Grant in Aid for Scientific Research on Priority Areas, "i-explosion" (No. 18049014), Grant-in-aid for Scientific Research (C)(No. 18560058) and Grant-in-Aid for the 21st Century COE "Frontiers of Computational Science".

References

1. Francis, J.G.F.: The QR transformation. a unitary analogue to the LR transformation. I. Comput. J. **4** (1961/1962) 265–271
2. Francis, J.G.F.: The QR transformation. II. Comput. J. **4** (1961/1962) 332–345
3. Kublanovskaya, V.N.: On some algorithms for the solution of the complete eigenvalue problem. U.S.S.R. Comput. Math. and Math. Phys. **3** (1961) 637–657
4. Bai, Z., Demmel, J.: On a block implementation of Hessenberg QR iteration. Int. J. of High Speed Computing **1** (1989) 97–112
5. Braman, K., Byers, R., Mathias, R.: The multishift QR algorithm. part I: Maintaining well-focused shifts and level 3 performance. SIAM Journal on Matrix Analysis and Applications **23** (2002) 929–947
6. Dubrulle, A.: The multishift QR algorithm: Is it worth the trouble? Palo Alto Scientific Center Report G320-3558x, IBM Corp. (1991)
7. Henry, G., Watkins, D.S., Dongarra, J.: A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. SIAM J. Sci. Comput. **24** (2002) 284–311
8. Watkins, D.S.: Bidirectional chasing algorithms for the eigenvalue problem. SIAM J. Matrix Anal. Appl. **14** (1993) 166–179
9. Watkins, D.S.: Shifting strategies for the parallel QR algorithm. SIAM J. Sci. Comput. **15** (1994) 953–958
10. Demmel, J.W.: Applied Numerical Linear Algebra. SIAM (1997)
11. Golub, G.H., van Loan, C.F.: Matrix Computations. Third edn. Johns Hopkins University Press (1996)
12. Watkins, D.S.: The transmission of shifts and shift blurring in the QR algorithm. Linear Algebra and Its Applications **241/243** (1996) 877–896
13. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK User's Guide. SIAM (1992)
14. Dackland, K., Kågström, B.: A hierarchical approach for performance analysis of ScaLAPACK-based routines using the distributed linear algebra machine. In: Proceedings of Workshop on Applied Parallel Computing in Industrial Computation and Optimization (PARA96). Number 1184 in Lecture Notes in Computer Science, Springer-Verlag (1996) 187–195
15. Cuenca, J., Gimenez, D., Gonzalez, J.: Architecture of an automatically tuned linear algebra library. Parallel Computing **30** (2004) 187–210
16. Cuenca, J., Garcia, L.P., Gimenez, D.G.: Empirical modelling of parallel linear algebra routines. In: Proceedings of the 5th International Conference on Parallel Processing and Applied Mathematics (PPAM2003). Number 3019 in Lecture Notes in Computer Science, Springer-Verlag (2004) 169–174
17. Katagiri, T., Kuroda, H., Kanada, Y.: A methodology for automatically tuned parallel tri-diagonalization on distributed memory parallel machines. In: Proceedings of VecPar2000, Faculdade de Engenharia da Universidade do Porto, Portugal (2000) 265–277

18. Yamamoto, Y.: Performance modeling and optimal block size selection for a BLAS-3 based tridiagonalization algorithm. In: Proceedings of HPC-Asia 2005, Beijing (2005) 249–256
19. Dongarra, J., Eijkhout, V.: Self-adapting numerical software for next generation applications. International Journal of High Performance Computing Applications **17** (2003) 125–131
20. Whaley, R., Petitet, A., Dongarra, J.: Automated empirical optimizations of software and the ATLAS project. Parallel Computing **27** (2001) 3–35
21. Bilmes, J., Asanovic, K., Chin, C.W., Demmel, J.: Optimizing matrix multiply using PhiPAC: a portable, high-performance, ANSI-C coding methodology. In: Proceedings of the 11th International Conference on Supercomputing, Vienna (1997) 340–347
22. Kressner, D.: Numerical Methods for General and Structured Eigenvalue Problems. Springer-Verlag (2005)