Vol.2009-MPS-74 No.9
2009/7/13

Speed-up in mdLVs by Limitation in Computational Number of Shift

Masami Takata[†] Kinji Kimura[††]
and Yoshimasa Nakamura[††]

In a singular value computation using the mdLVs (modified discrete Lotka-Volterra scheme with shift) scheme, most of the computational time is that for calculating shifts. In this paper, we aim to perform a speed-up of the mdLVs by using a limitation of the computational number of shifts. In the proposed algorithm for speed-up, a shift, which is calculated before some iteration, is successively used in update of shifts. The proposed algorithm can be expanded into a parallel algorithm for distributed memory systems. To evaluate the proposed sequential algorithm, we experiment using 100 1,000-dimensional matrices. As a result, since data on shifts is updated by using an old and safety shift, accuracy of singular values can be improved. Moreover, since the computational number of shifts is limited, computational time of the proposed algorithm becomes smaller.

シフトの計算回数の制限を用いた mdLVs 法の高速化

高田雅美[†] 木村欣司[††] 中村佳正[††]

mdLVs（modified discrete Lotka-Volterra scheme with shift）法を用いた特異値分解では，シフトの計算時間が全体の計算時間の大部分を占める．本論文において，我々は，シフトの計算回数を制限することによって mdLVs 法の高速化を試みる．この高速化アルゴリズムでは，数イタレーション前に計算されたシフトを用いて，各要素値の更新を行う．なお，このアルゴリズムは，分散メモリ型並列計算機用の並列アルゴリズムに容易に改良することが可能である．提案する逐次アルゴリズムの性能を調べるために，100 個の 1,000 次行列を用いて実験を行う．実験の結果，数イタレーション前のシフトを用いても，正定値が保たれるため，特異値の精度が向上することを確認した．また，シフトの計算回数が減少するため，全体の計算時間が小さくなった．

1. Introduction

Singular value decomposition is known as effective method for investigation of feature in rectangular matrices. Singular value decomposition is a matrix decomposition of a given rectangular matrix into a product of nonnegative definite diagonal matrix and two orthogonal matrices, where elements of the diagonal matrix are singular values and the orthogonal matrices consist of left and right singular vectors. In general, any rectangular matrix is decomposed into a product of an upper bidiagonal matrix and orthogonal matrices [1][2]. Note that singular values of the rectangular matrix are equal to those of the bidiagonal matrix. Therefore, a singular value decomposition of the bidiagonal matrix can be chosen instead of that of the rectangular matrix.

The QRs scheme [1][2][3][4][5] and the I-SVD (Integrable-Singular Value Decomposition) scheme [6][7][8][9] are known as today's standard singular value decomposition schemes for bidiagonal matrices. The computational time in the I-SVD scheme is shorter than that in the QRs scheme [9]. The orthogonality of singular vector in the I-SVD scheme is somewhat worse than that in the QRs scheme. However, the errors related to singular values and singular vectors in the I-SVD scheme are smaller than those in the QRs scheme [8].

An increase of data volume for data search system [10] and image processing [11], for example, which are calculated by using the singular value decomposition, brings us data matrices of large dimension. Consequently, singular value decompositions with high-speed should be developed. Hence the I-SVD scheme as well as the QRs must be improved to a parallel scheme.

In the I-SVD scheme, once singular values are calculated by using the mdLVs (modified discrete Lotka-Volterra scheme with shift) scheme, then singular vectors are obtained by using a singular vector computation with the twisted factorization [6][7]. In the mdLVs scheme, most of the computational time is that for calculating shifts. Consequently, we aim to perform a speed-up of the singular value computation by using limitation in number of shift calculations. In this paper, we propose an improvement of the mdLVs scheme. Note that this proposed algorithm is expanded into a parallel one of the mdLVs.

In Section 2, we explain the singular value computation by using the mdLVs scheme. In Section 3, we propose an algorithm, which has a limitation of the number of shifts. In Section 4, we valid accuracy of singular values and computational time of the proposed

[†] Nara Women's University
奈良女子大学
[††] Kyoto University
京都大学

Vol.2009-MPS-74 No.9
2009/7/13

algorithm. In Section 5, we improve the proposed sequential algorithm into a parallel one for distributed memory systems.

2. Modified discrete Lotka-Volterra scheme with shift

In Subsection 2.1, we give a summary of the singular value computation based on the dLV system. In Subsection 2.2, we show the outline of the mdLVs scheme. In Subsection 2.3, a programming of the mdLVs scheme is described shortly.

2.1 Singular value computation based on the discrete Lotka-Volterra system

In the mathematical biology, the LV (Lotka-Volterra) system is known as a fundamental prey–predictor model. In some case, the LV system is a completely integrable dynamical system which has explicit solutions and sufficiently many conservation laws. A time discretization

$$u_k^{(n+1)} = \frac{1+\delta^{(n)}u_{k+1}^{(n)}}{1+\delta^{(n+1)}u_{k-1}^{(n+1)}}u_k^{(n)} \tag{1}$$

of the LV system is known (cf. [6]). This system also has explicit solution and many conservation laws. Therefore, it is called the integrable discrete LV (dLV) system. Here k ($k = 1, 2,\dots, 2M-1$) indicates the k–th species and the discrete time n ($n = 0, 1, 2,\dots$) corresponds to an iteration number of the scheme, $u_k^{(n)}$ is the value of u_k at n, and the arbitrary nonzero number $\delta^{(n)}$ is a discrete step-size, and M is the dimension of a matrix. Let the initial value $u_k^{(0)}$ be positive. In the case where $\delta^{(n)} > 0$, any subtraction and division by zero do not occur in Eq.(1) and $u_k^{(n)}$ is always positive. Consequently, canceling and numerically instability do not emerge. Let us note here we do not need treat negative numbers in singular value computations.

The boundary condition and the initial condition are

$$u_0^{(n)} \equiv 0, u_{2M}^{(n)} \equiv 0, \tag{2}$$

$$u_k^{(0)} = \frac{(b_k)^2}{1+\delta^{(0)}u_{k-1}^{(0)}}, \tag{3}$$

respectively. Here b_{2i-1} (i: $1 \le i \le M$) and b_{2i} are diagonal and upper–subdiagonal elements of the $M \times M$ bidiagonal matrix B, respectively. When $n \to \infty$, $u_{2i-1}^{(n)}$ and $u_{2i}^{(n)}$ converge to the square of the i–th singular value σ_i and 0, respectively. Thus the dLV system gives rise to a stable scheme for computing singular values [6].

2.2 Speed up by means of the shifted discrete Lotka-Volterra scheme

The mdLVs scheme, the integrable dLV system with shift, can compute singular values in higher speed. The mdLVs scheme is formulated as follows [7].

Let us introduce new elements $w_k^{(n)}$ and $v_k^{(n)}$ by

$$w_k^{(n)} = u_k^{(n)}(1+\delta^{(n)}u_{k-1}^{(n)}), \tag{4}$$

$$v_k^{(n)} = u_k^{(n)}(1+\delta^{(n)}u_{k+1}^{(n)}). \tag{5}$$

By Eq.(3), the initial $w_k^{(0)}$ is just $b_k^{\,2}$. The shifted integrable dLV system is defined by adding a shift $S^{(n)}$, which is defined as $0 \le S^{(n)} < \sigma_m^2$ where σ_m is the minimal singular value of B, to Eq.(1). Namely,

$$w_{2i-1}^{(n+1)} = v_{2i-1}^{(n)} + v_{2i-2}^{(n)} - w_{2i-2}^{(n+1)} - S^{(n)}, \tag{6}$$

$$w_{2i}^{(n+1)} = \frac{v_{2i-1}^{(n)}v_{2i}^{(n)}}{w_{2i-1}^{(n)}}. \tag{7}$$

In general, the convergence is accelerated by enlarging $S^{(n)}$. However, since the positivity of $u_k^{(n)}$ may be destroyed by a larger $S^{(n)}$, it causes a numerical instability. It is proved in [7] that $u_k^{(n)} > 0$ if and only if $0 \le S^{(n)} < \sigma_m^2$. Hence the shift $S^{(n)}$ can be determined by using the Johnson bound [12] for estimating σ_m.

In the Johnson bound, $S^{(n)}$ is the minimum vlaue among $s_i^{(n)}$. $s_i^{(n)}$is calculated as follows.

$$\sqrt{s_i^{(n)}} = \sqrt{v_{2i-1}^{(n)}} - 0.5*\left(\sqrt{v_{2i}^{(n)}} + \sqrt{v_{2i-2}^{(n)}}\right). \tag{8}$$

In the Johnson bound, a shift $S^{(n)}$ becomes to quadratic of an approximate number in the minimal singular value σ_m

2.3 Algorithm for singular value computation based on the Lotka-Volterra system

Each iteration in an algorithm for the mdLVs scheme is described as follows.

(i) $u_k^{(n)}$ is calculated from $w_k^{(n)}$ by Eq.(4).
(ii) $v_k^{(n)}$is calculated from $u_k^{(n)}$ by Eq.(5).
(iii) $S^{(n)}$ is calculated by Eq.(8).
(iv) After checking $S^{(n)}$, $w_k^{(n+1)}$ is calculated.

- In the case of a valid $S^{(n)}$, $w_k^{(n+1)}$ is calculated from $v_k^{(n)}$ by Eq.(6) and (7).
- In other case, $w_k^{(n+1)} = v_k^{(n)}$.

(v) In the case that $w_{2i}^{(n+1)}$ is extremely smaller than $w_{2i-1}^{(n+1)}$, SPLIT or a deflation of dimension size are done.

SPLIT, which divides the matrix to two parts, and the deflation are defined in [11].

Arrays of the algorithm are calculated as follows. In Step(i), the array $U = (u_1^{(n)}, u_2^{(n)},\ldots, u_{2M-1}^{(n)})$ is calculated from the array $W = (w_1^{(n)}, w_2^{(n)},\ldots, w_{2M-1}^{(n)})$, where n represents the iteration number. Since the data at each n does not keep, each array consists of one-dimensional array corresponding to under suffix. In Step(ii), the array $V = (v_1^{(n)}, v_2^{(n)},\ldots, v_{2M-1}^{(n)})$ is calculated from U. In Step(iii), the sift $S^{(n)}$ is calculated from V. By using a valid $S^{(n)}$, W is overwritten by V in Step(iv).

In the loops of Step(i) and (ii), U and V are updated in ascending order of k. For the update of $u_k^{(n)}$, we use $w_k^{(n)}$ and $u_{k-1}^{(n)}$ in Step(i). For the update of $v_k^{(n)}$, we need $u_k^{(n)}$ and $u_{k+1}^{(n)}$ in Step(ii). For the calculate of $s_i^{(n)}$, we use $v_{2i-2}^{(n)}$, $v_{2i-1}^{(n)}$ and $v_{2i}^{(n)}$ in Step(iii). Then, $S^{(n)}$ is calculated from all $s_i^{(n)}$. For the update $w_k^{(n+1)}$ from $w_k^{(n)}$ in Step(iv), W is updated in ascending order of k.

3. Proposed algorithm

The shift calculation in Step(iii), in which the Johnson bound is calculated, needs a lot of calculation for square root. Therfore, the most of the computatinal time in the mdLVs scheme is Step(iii). Hence, to speed up the mdLVs scheme, we limit number of shift calculations in Step(iii).

Each iteration in a proposed algorithm is described as follows.

(I) $u_k^{(n)}$ is calculated from $w_k^{(n)}$ by Eq.(4).

(II) $v_k^{(n)}$is calculated from $u_k^{(n)}$ by Eq.(5).

(III) In the case of $L=Pend$, $S^{(n)}$ is calculated by Eq.(8).

(IV) Blanch instruction in L.

- In the case of $L=1+Pend-Nact$, $w_k^{(n+1)}$ is calculated, after checking $S^{(n)}$.
  - ➢ In the case of a valid $S^{(n)}$, $w_k^{(n+1)}$ is calculated from $v_k^{(n)}$ by Eq.(6) and (7).
  - ➢ In other case, $w_k^{(n+1)} = v_k^{(n)}$.
- In other case, $w_k^{(n+1)} = v_k^{(n)}$.

(V) Blanch instruction in L.

- In the case of $L=1+Pend-Nact$, $L=Pend$.
- In other case, $L=L-1$

(VI) In the case that $L=Pend-1$ and $w_{2i}^{(n+1)}$ is extremely smaller than $w_{2i-1}^{(n+1)}$, SPLIT or a deflation of dimension size are done.

- Once SPLIT is done, then *Nact* is checked and changed if necessary.
- If the deflation occurs and dimension size is smaller than $2*M*(Pend-1)/Nproc+1$, $Pend=Pend-1$.

Here, L means limit of computational number in the shift $S^{(n)}$, *Pend* is initialized as *Nproc* (*Nproc*: $1 \leq Nproc \leq M$) which is a partition number of strip-mining [13] in the matrix B, and *Nact* is a number of partition matrices while singular values are computed. Initial L and *Nact* are set to *Nproc*.

In this algorithm, the shift $S^{(n)}$ is calculated at *Nact* times. Hence, a computational time becomes shorter. In Step(IV), $w_{2i-1}^{(n+1)}$ is updated by using the shift $S^{(n-Nact)}$, which is calculated before *Nact* iteration. When iteration number n increases, $u_{2M-1}^{(n)}$ is approximated to the minimal singular value. Consequently, Step(IV) may be furthermore accelerated convergence in $u_{2M-1}^{(n)}$ and $u_{2M}^{(n)}$. In Step(V), L is updated. Step(III) and SPLIT in Step(VI) is done at the same iteration number.

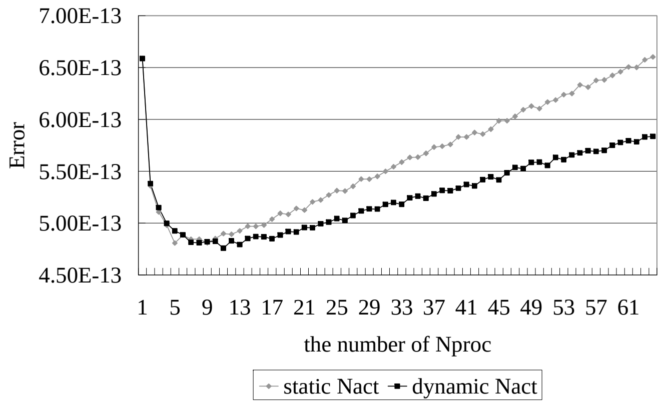

Figure 1 Error comparison of static *Nact* with dynamic one

4. Numerical experiments

We examine efficiency of the proposed sequential algorithm in Section 3.

For comparing numerical accuracy, we use a computer with CPU: Intel (R) Core (TM) 2 (*2.66*GHz), Memory: *8,192*MB, L1D: *32*KB, L1I: *32*KB, L2: *4,096*KB and *64*bit registers, on which Fedora Core 7 (Linux 2.6.21) is installed. For the compiler, GNU 4.1.2 with option “O3” is adopted.

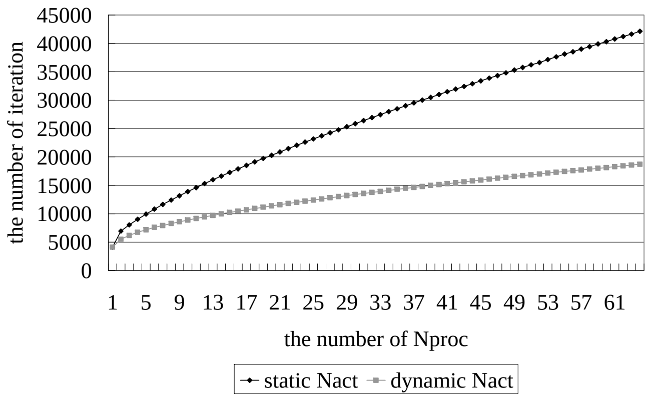

Figure 2 comparison of iteration in static *Nact* with that in dynamic one

In the proposed algorithm, $\delta^{(n)}=1$.

To discuss errors and computational time in singular value computation, we construct *100 1,000*-dimensional test matrices whose singular values are exactly known by using the Golub-Kahan-Lanczos method [2][14]. *1,000* singular values of each matrix are randomized on the interval *[1, 500]*.

In Subsection 4.1, we discuss about errors of singular values. In Subsection 4.2, we examine computational time in the proposed algorithm.

4.1 Experiments for accuracy

Figure 1 shows a comparison of static *Nact* (*Nact*=*Nproc*) with dynamic *Nact*, which includes the proposed algorithm. Here, the interval of *Nproc* is $1 \le Nproc \le 64$. Each error of the static *Nact* in *Nproc* has the same trend as the dynamic one. In the case of small *Nproc*, the error in the proposed algorithm is smaller than that in the original algorithm, which is the same as *Nproc=1*. It is caused that $w_k^{(n+1)}$ is updated by using the shift $S^{(n-Nact)}$, which has been calculated at *n-Nact*-th iteration. In the case of large *Nproc*, the error is nearly equal to the error in the original algorithm. This reason is too much increase number of iteration. In the case of $20 \le Nproc$, errors is more large as *Nproc* is sized up. Since many $w_k^{(n+1)}$ is updated without a shift, the convergence, which means the decrease iteration number, is not accelerated. Consequently, computational errors, which occur at each iteration, is stored up, and the accuracy in the case of $20 \le Nproc$ becomes worse.

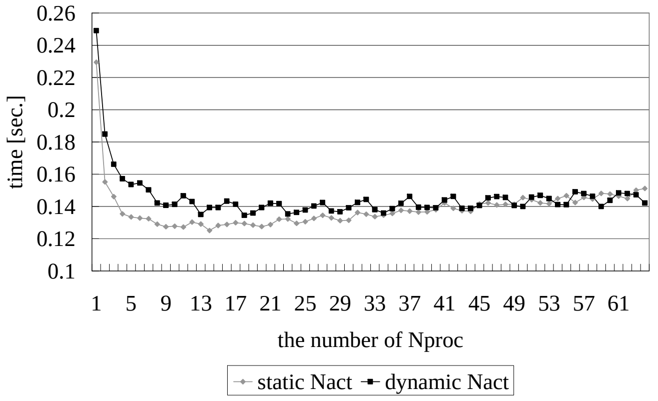

Figure 4 Time comparison of static *Nact* with dynamic one.

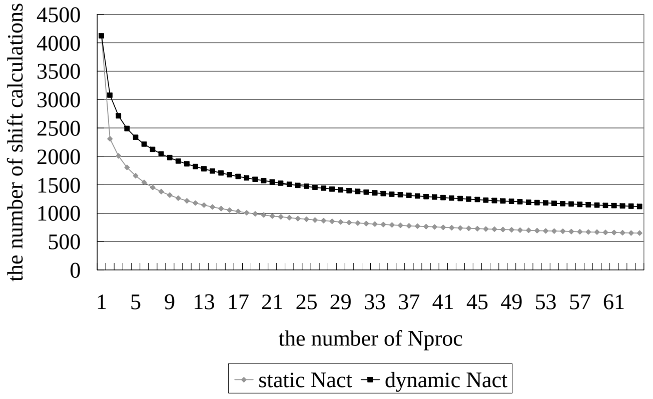

Figure 3 Comparison of the number of shift calculation in static *Nact* with that in dynamic one.

Figure 2 shows a comparison of static *Nact* with dynamic one. The debasement of accuracy in the static *Nact* is more accerelated than that in the dynamic one. In the case of the static *Nact*, the number of the shift calculation is not changed in any dimension size. Therefore, when dimension size becomes small because of SPLIT or a defration, the limitation in number for the shift calculation is too large. Consequently, it overbound that $w_k^{(n+1)}$ is updated without a shift. Hence, the dynamic *Nact* in the proposed algorithm is effective.

By the results in Figure 1 and Figure 2, the proposed sequential algorithm is useful.

Table 1 Time of each *Nproc*. [sec.]

Nproc	Average	Maximum	Minimum
1	10.77	13.13	7.13
2	9.96	11.60	6.38
4	9.12	10.83	5.66
8	8.53	10.00	5.49
16	7.95	9.68	5.27
32	7.58	9.86	4.58
64	7.38	9.05	4.83

Through *Nproc* is set to a suitable number, which may be obtained from a relationship to a dimension size, the accuracy of the proposed algorithm becomes better.

4.2 Experiments for computational time

Figure 3 shows the average time in each *Nproc* ($1 \leq Nproc \leq 64$). In the mdLvs scheme, the most of the computational time is a shift calculation. Therefore, computational time in the proposed algorithm, which has the limitation of the number in the shift calculation, is shorter than that in the original algorithm. In Figure 3, the relationship between *Nproc* and the computational time is drown using bodoly-dented line. It is caused that dimension size is changed by SPLIT.

Figure 4 shows a comparison of static *Nact* with dynamic *Nact*, in the case of $1 \leq Nproc \leq 64$. Number of the shift calculation in the static *Nact* is smaller than that in the dynamic one, since the number in dynamic *Nact* is changed at Step(VI). However, in Figure 3, the computational time in the static *Nact* is nearly equal to that in the dynamic one. It is caused that $w_k^{(n+1)}$ is updated without a shift in a lot of case. An adoption of a shift accerelates a convergence to singular values. Thus, when a shift is not adopted, iteration number increases and accuracy becomes wrong. Consequently, the dynamic *Nact* is effective in these test matrices.

To evaluate matrices with a large dimensional size, we construct 100 10,000-dimensional test matrices, of which elements are randomized on the interval from 1 to 100 and any true singular values can not be obtained. Table 1 shows computational time in the matrices, where *Nproc=1, 2, 4, 8, 16, 32, 64*. More *Nproc* is large, more computational time is small. Consequently, computational time in a singular valu calculation becomes shorter by using the proposed sequential algorithm.

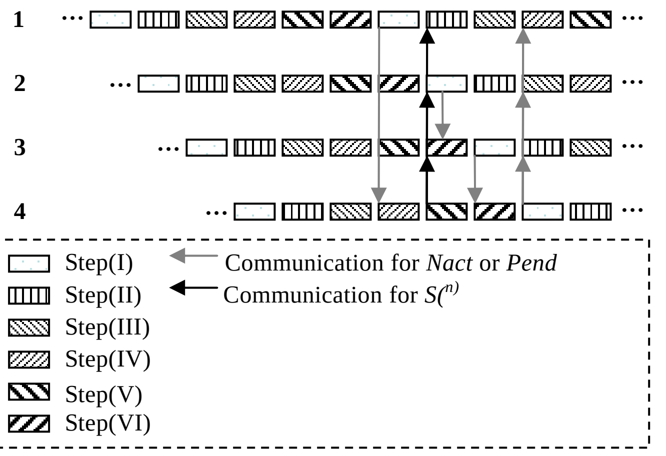

Figure 5 Concept of a parallel algorithm

5. An expansion for parallel system with distributed memory

In the singular vector computation of the I-SVD scheme, each singular vector is computed independently. Therefore, the singular vector computation can be paralleled. On the other hand, since the computation using the original mdLVs scheme has dependency relation, this unfits for parallelization.

On the other hand, the algorithm proposed in Section 3 can be improved a parallel algorithm for distributed memory systems. We adopt pipline progress for the parallel algorithm. Figure 5 shows a concept in the case of 4 processors, where each number means a processor id. Note that computational time in each Step is difference, although each square for Steps is drawn at the same size in Figure 5.

In Step(I), $u_k^{(n)}$ is calculated using $u_{k-1}^{(n)}$. In Eq.(6) and (7) of Step(IV), to update $w_k^{(n+1)}$, we need $w_{k-1}^{(n+1)}$. Consequently, these step should be calculated using pipeline progress.

In Step(II) and $s_i^{(n)}$ of Step(III), each element can be calculated independently. And also, when $w_k^{(n+1)}$ is updated without the shift $S^{(n)}$, each element does not has relationship. Consequently, these calculations can be paralleled.

Let *Nproc* be a processer number. The calculation for $S^{(n)}$ in Step(III) is calculated on the *Pend*-th processor. Then, the calculated shift $S^{(n)}$ is sent to each processor by using asynclonus communications. The calculated shift $S^{(n)}$ is used in Step(IV) after *Nact*-th iteration. Therefore, each processor has only to recieved until Step(IV) at *Nact*-th iteration. Since we choose asynclonus communications, the other steps can be carried on its calculations in each processor.

In Step(VI), SPLIT is occured on arbitraly processors. Thus, *Nact* may be changed on all processors. We adopt pipline for parallel systems. Consequently, the *Nact*, which is changed on an arbitraly processor, is sent to the *Pend*-th processor. Then, the *Pend*-th processor is sent the changed *Nact* to the *1+Pend-Nact*-th or later processors.

In Step(VI), a defration is occured only the *Pend*-th processor. Therefore, when *Pend* is chnaged, the *Pend+1*-th processor is sent the changed *Pend* to each processor. In the another case, the defration have no effect on calculations on processors other than *Pend*-th one.

To avoid communication barriers, the canged *Nact* and *Pend* must be finished communicating until *Nact* -th iteration like as Step(III). Consequently, Step(VI) should be calculated only in the case of *L=Pend-1*.

In this parallel algorithm, errors may be equal to the case of the sequential proposed algorithm.

6. Conclusion

A shift calculation occupies most of the computational time of the mdLVs scheme. To decrease the computational time, we propose an improvement of the mdLVs, which has a limitation of the number of shifts. Moreover, we expand it to a parallel algorithm, to which pipeline progress is adopted, for distributed memory systems. To evaluate the proposed sequential algorithm, we experiment using *100 1,000*-dimentional matrices. Since $w_k^{(n+1)}$ is updated with the shift $S^{(n-Nact)}$, which has been calculated before *Nact* iteration, we determined that errors in the proposed algorithm are smaller than that in the original algorithm (*Nproc=1*). In the proposed algorithm, *Nact* is changed dynamicaly at each iteration. Therefore, we compared with a static *Nact* and the dynamic one. By using the proposed sequential algorithm, computational time becomes about half. In this regard, we finded out that extrimery large number of *Nproc* does not influence decrease of computational time. It is caused that the calculation number to update $w_k^{(n+1)}$ without shift increases. In these results, we conclude that the proposed algorithm is effective.

In a future work, we should develop a parallel program based on the proposed parallel algorithm.

Acknowledgement

This work is partially supported by JSPS Grant-in-Aid for Scientific Research (A) Japan, No.20246027.

Reference

1) Demmel, J.: Applied Numerical Linear Algebra, SIAM, Philadelphia (1997).
2) Golub, G. and Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix, SIAM J. Numeri. Anal., Vol. 2, pp.205-224(1965).
3) Demmel, J. and Kahan, W.: Accurate singular values of bidiagonal matrices, SIAM J. Sci. Sta. Comput., Vol.67, pp.191-229 (1994).
4) Francis, J.G.F.: The QR transformation a unitary analogue to the LR transformation--part 1, Computer J., Vol.4, pp.265-271 (1961).
5) Golub, G. and Reinsch, C.: Singular value decomposition and least squares solutions, Numer. Math., Vol.14, pp.403-420 (1970).
6) Iwasaki, M., and Nakamura, Y.: On the convergence of a solution of the discrete Lotka-Volterra system, Inverse Problems Vol.18, pp.1569-1578 (2002).
7) Iwasaki, M., and Nakamura, Y.: Accurate computation of singular values in terms of the shifted integrable algorithm, Japan J. Indust. Appl. Math. Vol.23, pp.239-259 (2006).
8) Takata, M., Kimura, K., Iwasaki, M. and Nakamura, Y.: An Evaluation of Singular Value Computation by the Discrete Lotka-Volterra System, In Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, Vol.II, pp.410-416 (2005).
9) Takata, M., Kimura, K., Iwasaki, M. and Nakamura, Y.: Performance of a New Singular Value Decomposition Scheme for Large Scale matrices, In Proceedings of The IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN2006), pp.304-309 (2006).
10) Jiang, F., Kannon, R., Littman, M.L., and Vemphala, S.: Efficient Singular Value Decomposition via Improved Document Sampling. Department of Computer Science, Duke University: Technical Report CS-99-5 (1999).
11) Parlett, B. N., and Marques, O. A.: An Implementation of the dqds Algorithm (Positive Case), Proc. of the International Workshop on Accurate Solution of Eigenvalue Problems (University Park, PA, 1998), Lin. Alg. Appl. 309, No.1-3, pp.217-259 (2000).
12) Johnson, C. R.: A Gersgorin--type lower bound for the smallest singular value, Lin. Alg. Appl., Vol.112, pp.1-7 (1989).
13) Zima, H., and Chapman, B.: Supercompilers for Parallel and Vector Computers, Addison-Wesley Publishing (1991).
14) Takata, M., Kimura, K., Iwasaki, M., and Nakamura, Y.: Algorithms for Generating Bidiagonal Test Matrices, In Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'07), Vol.II, pp.732-738 (2007).