Proceedings of ALGORITMY 2002
Conference on Scientific Computing, pp. 113–120

PARALLEL SVD COMPUTATION IN UPDATING PROBLEMS OF LATENT SEMANTIC INDEXING *

GABRIEL OKŠA, MARTIN BEČKA AND MARIÁN VAJTERŠIC †

Abstract. In latent semantic indexing, the addition of documents (or the addition of terms) to some already processed text collection leads to the updating of the best rank-k approximation of the term-document matrix. The computationally most intensive task in this updating is the computation of the singular value decomposition (SVD) of certain square matrix, which is upper or lower triangular, and contains a diagonal block in its upper left corner. For the solution of this task, the new dynamic ordering of subproblems is compared with the up-to-now preferred static cyclic one in the parallel two-sided block-Jacobi SVD algorithm. The results of numerical experiments show that, for a given accuracy, the dynamic ordering is much more efficient that the static cyclic one with respect to the number of parallel iteration steps needed for the convergence of the two-sided block-Jacobi SVD algorithm.

Key words. latent semantic indexing, updating documents, updating terms, singular value decomposition, parallel two-sided block-Jacobi algorithm, static ordering, dynamic ordering

AMS subject classifications. 15-01, 15A03, 15A18, 65F50, 68P20, 68W10

1. Introduction. Latent semantic indexing (LSI) is a concept-based automatic indexing method for overcoming the two fundamental problems which exist in the traditional lexical-matching retrieval schemes: synonymy and polysemy [5]. With respect to the synonymy, several different words can be used to express a concept and the keywords in a user's query may not match those in the relevant documents. On the other hand, the polysemy means that the words can have multiple meanings and the user's words may match those in the irrelevant documents. LSI is an extension of the vector space model for information retrieval [6, 5]. In the vector space model, the collection of text documents is represented by a *term-document* matrix $A = (a_{ij}) \in \mathbb{R}^{m\times n}$, where a_{ij} is based on the number of times the term i appears in the document j, m is the number of terms, and n is the number of documents in the collection. Hence, a document becomes a column vector, and a user's query can also be represented as a vector of the same dimension. The similarity between a query vector and a document vector is usually measured by the cosine of the angle between them, and for each query a list of documents ranked in a decreasing order of similarity is returned to the user.

LSI modifies this vector space model by modeling the term-document relationship using a *reduced-dimension representation* (RDR) of term-document matrix A computed by its singular value decomposition (SVD). Let

$$A = P\Sigma Q^T, \quad \Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{\min\{m,n\}}), \quad \sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_{\min\{m,n\}},$$

be the SVD of A. Then the RDR is given by the best rank-k approximations $A_k = P_k\Sigma_k Q_k^T$, $k < \min\{m,n\}$, where P_k and Q_k consist of the first k columns of P and

*This work was supported by VEGA Grant no. 2/1097/21 from the Scientific Grant Agency of Slovak Republic and Slovak Academy of Sciences, and by the DAAD (German Academic Exchange Service) during the stay of the third co-author at the Department of Informatics, Technical University of Munich, Munich, Germany.

†Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, P. O. Box 56, 840 00 Bratislava, Slovak Republic ((gabo, mato, marian)@ifi.savba.sk)

Q, respectively, and Σ_k is the kth leading principal sub-matrix of Σ. Each of the k reduced dimensions represents a so-called *pseudo-concept* [6], which may not have any explicit semantic content but helps to discriminate documents [6, 7].

In rapidly changing environments such as the World Wide Web, the document collection is frequently updated with new documents and terms constantly being added. Hence, the task arises to efficiently update the old LSI-generated RDR after an addition of new documents and terms. In Section 2, the mathematical model of updating is briefly presented, which is based on algorithms derived in [8]. It turns out that the computationally most intensive task in the correct updating is the SVD computation of some upper or lower triangular matrix. In Section 3 we present the parallel two-sided block-Jacobi SVD algorithm for solving this problem. In the numerical experiments, the dynamic ordering of subproblems is compared with the cyclic one, and the efficiency of the former is documented and discussed. Section 4 concludes the paper.

2. Two updating problems in LSI.

2.1. Updating documents. Let us suppose that the RDR of order k was already computed and stored for some term-document matrix A, and the original matrix was discarded (e.g. for the memory reasons), so that only $A_k = P_k \Sigma_k Q_k^T$ is available in the factored form. Let $D \in \mathbb{R}^{m\times p}$ be p new documents. The task is to compute the best rank-k approximation of the column partitioned matrix

$$B \equiv (A_k,\, D)\ .$$

Using the factorization of A_k, the matrix B can be written as

$$\begin{aligned} B &= \left(P_k \Sigma_k Q_k^T,\, D\right) \\ &= \left(P_k,\, (I_m - P_k P_k^T)\, D\right) \cdot \begin{pmatrix} \Sigma_k & P_k^T D \\ 0 & I_p \end{pmatrix} \cdot \begin{pmatrix} Q_k^T & 0 \\ 0 & I_p \end{pmatrix}\ . \end{aligned}$$

Note that $I_m - P_k P_k^T$ is the matrix representation of the orthogonal projection, which maps the columns of matrix D into the subspace $\mathcal{P}_k^{\perp}$ that is orthogonal to the column range of matrix P_k. Let $(I_m - P_k P_k^T)D = \hat{P}_p\, R$ be the QR decomposition of the matrix $(I_m - P_k P_k^T)D$. Then

$$B = (P_k,\, \hat{P}_p) \cdot \begin{pmatrix} \Sigma_k & P_k^T D \\ 0 & R \end{pmatrix} \cdot \begin{pmatrix} Q_k^T & 0 \\ 0 & I_p \end{pmatrix}\ . \tag{2.1}$$

The crucial point in the above derivation is the observation that the p orthonormal columns of matrix $\hat{P}_p$ are mutually orthogonal to the k orthonormal columns of matrix P_k because the columns of $\hat{P}_p$ constitute the orthonormal basis of the subspace $\mathcal{P}_k^{\perp}$. Note that two exterior matrices on the right hand side of Eq. (2.1) are orthogonal, but the inner matrix is not diagonal. Hence, from the computational point of view, the updating problem is reduced to the SVD of the inner matrix in Eq. (2.1).

Based on these facts, Zha and Simon [8] have derived a method for solving the problem of updating documents. Their approach is summarized in Algorithm 1.

Notice that Step 4 in Algorithm 1 requires the SVD of structured matrix $\hat{B}$, which is upper triangular with the diagonal left upper block of order $k \times k$. Simultaneously, this step represents the most intensive computation in Algorithm 1.

Algorithm 1 Algorithm for updating documents

1: *Input:* k, $P_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, $Q_k \in \mathbb{R}^{n\times k}$, $D \in \mathbb{R}^{m\times p}$.
2: Compute the projection: $\hat{D} = (I_m - P_k\, P_k^T)\, D$.
3: Compute the QR decomposition: $\hat{D} = \hat{P}_p\, R$, where $\hat{P}_p \in \mathbb{R}^{m\times p}$, $R \in \mathbb{R}^{p\times p}$.
4: Compute the SVD of matrix

$$\hat{B} \equiv \begin{pmatrix} \Sigma_k & P_k^T D \\ 0 & R \end{pmatrix} \in \mathbb{R}^{(k+p)\times(k+p)}$$

in the form:

$$\hat{B} = (U_k,\, U_k^\perp)\cdot \mathrm{diag}(\hat{\Sigma}_k,\, \hat{\Sigma}_p)\cdot (V_k,\, V_k^\perp)^T\,,$$

where U_k, $V_k \in \mathbb{R}^{(k+p)\times k}$ and $\hat{\Sigma}_k \in \mathbb{R}^{k\times k}$.
5: *Output:* The best rank-k approximation of $B = (A_k,\, D)$ is given by:

$$B_k \equiv \Big[(P_k,\, \hat{P}_p)\, U_k\Big]\cdot \hat{\Sigma}_k \cdot \left[\begin{pmatrix} Q_k & 0 \\ 0 & I_p \end{pmatrix} V_k\right]^T .$$

2.2. Updating terms. In this case, let $T \in \mathbb{R}^{q\times n}$ be the q new term vectors that should be added to the existing terms at the bottom of the old term-document matrix. The task is to compute the best rank-k approximation of the row partitioned matrix

$$C \equiv \begin{pmatrix} A_k \\ T \end{pmatrix} .$$

Using steps similar to those in the previous paragraph (see [8]), one gets the Algorithm 2 for the correct updating of terms.

Similarly to the problem of updating documents, the computationally most intensive step is the SVD of the lower triangular matrix $\hat{C}$ with the upper left diagonal block. Since the upper and lower triangular matrices are related by the matrix transposition that affects the SVD only by interchanging the left and right singular vectors, in the following we focus on the upper triangular matrix $\hat{B}$ in Algorithm 1. The conclusions with respect to the efficiency of the SVD computation will be valid for both updating problems.

3. Parallel SVD computation. In this section, the emphasis is on the parallel computation of the SVD of matrix $\hat{B}$ in Step 4 of Algorithm 1. We briefly introduce the parallel two-sided block-Jacobi SVD algorithm with the dynamic ordering of individual subproblems, which is described in more detail in [4].

Recall that $\hat{B}$ is the upper triangular, square matrix of order $v = k + p$ with the diagonal upper left block of order k. In practice, the parameter k depends on the text collection and covers the range from 100 to 300 (cf. [6, 5, 7, 8]). It is clear from the output of Algorithm 1 that only k largest singular values and their left and right singular vectors are needed for the construction of B_k so that some iterative method for the *partial* SVD can be considered in this case. On the other hand, the Jacobi SVD algorithm computes the *complete* SVD. However, when $k \gg p$ (or $k \gg q$)—i.e., the number of added documents (or added terms) is small as compared to k—and

Algorithm 2 Algorithm for updating terms

1: *Input:* k, $P_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, $Q_k \in \mathbb{R}^{n\times k}$, $T \in \mathbb{R}^{q\times n}$.
2: Compute the projection: $\hat{T} = (I_n - Q_k\, Q_k^T)\, T^T \in \mathbb{R}^{n\times q}$.
3: Compute the QR decomposition: $\hat{T} = \hat{Q}_q\, L^T$, where $\hat{Q}_q \in \mathbb{R}^{n\times q}$, $L \in \mathbb{R}^{q\times q}$.
4: Compute the SVD of matrix

$$\hat{C} \equiv \begin{pmatrix} \Sigma_k & 0 \\ TQ_k & L \end{pmatrix} \in \mathbb{R}^{(k+q)\times(k+q)}$$

in the form:

$$\hat{C} = (U_k,\, U_k^{\perp}) \cdot \mathrm{diag}(\hat{\Sigma}_k,\, \hat{\Sigma}_q) \cdot (V_k,\, V_k^{\perp})^T\,,$$

where U_k, $V_k \in \mathbb{R}^{(k+q)\times k}$ and $\hat{\Sigma}_k \in \mathbb{R}^{k\times k}$.

5: *Output:* The best rank-k approximation of $C = \begin{pmatrix} A_k \\ T \end{pmatrix}$ is given by:

$$C_k \equiv \left[\begin{pmatrix} P_k & 0 \\ 0 & I_q \end{pmatrix} U_k\right] \cdot \hat{\Sigma}_k \cdot \left[(Q_k,\, \hat{Q}_q)\, V_k\right]^T\,.$$

when the parallel computation is performed with a sufficient speedup, the usage of the parallel two-sided block-Jacobi SVD algorithm can be justified in the updating problems of LSI.

3.1. Two-sided block-Jacobi method with dynamic ordering. The two-sided block-Jacobi method for computing the SVD operates on an $l \times l$ block partition of $\hat{B}$, where l is a chosen blocking factor. The kernel operation of the method is the SVD of 2×2 block subproblems

$$S_{ij} = \begin{pmatrix} \hat{B}_{ii} & \hat{B}_{ij} \\ \hat{B}_{ji} & \hat{B}_{jj} \end{pmatrix}\,, \tag{3.1}$$

where, for a given pair (i,j), $i,j = 0,\, 1, \ldots,\, l-1,\ i \neq j$, the orthogonal matrices X_{ij} and Y_{ij} are generated such that the product

$$X_{ij}^T S_{ij} Y_{ij} = D_{ij}$$

is a block diagonal matrix of the form

$$D_{ij} = \begin{pmatrix} J_{ii} & 0 \\ 0 & J_{jj} \end{pmatrix}\,.$$

In the *cyclic* method, the ordering of pairs (i,j) is prescribed according to some static list. The solution of (3.1), for each pair (i,j), $i,j = 0, 1, \ldots, l-1$, $i \neq j$, constitutes one *sweep* of the method. Hence, the number of the subproblems to be solved in one sweep is $L = l(l-1)/2$.

The termination criterion of the whole process is

$$F(\hat{B},\, l) = \sqrt{\sum_{i,j=0,\, i\neq j}^{l-1} \|\hat{B}_{ij}\|_F^2} < \epsilon\,, \tag{3.2}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, $\epsilon = prec \cdot \|\hat{B}\|_F$ is the required accuracy (measured relatively to the Frobenius norm of the original matrix $\hat{B}$), and *prec* is a suitably chosen small constant.

A subproblem (3.1) is solved only if

$$F(S_{ij}, l) = \sqrt{\|\hat{B}_{ij}\|_F^2 + \|\hat{B}_{ji}\|_F^2} \geq \delta\,, \tag{3.3}$$

where $\delta = \epsilon/L$ is a given subproblem accuracy.

The two-sided block-Jacobi SVD algorithm with some static cyclic ordering can be efficiently parallelized as shown in [2, 3].

In [4], a new variant of the parallel two-sided block-Jacobi SVD algorithm was designed, implemented and tested. Let the SVD be computed on w processors, and let the blocking factor be $l = 2w$. In one parallel iteration step that represents w serial iteration steps, one would like to decrease the norm of the off-diagonal matrix blocks as much as possible. This task can be formulated in terms of graph theory as the *maximum-weight perfect matching problem.*

Consider a weighted complete graph $G = (\mathcal{V}, \mathcal{E}) = K_l$, where the nodes are numbered from 0 to $l-1$, $\mathcal{E} = \{(i,j) \mid i < j\}$ and the edge (i,j) has the weight $z_{ij} = \|\hat{B}_{ij}\|_F + \|\hat{B}_{ji}\|_F$. We can achieve the maximum-weight perfect matching of this graph in time $O(w^3)$. When an approximate solution is good enough, a simple greedy approach can be used. Let us sort the edges with respect to their weights in nonincreasing order. Then let us scan this ordered sequence from left to right and add an edge to the matching only if neither of its endpoints is an endpoint of any previously selected edge. The complexity of this greedy algorithm is $O(w^2 \log w)$ due to the complexity of sorting. We refer to our paper [4] for the detailed discussion regarding the complexity of the whole parallel two-sided block-Jacobi SVD algorithm.

The parallel algorithm for the processor me, $me = 0, 1, \ldots, w-1$, can be written in the form of Algorithm 3, where $U_{:i}$ denotes the ith $v \times (v/2w)$ block column of matrix U (similarly for matrices $\hat{B}$ and V).

The procedure `SVD`(S_{ij}) in the inner loop of Algorithm 3 computes the SVD of subproblem (3.1). In the procedure `AllGather`, each processor sends its matrix X_{ij} to all other processors, so that each processor maintains an array (denoted by XX) of w matrices. The procedure `ReOrderingComp` computes the optimal reordering destinations of all block columns residing in a given processor (*dest*1 and *dest*2) and their locations at new position (*tag*1 and *tag*2), based on the updated weight matrix Z. The argument *tag* provides the matching between the corresponding `send` and `receive` calls.

3.2. Numerical experiments. In this section, the numerical experiments on a parallel supercomputer with a variable number of processors are described and their results are discussed. We compare the performance of Jacobi algorithm using the dynamic ordering with that using the up-to-now preferred cyclic ordering of subproblems.

The above parallel algorithm was implemented in Fortran on an SGI—Cray Origin 2000 parallel computer using the Message Passing Interface (MPI) library. The number of used processors covered the range $w = 2, 5, 10$ and 15. For the matrix multiplications, the BLAS procedure `zgemm` was applied, and the LAPACK procedure `zgesvd` [1] was used for the computation of the embedded SVD in the inner loop of Algorithm 3. The constant $prec = 10^{-10}$ was chosen for the computation of ϵ and δ (see Eqns. (3.2) and (3.3)). All computations were made using the IEEE standard double precision floating point arithmetic with the machine precision $\epsilon_M \approx 1.11 \times 10^{-16}$.

Algorithm 3 Parallel block-Jacobi SVD algorithm with dynamic ordering

$U = I_v$, $V = I_v$
$(i, j) = (2 \cdot me,\ 2 \cdot me + 1)$
while $F(\hat{B}, l) \geq \epsilon$ **do**
  if $F(S_{ij}, l) \geq \delta$ **then**
    % Computation of X_{ij} and Y_{ij} by SVD of S_{ij}
    SVD(S_{ij}) $\rightarrow$ X_{ij}, Y_{ij}
    % Update of block columns
    $(\hat{B}_{:i}, \hat{B}_{:j}) = (\hat{B}_{:i}, \hat{B}_{:j}) \cdot Y_{ij}$
    $(U_{:i}, U_{:j}) = (U_{:i}, U_{:j}) \cdot X_{ij}$
    $(V_{:i}, V_{:j}) = (V_{:i}, V_{:j}) \cdot Y_{ij}$
  else
    $X_{ij} = I_{v/w}$
  end if
  AllGather(X_{ij}, i, j) $\rightarrow$ $XX(t) = (X_{rs}, r, s)$, $t = 0, 1, \ldots, w-1$
  for $t = 0$ to $w - 1$ **do**
    % Update of block rows
    $\begin{pmatrix} \hat{B}_{ri} & \hat{B}_{rj} \\ \hat{B}_{si} & \hat{B}_{sj} \end{pmatrix} = X^T_{rs,t} \cdot \begin{pmatrix} \hat{B}_{ri} & \hat{B}_{rj} \\ \hat{B}_{si} & \hat{B}_{sj} \end{pmatrix}$
  end for
  update(Z)
  ReOrderingComp(i, j, Z, me) $\rightarrow$ $dest1$, $dest2$, $tag1$, $tag2$
  copy($\hat{B}_{:i}$, $U_{:i}$, $V_{:i}$, i) $\rightarrow$ $\hat{B}_{:r}$, $U_{:r}$, $V_{:r}$, r
  copy($\hat{B}_{:j}$, $U_{:j}$, $V_{:j}$, j) $\rightarrow$ $\hat{B}_{:s}$, $U_{:s}$, $V_{:s}$, s
  send($\hat{B}_{:r}$, $U_{:r}$, $V_{:r}$, r, $dest1$, $tag1$)
  send($\hat{B}_{:s}$, $U_{:s}$, $V_{:s}$, s, $dest2$, $tag2$)
  receive($\hat{B}_{:i}$, $U_{:i}$, $V_{:i}$, i, 1)
  receive($\hat{B}_{:j}$, $U_{:j}$, $V_{:j}$, j, 2)
end while

In the experiments, the order of the square upper triangular matrix $\hat{B}$ together with the size of its upper left diagonal block Σ_k were fixed: $v = k + p = 500$ and $k = 150$. The elements of $\hat{B}$ were generated randomly in two steps using two positive constants α and β. First, k values uniformly distributed in the interval [0, 1] were obtained and multiplied by α; they constituted the diagonal of Σ_k. Next, the remaining elements of matrix blocks $P_k^T D$ and R uniformly distributed in the interval $[-1, 1]$ were generated and multiplied by β. The adopted approach enabled us to modify the ratio between the Frobenius norm of Σ_k and that of the rest of matrix $\hat{B}$. This is equivalent to the modeling of the relative weight that the new documents brought to the document collection. In our experiments, the value of $\alpha = 100$ was fixed and $\beta = 1$, 2, 5, 10, 20 and 50.

The experimental results are presented in the following tables. For a given number of processors w, the parallel computational time in seconds (first column) and the corresponding number of parallel iteration steps (second column) are shown for all values of parameter β mentioned above. In Table 3.1, the performance of the parallel algorithm is documented using the sweep technique with the static cyclic odd-even ordering ($CO(0)$, see [2]). Table 3.2 contains the results of the new dynamic ordering method. Let us discuss these results in more detail.

TABLE 3.1
Results for the static cyclic ordering $CO(0)$

β / w	2		5		10		15	
1	43.5	21	32.5	90	22.7	228	17.1	348
2	43.9	21	29.1	99	23.4	247	18.2	348
5	64.5	27	42.5	117	35.4	266	29.8	406
10	71.3	30	62.7	153	53.0	285	46.2	425
20	81.8	36	65.0	153	55.3	342	49.7	522
50	72.3	30	52.4	153	48.4	342	45.4	580

TABLE 3.2
Results for the greedy dynamic ordering

β / w	2		5		10		15	
1	22.4	10	17.1	47	13.5	107	10.0	173
2	24.2	10	17.4	48	14.7	111	11.4	183
5	27.2	11	21.2	49	15.2	112	12.1	187
10	29.7	11	20.8	51	17.5	115	14.9	195
20	29.7	12	19.9	53	18.1	121	15.0	198
50	29.4	12	22.1	55	20.3	126	17.7	211

For both orderings, the number of parallel iteration steps increases with an increase of the blocking factor $l = 2w$, i.e., with an increase of the number of processors w. This is in accordance with the statistical analysis given in [4].

The number of parallel iterations steps needed for the convergence is an objective, machine independent measure of the algorithm's performance. (Note that the computational time depends very much on the organizational details of computations adopted in a parallel computer, and the user has usually no direct access to influence these rules.) Comparing Table 3.1 and 3.2, the greedy dynamic ordering clearly outperforms the cyclic one for all combinations of β and w. The ratio of the number of parallel iteration steps between the old method and the new one lies in the range 1.9–3.0 (average is 2.4). For a given number of processors w, this ratio has a tendency to grow with an increase of β, i.e., the greedy dynamic ordering is more efficient in reducing the relatively larger off-diagonal norms of matrix blocks than the cyclic ordering. This observation can be explained by the inherent property of the greedy dynamic ordering to pair the matrix blocks with the maximal sum of Frobenius norms.

Figure 3.1 depicts (in the logarithmic scale) the decrease of Frobenius norm of the off-diagonal blocks for $\beta = 50$ and $w = 10$. Similar behavior can be observed also for other combinations of β and w. The different final norms for the dynamic and static cyclic ordering result from the fact that, in the case of cyclic ordering, the whole sweep must end before the convergence criterion is checked. Notice that for the static cyclic ordering there are many "empty" parallel iteration steps that do not change the Frobenius norm of the off-diagonal blocks at all due to the prescribed combinations of nondiagonal blocks that do not fulfil the criterion given by Eq. (3.3). These steps correspond to the horizontal segments on the curve for the cyclic ordering in Fig. 3.1. In other words, the static cyclic ordering of subproblems is, so to say, "blind", because it does not take into account the actual status of the matrix, i.e., how the overall Frobenius norm is spread over the individual off-diagonal matrix blocks. Since the

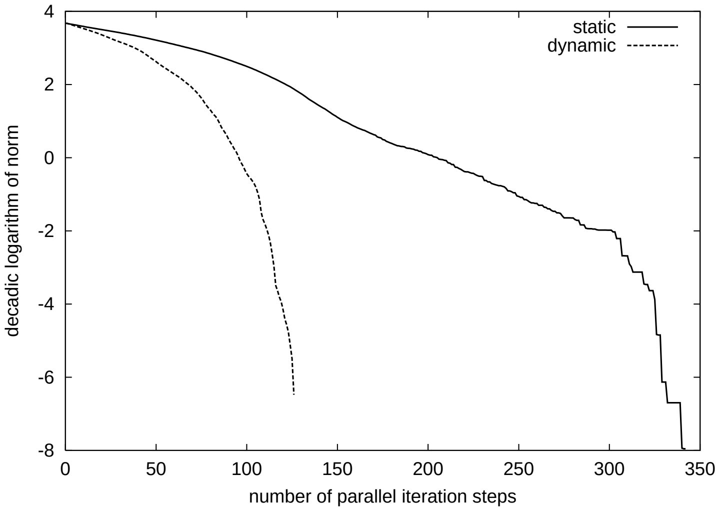

FIG. 3.1. *Decrease of Frobenius norm of the off-diagonal blocks for* $\beta = 50$ *and* $w = 10$

dynamic ordering combines the nondiagonal blocks with maximal Frobenius norms, no such effect is observed in this case, and the Frobenius norm of the off-diagonal blocks decreases strictly monotonically.

4. Conclusion. We have applied the new approach to the SVD computation based on the parallel two-sided block-Jacobi algorithm with the greedy dynamic ordering to the updating problems in the LSI. First experimental results show that, for a given accuracy, the greedy dynamic ordering is typically 2–3 times more efficient with respect to the number of parallel iteration steps needed for the convergence than the static cyclic ordering. More experimental and theoretical work needs to be done to answer the question, if it is possible to explore fully the special, upper (lower) triangular structure of certain matrices arising in Algorithms 1 and 2 using parallel Jacobi-like algorithms.

REFERENCES

[1] A. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, S. OSTROUCHOV AND D. SORENSEN, *LAPACK Users' Guide*, Second ed., SIAM, Philadelphia, 1999.

[2] M. BEČKA AND M. VAJTERŠIC, *Block-Jacobi SVD algorithms for distributed memory systems: I. Hypercubes and rings*, Parallel Algorithms Appl., 13 (1999), pp. 265–287.

[3] M. BEČKA AND M. VAJTERŠIC, *Block-Jacobi SVD algorithms for distributed memory systems: II. Meshes*, Parallel Algorithms Appl., 14 (1999), pp. 37–56.

[4] M. BEČKA, G. OKŠA AND M. VAJTERŠIC, *Dynamic ordering for a parallel block-Jacobi SVD algorithm*, Parallel Computing, 28 (2002), pp. 243–262.

[5] M. W. BERRY AND M. BROWNE, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, First ed., SIAM, Philadelphia, PA, 1999.

[6] M. W. BERRY, Z. DRMAČ AND E. R. JESSUP, *Matrices, vector spaces, and information retrieval*, SIAM Review, 41 (1999), pp. 335–362.

[7] H. ZHA, *A subspace-based model for information retrieval with applications in latent semantic indexing*, in Proceedings of Irregular '98, Lecture Notes in Computer Science 1457, Springer Verlag, New York, NY, 1998, pp. 29–42.

[8] H. ZHA AND H. D. SIMON, *On updating problems in latent semantic indexing*, SIAM J. Sci. Comput., 21 (1999), pp. 782–791.