UNIVERSITÄT SALZBURG

Parallel Methods in Updating/Downdating Problems of the Latent Semantic Indexing

Gabriel Okša[a] Marián Vajteršic

[a]Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic

Technical Report 2006-03 June 2006

Department of Computer Sciences

Jakob-Haringer-Straße 2
5020 Salzburg
Austria
www.cosy.sbg.ac.at

Technical Report Series

Parallel Methods in Updating/Downdating Problems of the Latent Semantic Indexing

Gabriel Okša* and Marián Vajteršic†

Abstract. *In the case of large databases, which are encoded on some sort of a parallel computer (e.g., a supercomputer or a cluster of personal computers), it is frequently needed to update or downdate either documents or terms. This task can be done in parallel based on the theory of the Singular Value Decomposition (SVD) of a Term-Document Matrix (TDM). However, the TDM is usually not explicitly stored and only its truncated SVD in the form of a chosen set of left and right singular vectors and corresponding singular values is at our disposal. Moreover, in the case of huge databases, these components of the truncated SVD may be themselves distributed over a set of processors rather than placed on one processor. For such a distributed system, we will analyze the data structure and computation flow in an updating/downdating problem. It turns out that the most important feature is a special, near-to-triangular form of the TDM, which has to be processed. Therefore, the first step of our method consists of transforming the TDM to triangular form. For triangular matrices, we design a parallel algorithm based on the parallel block Kogbetliantz method under the Message Passing paradigm of communication between processors.*

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

*Mathematical Institute, Department of Informatics, Slovak Academy of Sciences, Bratislava, Slovak Republic, email: Gabriel.Oksa@savba.sk.

†Department of Computer Sciences, University of Salzburg, Salzburg, Austria, email: marian@cosy.sbg.ac.at.

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

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

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 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, 15].

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 [16]. 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 4 we design the parallel SVD algorithm for solving this problem that is based on the Kogbetliantz method. Section 5 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

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

Note that $I_m - P_kP_k^T$ 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 $(I_m - P_kP_k^T)D = \hat{P}_p\, R$ be the QR decomposition of the matrix $(I_m - P_kP_k^T)D$. Then

$$B = (P_k,\, \hat{P}_p) \cdot \begin{pmatrix} \Sigma_k & P_k^TD \\ 0 & R \end{pmatrix} \cdot \begin{pmatrix} Q_k^T & 0 \\ 0 & I_p \end{pmatrix}. \tag{1}$$

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. (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. (1).

Based on these facts, Zha and Simon [16] have derived a method for solving the problem of updating documents. Their approach is summarized in Algorithm 2.1. Notice that Step 4 in

---

2.1 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:

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

Algorithm 2.1 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 2.1.

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 [16]), one gets the Algorithm 2.2 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,

2.2 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:

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

in the following we focus on the upper triangular matrix $\hat{B}$ in Algorithm 2.1. The conclusions with respect to the efficiency of the SVD computation will be valid for both updating problems.

3 Two downdating problems in LSI

In downdating problems, there exists the k-dimensional approximation of the original term-document matrix A in the form $A_k = P_k\Sigma_k Q_k^T$. As above we assume that only factors P_k, Σ_k and Q_k are available. In contrast with updating problems, our task is now either to delete p documents, i.e. the matrix D of order $m \times p$ from the representation $A_k = [D,\, \tilde{A}_k]$ (downdating the documents), or to delete t terms, i.e. the matrix T of order $t \times n$ from the representation $A_k = \begin{pmatrix} T \\ \hat{A}_k \end{pmatrix}$. Our new database is represented by matrix $\tilde{A}_k$ or $\hat{A}_k$, and we must end with the k-dimensional SVD representation of a reduced matrix in either case. Next we describe effective serial algorithms published in [14], which solve both downdating problems. Since both algorithms are very similar, we will describe in detail only the algorithm for downdating the documents and comment on differences by downdating of terms.

3.1 Downdating documents

Let us start with the representation $A_k = [D,\, \tilde{A}_k] = P_k\Sigma_k Q_k^T$. Let the matrix $I_n^{1:p}$ denote the first p rows of the identity matrix of order n (we assume $n > p$, which is natural—not all n documents are deleted from a database). Let us define the matrix $\hat{Q}$ as an $n\times(k+p)$ orthogonal matrix of the form

$$\hat{Q} = (Q_k,\, S),$$

where S contains p orthogonal columns of length n, which are orthogonal also to the columns of Q_k — i.e., $Q_k^T S = 0$ (to be sure that such S exists, we assume that $k + p \leq n$). Then one can check by direct computation that the following decomposition is valid:

$$\begin{pmatrix} I_p & 0 \\ 0 & P_k^T \end{pmatrix} \cdot \begin{pmatrix} I_n^{1:p} \\ A_k \end{pmatrix} \cdot \hat{Q} = \begin{pmatrix} Q_k^{1:p} & S^{1:p} \\ \Sigma_k & 0 \end{pmatrix} \equiv W.$$

Thus the right-hand side matrix above, referred to as W, is composed of the first p rows of Q_k followed by the first p rows of S.

The key step now is to reduce W^T by orthogonal transformations into a special form, which will contain the identity I_p as the left upper diagonal block. Since Σ_k is diagonal, we can write

$$W^T = \begin{pmatrix} (Q_k^{1:p})^T & \Sigma_k \\ (S^{1:p})^T & 0 \end{pmatrix}.$$

Notice the special structure of W^T. First p columns are dense (in fact, these are the first p orthogonal rows of the orthogonal matrix $\hat{Q}$), but next k columns are very sparse, because Σ_k is diagonal and the bottom diagonal block is zero. It is this special structure which allows to use left and right Givens rotations in a so-called non-zero chasing scheme [7, pp.145-149] to obtain:

$$G_l W^T G_r = G_l \begin{pmatrix} Q_k^{1:p} & S^{1:p} \\ \Sigma_k & 0 \end{pmatrix}^T G_r = \begin{pmatrix} I_p & 0 \\ Y & \tilde{B} \end{pmatrix}^T,$$

where $\tilde{B}$ is the lower triangular matrix. Here G_l and G_r are orthogonal matrices of order $(k+p)$ constructed as products of individual Givens rotations. It can be shown that G_r^T does not act on first p rows of the composed matrix. Therefore

$$G_r^T \begin{pmatrix} I_p & 0 \\ 0 & P_k^T \end{pmatrix} = \begin{pmatrix} I_p & 0 \\ 0 & \bar{P}_k^T \end{pmatrix}.$$

However, G_l^T reduces exactly p first columns of $\hat{Q}$ to $(I_p, 0)^T$, so that

$$\hat{Q} G_l^T = \begin{pmatrix} I_p & 0 \\ 0 & \bar{Q}_k \end{pmatrix}$$

(since the columns of $\hat{Q}$ are orthogonal and remain so also after the orthogonal transformation, the block 12 must be zero). Then it follows that

$$\begin{pmatrix} I_p & 0 \\ 0 & \bar{P}_k^T \end{pmatrix} \cdot \begin{pmatrix} I_p & 0 \\ D & \tilde{A}_k \end{pmatrix} \cdot \begin{pmatrix} I_p & 0 \\ 0 & \bar{Q}_k \end{pmatrix} = \begin{pmatrix} I_p & 0 \\ Y & \tilde{B} \end{pmatrix},$$

and the second row yields the downdated $\tilde{A}_k$ given by

$$\bar{P}_k^T \tilde{A}_k \bar{Q}_k = \tilde{B},$$

where $\tilde{B}$ is the lower triangular, banded matrix of order k. If the full SVD of $\tilde{B}$ is

$$\tilde{B} = P_B \, \Sigma_B \, Q_B^T,$$

then the SVD of $\tilde{A}_k$ is given by

$$\tilde{A}_K = (\bar{P}_k \, P_B) \cdot \Sigma_B \cdot (\bar{Q}_k \, Q_B)^T \equiv \tilde{P}_k \, \tilde{\Sigma}_k \, \tilde{Q}_k^T$$

3.1 Algorithm for downdating 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: Complete Q_k into the orthonormal matrix $\hat{Q} = (Q_k,\, S)$ of order $n \times (k+p)$ by taking p random vectors and orthogonalizing them by the modified Gramm-Schmidt process.

3: Form the matrix $W = \begin{pmatrix} Q_k^{1:p} & S^{1:p} \\ \Sigma_k & 0 \end{pmatrix}$ and find orthogonal matrices G_l and G_r so that

$$G_l W^T G_r = G_l \begin{pmatrix} Q_k^{1:p} & S^{1:p} \\ \Sigma_k & 0 \end{pmatrix}^T G_r = \begin{pmatrix} I_p & 0 \\ Y & \tilde{B} \end{pmatrix}^T,$$

where $\tilde{B}$ is the lower triangular matrix of order k.

4: Compute $\bar{P}_k$ and $\bar{Q}_k$ by:

$$G_r^T \begin{pmatrix} I_p & 0 \\ 0 & P_k^T \end{pmatrix} = \begin{pmatrix} I_p & 0 \\ 0 & \bar{P}_k^T \end{pmatrix}, \quad \hat{Q} G_l^T = \begin{pmatrix} I_p & 0 \\ 0 & \bar{Q}_k \end{pmatrix}.$$

5: Compute the SVD of $\tilde{B}$, $\tilde{B} = P_B\, \Sigma_B\, Q_B^T$. All matrices are square of order k.

6: *Output:* The best rank-k approximation of $\tilde{A}_k$ is given by $\tilde{P}_k = \bar{P}_k\, P_B$, $\tilde{\Sigma}_k = \Sigma_B$ and $\tilde{Q}_k^T = (\bar{Q}_k\, Q_B)^T$.

with $\tilde{P}_k = \bar{P}_k\, P_B$, $\tilde{\Sigma}_k = \Sigma_B$ and $\tilde{Q}_k^T = (\bar{Q}_k\, Q_B)^T$.

All steps required for deleting a block of documents are summarized in the following Algorithm 3.1. The most computationally demanding task in Algorithm 3.1 is the SVD of the lower triangular matrix $\tilde{B}$ in step 5.

3.2 Downdating terms

This case is indeed very similar to the above one for deleting documents. Let $A_k = \begin{pmatrix} T \\ \tilde{A}_k \end{pmatrix} = P_k \Sigma_k Q_k^T$ be our original database of order $m \times n$, from which q terms should be removed. These terms are placed on the top and are present in all documents, so that their influence is defined by the matrix T of order $q \times n$. Notice that the matrix A_k^T has the structure identical to the case of deleting documents. Hence, to delete a block of terms, one can work with the representation of A_k^T and use the algorithm from previous subsection.

However, similar steps as in the previous subsection, applied directly to the SVD factors of A_k, lead to Algorithm 3.2. Again, the most computationally expensive task is the SVD of the upper triangular matrix $\tilde{B}$.

When comparing together Algorithms 2.1, 2.2, 3.1 and 3.2, the computational pattern is similar—each task requires the SVD of a lower or upper triangular matrix, which can have some interesting additional structure. Next part of this report describes the parallel Kogbetliantz variant of the Jacobi method for doing this.

3.2 Algorithm for downdating 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: Complete P_k into the orthonormal matrix $\hat{P} = (P_k,\, Z)$ of order $m \times (k+q)$ by taking q random vectors and orthogonalizing them by the modified Gramm-Schmidt process.

3: Form the matrix $H = \begin{pmatrix} P_{1:q,k}^T & \Sigma_k \\ Z_{1:q}^T & 0 \end{pmatrix}$ and find orthogonal matrices G_l and G_r so that

$$G_l H G_r = \begin{pmatrix} I_q & Y \\ 0 & \tilde{B} \end{pmatrix},$$

where $\tilde{B}$ is the upper triangular matrix of order k. Here $P_{1:q,k}^T$ denotes first q columns of P_k^T; similarly for $Z_{1:q}^T$.

4: Compute $\bar{P}_k$ and $\bar{Q}_k$ by:

$$G_l \hat{P}^T = \begin{pmatrix} I_q & 0 \\ 0 & \bar{P}_k^T \end{pmatrix}, \quad \begin{pmatrix} I_q & 0 \\ 0 & Q_k \end{pmatrix} G_r = \begin{pmatrix} I_p & 0 \\ 0 & \bar{Q}_k \end{pmatrix}.$$

5: Compute the SVD of $\tilde{B}$, $\tilde{B} = P_B\, \Sigma_B\, Q_B^T$. All matrices are square of order k.

6: *Output:* The best rank-k approximation of $\tilde{A}_k$ is given by $\tilde{P}_k = \bar{P}_k\, P_B$, $\tilde{\Sigma}_k = \Sigma_B$ and $\tilde{Q}_k^T = (\bar{Q}_k\, Q_B)^T$.

4 Kogbetliantz method for triangular matrices

Special form of the Jacobi method for obtaining the SVD of (upper or lower) triangular matrices was proposed by Kogbetliantz; see [11, 12]. However, in his original proposal the method was used for the solution of a system of linear equations, where the coefficient matrix was first transformed to a triangular form by the QR decomposition; then the R-factor was diagonalized by two-sided unitary (orthogonal, in real case) transformations. However, today his method is mainly used for the SVD computation of triangular matrices.

From the numerical point of view, the Kogbetliantz algorithm is relatively stable [8], i.e. the tiniest singular values are computed with high relative accuracy. This property is similar to the one-sided Jacobi method. The convergence criterion can be checked without any extra cost, whereas the one-sided Jacobi method requires approximately $n^2/2$ dot products to do this. However, the main weakness of the Kogbetliantz method is its need to update *both* matrix columns and rows, which means twice as many matrix multiplications as compared with an one-sided method.

We start with the serial approach and describe a special, so-called *butterfly* form of triangular matrix. The second 'brick' of the method is the modulus pivot strategy, which essentially preserves the butterfly form during the whole iterative process. Although the scalar algorithm which deals with individual elements of a matrix can be parallelized, better efficiency is achieved when working with matrix blocks because the BLAS-3 algorithms for matrix multiplication can be used. Therefore, the last subsection describes the approach when a matrix is divided into blocks and possible parallelization of the Kogbetliantz algorithm is also discussed.

4.1 Butterfly form of triangular matrices

For $n = 6$ and $n = 7$, the butterfly form of a square matrix A of order n has the following form:

$$A = \begin{pmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & x \\ x & x & x & 0 & x & x \\ x & x & x & x & x & x \\ x & x & 0 & 0 & x & x \\ x & 0 & 0 & 0 & 0 & x \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} x & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 & x \\ x & x & x & 0 & 0 & x & x \\ x & x & x & x & x & x & x \\ x & x & x & 0 & x & x & x \\ x & x & 0 & 0 & 0 & x & x \\ x & 0 & 0 & 0 & 0 & 0 & x \end{pmatrix}.$$

It is shown in [9] that each dense square (even rectangular) matrix A can be reduce to the butterfly form by a series of Householder reflections and Givens rotations applied from the left.

In our updating/downdating problems, a matrix under interest is upper or lower triangular. Let us concentrate to the upper triangular form (the lower triangular case is similar and by transposition can be brought to the upper triangular one). If T is a general, upper triangular matrix, then an example in [9] shows how T can be transformed into B, which is in the butterfly form, using a very cheap similarity transformation by a permutation matrix: $B = P^T T P$. The permutation matrix P is composed of the product of simple transposition matrices:

$$P = \begin{cases} I_{12} I_{13} (I_{14} I_{23}) (I_{15} I_{24}) (I_{16} I_{25} I_{34}) \cdot (I_{1,n} I_{2,n-1} \cdots I_{k,k+1}) & \text{if } n = 2k, \\ I_{12} I_{13} (I_{14} I_{23}) (I_{15} I_{24}) (I_{16} I_{25} I_{34}) \cdot (I_{1,n} I_{2,n-1} \cdots I_{k,k+2}) & \text{if } n = 2k+1. \end{cases}$$

Here, $I_{pq} = (e_1, \ldots, e_q, \ldots, e_p, \ldots, e_n)$, $p < q$, is the transposition of columns p and q, where e_i is the ith column of the identity matrix I_n. The parenthesis emphasize those transpositions that can be performed in parallel, because the corresponding pairs of indices are mutually disjunct. For example, for $n = 6$ we have $k = n/2 = 3$, and the transformation can be depicted as follows:

$$\begin{pmatrix} x & \star & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \end{pmatrix} \mapsto \begin{pmatrix} x & 0 & \star & x & x & x \\ x & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \end{pmatrix} \mapsto \begin{pmatrix} x & 0 & 0 & \star & x & x \\ x & x & \star & x & x & x \\ x & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \end{pmatrix}$$

$$\mapsto \begin{pmatrix} x & 0 & 0 & 0 & \star & x \\ x & x & 0 & \star & x & x \\ x & 0 & x & x & x & x \\ x & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \end{pmatrix} \mapsto \begin{pmatrix} x & 0 & 0 & 0 & 0 & \star \\ x & x & 0 & 0 & \star & x \\ x & 0 & x & \star & x & x \\ x & x & 0 & x & x & x \\ x & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \end{pmatrix} \mapsto \begin{pmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & x \\ x & x & x & 0 & x & x \\ x & x & x & x & x & x \\ x & x & 0 & 0 & x & x \\ x & 0 & 0 & 0 & 0 & x \end{pmatrix}.$$

The symbol $\star$ denotes the position of one *pivot* element, the subscripts of which define the rows and columns which are to be swapped. It can be easily seen that the whole transformation can be performed in $n - 1$ parallel steps on $n/2$ processors. This is an example of a 'fine-grained' parallelism, because the number of processors increases linearly with the matrix order n, which is not feasible for very large n.

4.2 Modulus pivot strategy

In general, the *pivot strategy* is a fixed list containing the order in which the off-diagonal matrix elements of matrix A are nullified. For triangular matrices, the special, so-called *modulus* strategy was proposed in [13]. It is defined by the *modulus ordering* of the set $\mathcal{P}_n = \{(p,q) : 1 \le p < q \le n\}$ and is illustrated below for $n = 7$.

$$\begin{pmatrix} . & 6 & 9 & 11 & 14 & 16 & 19 \\ & . & 12 & 15 & 17 & 20 & 1 \\ & & . & 18 & 21 & 2 & 4 \\ & & & . & 3 & 5 & 7 \\ & & & & . & 8 & 10 \\ & & & & & . & 13 \\ & & & & & & . \end{pmatrix} \quad \begin{array}{l} \mathcal{S}_1 = \{(2,7),(3,6),(4,5)\} \\ \mathcal{S}_2 = \{(3,7),(4,6),(1,2)\} \\ \mathcal{S}_3 = \{(4,7),(5,6),(1,3)\} \\ \mathcal{S}_4 = \{(5,7),(1,4),(2,3)\} \\ \mathcal{S}_5 = \{(6,7),(1,5),(2,4)\} \\ \mathcal{S}_6 = \{(1,6),(2,5),(3,4)\} \\ \mathcal{S}_7 = \{(1,7),(2,6),(3,5)\} \end{array} \quad \begin{pmatrix} . & 2 & 3 & 4 & 5 & 6 & 7 \\ & . & 4 & 5 & 6 & 7 & 1 \\ & & . & 6 & 7 & 1 & 2 \\ & & & . & 1 & 2 & 3 \\ & & & & . & 3 & 4 \\ & & & & & . & 5 \\ & & & & & & . \end{pmatrix}$$

The leftmost matrix represents the ordering in which the pivot elements are annihilated within one sweep. By $\mathcal{S}_t$, $1 \le t \le 7$, we denote the so-called *rotation sets* containing index pairs of matrix elements which can be annihilated *simultaneously* because all index pairs are mutually disjoint (or commuting). Finally, the rightmost matrix depicts the pivot positions according to rotation sets which can be zeroed in parallel.

Hence, the modulus pivoting for triangular matrices enables to introduce a parallel algorithm based on rotation sets. At parallel step t, the rotation set $\mathcal{S}_t$ determines which elements will be nullified. Since the Kogbetliantz method is iterative, the algorithm goes through a sequence

$$\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n, \ldots,$$

until convergence is achieved.

Let $\mathrm{Piv}(t)$ denote the *pivot set* that is currently used as a rotation set. We start with $B^{[1]} = B$, and at the beginning of time step t, $t \ge 1$, all rotation matrices (i.e., all rotation angles) $U_{ij}^{[t]}$, $V_{ij}^{[t]}$, $(i,j) \in \mathrm{Piv}(t)$ are computed using the elements of the same matrix $B^{[t]}$. Then the transformation

$$B^{[t+1]} = {U^{[t]}}^T B^{[t]} V^{[t]}, \quad U^{[t]} = \prod_{(i,j)\in \mathrm{Piv}(t)} U_{ij}^{[t]}, \quad V^{[t]} = \prod_{(i,j)\in \mathrm{Piv}(t)} V_{ij}^{[t]} \tag{2}$$

is performed. Here $U^{[t]}$ and $V^{[t]}$ are not computed explicitly; only all $V_{ij}^{[t]}$, $(i,j) \in \mathrm{Piv}(t)$ are applied simultaneously, and afterwards the same is done with $U_{ij}^{[t]}$, $(i,j) \in \mathrm{Piv}(t)$. If the right and/or left singular vectors are needed, then the right transformation $V^{[t]}$ can be accumulated into the orthogonal matrix V during iterations, and then U can be computed a posteriori from the equation $B\,V = U\,\Sigma$. Alternatively, one can accumulate $U^{[t]}$ into U and then compute V a posteriori.

The advantages of using the butterfly form together with the modulus pivot strategy in the Kogbetliantz method is discussed in detail in][9]. If B is in the butterfly norm then it is permutationally similar to the upper triangular matrix (B is PST). Therefore, it is also *essentially triangular* (ET) since it holds: $b_{pq}\, b_{qp} = 0$ for $p < q$. Moreover, it can be shown that if one starts with a triangular matrix in the butterfly form, then all matrices generated by the

Kogbetliantz method using the modulus strategy are PST. In particular, when $B^{[t]}$ denotes the iterated matrix in the time step t with $t > n$ (n is the size of B), then $B^{[t]}$ and $B^{[t-n]}$ have zero structures which are transposed to each other. Each matrix $B^{[t]}$ is PST, therefore it is ET, and can be compactly stored in the upper triangle of square array. Hence, the upper triangular matrix $G^{[t]}$ can be defined by prescription

$$G^{[t]} + G^{[t]^T} = B^{[t]} + B^{[t]^T}.$$

Then the Kogbetliantz method with modulus strategy (KMMS) can be formulated in terms of matrices $G^{[t]}$ (see [9]). The result is a sequential KMMS algorithm which works with the upper triangular matrices and in each step applies approximately $n/2$ non-commuting rotations.

These rotations can be applied in parallel, but the disadvantage of such 'direct' parallelization of the KMMS is its low efficiency. We need approximately $n/2$ processors to exploit fully the inherent parallelism of the algorithm, which is certainly not efficient for large n. The parallelization strategy, which uses the number of processors as a (linear) function of the matrix size, belongs to a 'fine-grained' approach and can be very inefficient with respect to the cost of inter-processor communication for large n. Much better way is to work with matrix blocks, whereby the size of a block is given by the size of the matrix divided by a given number of processors. Next we describe a parallelization approach for the block KMMS.

4.3 Block version and parallelism

When working with matrix blocks, the numerical algorithms become much more efficient in general, because the memory hierarchy of modern computers can be used. However, the size of matrix blocks should be tuned according to the size of fast cache memory of a processor. Ideally, the whole matrix block should fit into the cache so that no additional calls for data will be made when working with that matrix block. In this way the algorithm can use the advantage of so called BLAS-3 matrix multiplications which are very fast. They are implemented also in modern linear algebra libraries, e.g., LAPACK and ScaLAPACK. Moreover, working with matrix blocks leads to the 'coarse-grained' parallelism, in which the number of processors is given beforehand and can be quite small.

We start with the block upper triangular matrix T of order n in the form

$$T = \begin{pmatrix} T_{11} & T_{12} & \dots & T_{1m} \\ 0 & T_{22} & \dots & T_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & T_{mm}. \end{pmatrix}$$

Each diagonal block T_{ii} is of order $n_i \geq 1$, so that $\mathcal{M} = \{n_1, n_2, \dots, n_m\}$ is the partition of n. We can assume $n_1 = n_2 = \cdots = n_m = n/m$.

To reduce T to the *block butterfly form* we can use permutations similar to the scalar case. However, I_{ij} is now a product of simple transpositions The effect of $I_{ij}^T T$ is to swap block rows i and j of T; similarly, $T\, I_{ij}$ means swapping the appropriate block columns.

4.3.1 Parallel step zero

Let $B^{[0]} = B$ be a matrix in the block butterfly form. Before starting the iteration process, some matrix *preprocessing* is needed which is called the *parallel step zero*. It can be described as follows:

$$B^{[1]} = U^{[0^T]} B^{[0]} V^{[0]}, \quad U^{[0]} = \prod_{(i,j)\in \mathrm{piv}(m)} U_{ij}^{[0]}, \quad V^{[0]} = \prod_{(i,j)\in \mathrm{piv}(m)} V_{ij}^{[0]},$$

where $\mathrm{piv}(m) = \{(1,m),\, (2,m-1),\, \ldots,\, (m/2, m/2+1)\}$ is the mth *pivot set* associated with the block algorithm (notice that it is defined w.r.t. the block index).

The result of this zero step can be summarized as follows:

- The matrix blocks $B_{1,m}$, $B_{2,m-1}$, $\ldots B_{m/2,m/2+1}$ on the upper half of the main block anti-diagonal are nullified. Recall that B is in the block butterfly form so that the lower part of the main block anti-diagonal is zero.
- All diagonal blocks B_{ii} are diagonalized by computing their SVDs. For this, any numerically reliable serial SVD algorithm can be used.

Moreover, $B^{[1]}$ and all subsequent iteration matrices $B^{[2]}$, $B^{[3]}$, $\ldots$ are represented in the *factored form*

$$B^{[t]} = E^{[t^T]}\, C^{[t]}\, F^{[t]},$$

where $E^{[t]}$ and $F^{[t]}$ are block diagonal and orthogonal. The main idea behind the factorization is to work with relatively small matrix blocks in updating matrix iterates, so that all matrix multiplications can be done in the fast cache memory by calling the appropriate data only once [8]. In addition, the diagonal elements of the current iterate $B^{[t]}$ are kept separately in the vector $\gamma^{[t]}$.

Consequently, the parallel step zero must provide the formulae for computing $E^{[1]}$, $C^{[1]}$, $F^{[1]}$ and $\gamma^{[1]}$. Let $B^{[0]} = (B_1^{[0]}, B_2^{[0]}, \ldots, B_m^{[0]})$ be the block column partition of $B^{[0]}$. Then the algorithmic description of the parallel step zero is depicted below as Algorithm 4.1.

Several remarks are in order to better understand the various tricks 'behind the scene':

1. The trick with the factorization of $B^{[0]}$ into the product of three matrices, $B^{[t]} = E^{[t]^T}\, C^{[t]}\, F^{[t]}$, is taken from [10]. The main idea here is to arrive at small enough matrices with nice numerical properties (e.g., orthogonality) which can be handled in the cache memory of a processor. It is well known that the cache memory is up to $6-8$ times faster than the main memory. Therefore, even when the number of flops using the three-term recursion is larger than the direct approach to updating, the exclusive use of the cache memory can overcome this shortage w.r.t. the time complexity of the whole algorithm.
2. Consequently, all matrix multiplications in subsequent step for updating B, B' and $\bar{B}'$ are made in the cache memory—hence, they are fast.

3. The cosine-sine (CS) decomposition of an orthogonal matrix has a special structure and special properties; see [10]. It is still an open question how to compute it in a numerically reliable way for this class of matrices. Another interesting problem is its efficient parallelization.

4. The logical variable `left` controls which set of singular vectors is computed during iterations. Only one set of singular vectors is computed in step zero (and in the iterative process below), The other set is computed *a posteriori* after finishing the process by solving the linear systems of equations

$$B\,V = U\,\Sigma \quad \text{or} \quad B^T\,U = V\,\Sigma.$$

This approach almost halves the number of matrix multiplications in each iteration step as compared to the iterative computation of both sets of singular vectors.

Algorithm 4.1: Parallel step zero

1: **for** $i = 1$ to $m/2$ in parallel **do**
2: Set $j = m + 1 - i$.
3: Compute the SVD:

$$\begin{pmatrix} B_{ii}^{[0]} & B_{ij}^{[0]} \\ 0 & B_{jj}^{[0]} \end{pmatrix} = \mathbf{U}_{ij}^{[0]}\,\mathbf{\Gamma}_i\,\mathbf{V}_{ij}^{[0]T}.$$

4: Compute the CS decomposition of $\mathbf{U}_{ij}^{[0]}$ and $\mathbf{V}_{ij}^{[0]}$:

$$\mathbf{U}_{ij}^{[0]} = \begin{pmatrix} \dot{U}_{ii} & 0 \\ 0 & \dot{U}_{jj} \end{pmatrix} \Theta_{ij} \begin{pmatrix} \ddot{U}_{ii} & 0 \\ 0 & \ddot{U}_{jj} \end{pmatrix}, \; \mathbf{V}_{ij}^{[0]} = \begin{pmatrix} \dot{V}_{ii} & 0 \\ 0 & \dot{V}_{jj} \end{pmatrix} \Phi_{ij} \begin{pmatrix} \ddot{V}_{ii} & 0 \\ 0 & \ddot{V}_{jj} \end{pmatrix}.$$

5: Apply: $B_i' = B_i \dot{V}_{ii}$, $B_{ij}' = B_j \dot{V}_{jj}$.
6: Apply: $(B_i'', B_j'') = (B_i', B_j')\,\Phi_{ij}$.
7: Transpose: $\bar{B} = (B'')^T$ and let $\bar{B} = (\bar{B}_1, \bar{B}_2, \ldots, \bar{B}_m)$ be the block column partition of $\bar{B}$.
8: Apply: $\bar{B}_i' = \bar{B}_i\,\dot{U}_{ii}$, $\bar{B}_j' = \bar{B}_j\,\dot{U}_{jj}$.
9: Apply: $(\bar{B}_i'', \bar{B}_j'') = (\bar{B}_i', \bar{B}_j')\,\Theta_{ij}$.
10: Transpose: $C^{[1]} = (\bar{B}'')^T$.
11: Copy: $E_{ii}^{[1]} = \ddot{U}_{ii}$, $E_{jj}^{[1]} = \ddot{U}_{jj}$, $F_{ii}^{[1]} = \ddot{V}_{ii}$, $F_{jj}^{[1]} = \ddot{V}_{jj}$.
12: Copy the first n_i and last n_j diagonal elements of $\mathbf{\Gamma}_i$ into the appropriate parts of the vector $\gamma^{[1]}$.
13: **if** (`left`) **then**
14: $U^{[1]} = {E^{[1]}}^T$
15: **else**
16: $V^{[1]} = {F^{[1]}}^T$
17: **end if**
18: **end for**

4.3.2 Iterative process

Recall that after the parallel step zero the matrix $B^{[1]}$ is is the block-butterfly form. In the iterative process, the block-modulus pivot strategy is applied in each parallel step until convergence. Thus, the algorithm in the parallel step t proceeds by annihilating the off-diagonal pivot submatrices $B_{ij}^{[t]}$, $(i,j) \in \mathrm{piv}(t)$ and by diagonalizing the diagonal blocks $B_{ii}^{[t]}$, $B_{jj}^{[t]}$. Hence, at the beginning of the parallel step $t+1$ the new Frobenius off-norm is given by

$$\|\Omega(B^{[t+1]}\|^2 = \|\Omega(B^{[t]}\|^2 - \sum_{(i,j)\in \mathrm{piv}(t)} \|B_{ij}^{[t]}\|^2.$$

Since $B^{[t]}$ is kept in the factored form of a matrix triple $E^{[t]}$, $C^{[t]}$, $F^{[t]}$, one has to derive the recursions for updating these matrices together with vector $\gamma^{[t]}$.

The main equation of the Kogbetliantz method is given by (2). For each pair $(i,j) \in \mathrm{piv}(t)$ this orthogonal transformation can be written as

$$\mathbf{B}_{ij}^{[t]} = \begin{pmatrix} B_{ii}^{[t]} & B_{ij}^{[t]} \\ 0 & B_{jj}^{[t]} \end{pmatrix} = \mathbf{U}_{ij}^{[t]}\, \mathbf{\Gamma}_i\, \mathbf{V}_{ij}^{[t]^T}, \quad \mathbf{\Gamma}_i \text{ is diagonal.} \tag{3}$$

Notice that this equation is the SVD of $\mathbf{B}_{ij}^{[t]}$. We assume that the diagonal blocks $B_{ii}^{[t]}$ and $B_{jj}^{[t]}$ are diagonal matrices which is certainly true, by construction, for the initial matrix $B^{[1]}$. Here, $\mathbf{U}_{ij}^{[t]}$ and $\mathbf{V}_{ij}^{[t]}$ are orthogonal matrices of order $(n_i+n_j)\times(n_i+n_j)$; they are called *block rotations* in [10].

However, $B^{[t]}$ is given in its factored form, so that the upper-triangular matrix $\mathbf{B}_{ij}^{[t]}$ can be computed as follows:

$$\begin{pmatrix} B_{ii}^{[t]} & B_{ij}^{[t]} \\ 0 & B_{jj}^{[t]} \end{pmatrix} = \begin{pmatrix} E_{ii}^{[t]} & 0 \\ 0 & E_{jj}^{[t]} \end{pmatrix}^T \begin{pmatrix} C_{ii}^{[t]} & C_{ij}^{[t]} \\ 0 & C_{jj}^{[t]} \end{pmatrix} \begin{pmatrix} F_{ii}^{[t]} & 0 \\ 0 & F_{jj}^{[t]} \end{pmatrix} = \begin{pmatrix} E_{ii}^{[t]^T} C_{ii}^{[t]} F_{ii}^{[t]} & E_{ii}^{[t]^T} C_{ij}^{[t]} F_{jj}^{[t]} \\ 0 & E_{jj}^{[t]^T} C_{jj}^{[t]} F_{jj}^{[t]} \end{pmatrix}.$$

Since $B_{ii}^{[t]}$ and $B_{jj}^{[t]}$ are diagonal, we can fill them by zeros and then copy appropriate diagonal elements from the vector $\gamma^{[t]}$ onto the diagonal of $B_{ii}^{[t]}$ and $B_{jj}^{[t]}$ (hence, the diagonal blocks of $\mathbf{B}_{ij}^{[t]}$ are *not* computed explicitly). After that we need to compute $E_{ii}^{[t]^T} C_{ij}^{[t]} F_{jj}^{[t]}$ on the processor which is associated with the pair $(i,j) \in \mathrm{piv}(t)$ using the fast BLAS-3 LAPACK routine `*GEMM`.

Next, the SVD of $\mathbf{B}_{ij}^{[t]}$ is computed according to (3). Since $\mathbf{B}_{ij}^{[t]}$ is upper triangular with diagonal blocks being diagonal matrices, one can here choose among several fast and accurate serial methods, e.g., the one-sided Jacobi or (cyclic or modulus) Kogbetliantz algorithm. This SVD is computed serially for one $\mathbf{B}_{ij}^{[t]}$, but, of course, $m/2$ processors compute in parallel for $m/2$ pairs of indices (i,j), $1 \le i < j \le n$.

The next step is the CS decomposition of orthogonal matrices $\mathbf{U}_{ij}^{[t]}$ and $\mathbf{V}_{ij}^{[t]}$, which can be written in the form (see [10])

$$\mathbf{U}_{ij}^{[t]} = \begin{pmatrix} \dot{U}_{ii}^{[t]} & 0 \\ 0 & \dot{U}_{jj}^{[t]} \end{pmatrix} \Theta_{ij}^{[t]} \begin{pmatrix} \ddot{U}_{ii}^{[t]} & 0 \\ 0 & \ddot{U}_{jj}^{[t]} \end{pmatrix}, \quad \mathbf{V}_{ij}^{[t]} = \begin{pmatrix} \dot{V}_{ii}^{[t]} & 0 \\ 0 & \dot{V}_{jj}^{[t]} \end{pmatrix} \Phi_{ij}^{[t]} \begin{pmatrix} \ddot{V}_{ii}^{[t]} & 0 \\ 0 & \ddot{V}_{jj}^{[t]} \end{pmatrix}. \tag{4}$$

The matrices $\Theta_{ij}^{[t]}$ and $\Phi_{ij}^{[t]}$ are orthogonal products of at most $\min\{n_i, n_j\}$ commuting plane rotations (see [10]).

Now comes the parallel computation of the next iteration matrix, $B^{[t+1]}$, using all available pairs (i, j). It is computed as

$$B^{[t+1]} = U^{[t]^T} (B^{[t]} V^{[t]}),$$

where $U^{[t]}$ and $V^{[t]}$ is composed from all available matrices $\mathbf{U}_{ij}^{[t]}$ and $\mathbf{V}_{ij}^{[t]}$, respectively. However, $B^{[t+1]}$ is never computed explicitly. Recall that we have it in the factored form, so actually we need recursions how to compute $E^{[t+1]}$, $C^{[t+1]}$ and $F^{[t+1]}$. To this end, let us introduce the matrix $J_{ij} = (J_i, J_j)$ where $I_n = (J_1, J_2, \ldots, J_m)$ is the block-column partition of the identity. Then

$$B^{[t]} J_{ij} = (B_i^{[t]}, B_j^{[t]}).$$

and

$$B^{[t+1]} \equiv E^{[t+1]^T} C^{[t+1]} F^{[t+1]} = U^{[t]^T} (E^{[t]^T} C^{[t]} F^{[t]}) V^{[t]}.$$

Post-multiplying the last equality above by J_{ij} and writing the identity between C and F on both sides as $J_{ij}^T J_{ij}$, we obtain:

$$\begin{aligned} E^{[t+1]^T} (C_i^{[t+1]}, C_j^{[t+1]}) \begin{pmatrix} F_{ii}^{[t+1]} & 0 \\ 0 & F_{jj}^{[t+1]} \end{pmatrix} &= U^{[t]^T} E^{[t]^T} (C_i^{[t]}, C_j^{[t]}) \begin{pmatrix} F_{ii}^{[t]} & 0 \\ 0 & F_{jj}^{[t]} \end{pmatrix} \mathbf{V}_{ij}^{[t]} \\ &= U^{[t]^T} E^{[t]^T} (C_i^{[t]}, C_j^{[t]}) \left[\left[\begin{pmatrix} F_{ii}^{[t]} & 0 \\ 0 & F_{jj}^{[t]} \end{pmatrix} \begin{pmatrix} \dot{V}_{ii}^{[t]} & 0 \\ 0 & \dot{V}_{jj}^{[t]} \end{pmatrix} \right] \Phi_{ij}^{[t]} \right] \begin{pmatrix} \ddot{V}_{ii}^{[t]} & 0 \\ 0 & \ddot{V}_{jj}^{[t]} \end{pmatrix}. \end{aligned}$$

Hence, we have immediately the first set of updates:

$$F_{ii}^{[t+1]} = \ddot{V}_{ii}^{[t]},\ F_{jj}^{[t+1]} = \ddot{V}_{jj}^{[t]},\ (\bar{C}_i^{[t]}, \bar{C}_j^{[t]}) = (C_i^{[t]}, C_j^{[t]}) \begin{pmatrix} F_{ii}^{[t]} \dot{V}_{ii}^{[t]} & 0 \\ 0 & F_{jj}^{[t]} \dot{V}_{jj}^{[t]} \end{pmatrix} \Phi_{ij}^{[t]}. \tag{5}$$

These updates can be performed in parallel for all $(i, j) \in \mathrm{piv}(t)$. This results in the matrix $F^{[t+1]}$ and auxiliary matrix $\bar{C}$.

The second set of updates starts with the equation

$$E^{[t+1]^T} C^{[t+1]} = U^{[t]^T} E^{[t]^T} \bar{C}^{[t]}.$$

After pre-multiplying it by J_{ij}^T ($J_i^T X$ is the ith block-row of X) and using the decomposition of identity $I = J_{ij} J_{ij}^T$ between E and C on both sides, we obtain:

$$\begin{aligned} \begin{pmatrix} E_{ii}^{[t+1]} & 0 \\ 0 & E_{jj}^{[t+1]} \end{pmatrix}^T \begin{pmatrix} J_i^T C^{[t+1]} \\ J_j^T C^{[t+1]} \end{pmatrix} &= \mathbf{U}_{ij}^{[t]^T} \begin{pmatrix} E_{ii}^{[t]} & 0 \\ 0 & E_{jj}^{[t]} \end{pmatrix}^T \begin{pmatrix} J_i^T \bar{C}^{[t]} \\ J_j^T \bar{C}^{[t]} \end{pmatrix} \\ &= \begin{pmatrix} \ddot{U}_{ii}^{[t]} & 0 \\ 0 & \ddot{U}_{jj}^{[t]} \end{pmatrix}^T \left[\Theta_{ij}^{[t]^T} \begin{pmatrix} \dot{U}_{ii}^{[t]} & 0 \\ 0 & \dot{U}_{jj}^{[t]} \end{pmatrix}^T \begin{pmatrix} E_{ii}^{[t]} & 0 \\ 0 & E_{jj}^{[t]} \end{pmatrix}^T \begin{pmatrix} J_i^T \bar{C}^{[t]} \\ J_j^T \bar{C}^{[t]} \end{pmatrix} \right], \end{aligned}$$

and we have the second set of updates:

$$E_{ii}^{[t+1]} = \ddot{U}_{ii}^{[t]},\ E_{jj}^{[t+1]} = \ddot{U}_{jj}^{[t]},\ (C_i^{[t+1]^T}, C_j^{[t+1]^T}) = (\bar{C}_i^{[t]^T}, \bar{C}_j^{[t]^T}) \begin{pmatrix} E_{ii}^{[t]} \dot{U}_{ii}^{[t]} & 0 \\ 0 & E_{jj}^{[t]} \dot{U}_{jj}^{[t]} \end{pmatrix} \Theta_{ij}^{[t]}. \tag{6}$$

Again, these updates can be performed in parallel for all $(i,j) \in \mathrm{piv}(t)$. Notice, that the auxiliary matrix $\bar{C}^{[t]}$, which was computed in (5), is to be transposed in (6). Then it is updated from the right hand side and $C^{[t+1]^T}$ is obtained. Finally, the diagonal elements of $\mathbf{\Gamma}_i$ are copied to the appropriate positions of the vector $\gamma^{[t+1]}$.

Equations (3), (4), (5) and (6) constitute one iterative step of the parallel block-Kogbetliantz algorithm. The parallelism is achieved by computing all updates for pivot indices $(i,j) \in \mathrm{piv}(t)$ simultaneously. This means that there are $p = m/2$ processors (recall that m is the blocking factor) whereby each processor works over 2 block columns of matrix data.

We write the iterative part in the form of a pseudocode. The iteration index $[t]$ is omitted. The array C is $n \times n$, arrays E and F are $\mathrm{nb} \times \mathrm{nb}$ where $\mathrm{nb} = \max_i\{n_i\}$. The block-column partition of C is given by $C = (C_1, \ldots, C_m)$. We denote $E_{ii}^{[t]}$ by E_i and similarly for F_i. The vector g is for $\gamma^{[t]}$ and several arrays U, B, V are of size $2\mathrm{nb} \times 2\mathrm{nb}$. The matrices of left or right singular vectors are updated in arrays VECL or VECR, respectively, according to the logical variables `left` and `right`.

Algorithm 4.2: Iteration step

1: **for** $(i,j) \in \mathrm{piv}(t)$ in parallel **do**
2: Compute: $B_{12} = E_i^T\, C_{ij}\, F_j$.
3: Copy the appropriate elements from g to $\mathrm{diag}(B_{11})$ and $\mathrm{diag}(B_{22})$.
4: Form: $B = \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix}$ where diagonal blocks are diagonal matrices.
5: Compute the SVD: $B = U\,\Gamma\,V^T$.
6: Update: $g \leftarrow \Gamma$ (copy to appropriate positions).
7: Compute the CS decompositions

$$U = \begin{pmatrix} \mathrm{U1} & 0 \\ 0 & \mathrm{U2} \end{pmatrix} H \begin{pmatrix} \mathrm{U3} & 0 \\ 0 & \mathrm{U4} \end{pmatrix}, \quad V = \begin{pmatrix} \mathrm{V1} & 0 \\ 0 & \mathrm{V2} \end{pmatrix} K \begin{pmatrix} \mathrm{V3} & 0 \\ 0 & \mathrm{V4} \end{pmatrix}.$$

8: Compute: $X = F_i\,\mathrm{V1}, \quad Y = F_j\,\mathrm{V2}$.
9: Update block columns of C: $C_i \leftarrow C_i X, \quad C_j \leftarrow C_j Y$.
10: If (`right`) update: $\mathrm{VECR}_i \leftarrow \mathrm{VECR}_i\, X, \quad \mathrm{VECR}_j \leftarrow \mathrm{VECR}_j\, Y$.
11: Update: $(C_i, C_j) \leftarrow (C_i, C_j)\, K$.
12: If (`right`) update: $(\mathrm{VECR}_i, \mathrm{VECR}_j) \leftarrow (\mathrm{VECR}_i, \mathrm{VECR}_j)\, K$.
13: Update: $F_i \leftarrow \mathrm{V3}, \quad F_j \leftarrow \mathrm{V4}$.
14: Transpose: $C \leftarrow C^T$.
15: Compute: $X = E_i\,\mathrm{U1}, \quad Y = E_j\,\mathrm{U2}$.
16: Update block columns of C^T (i.e., block rows of C): $C_i \leftarrow C_i X, \quad C_j \leftarrow C_j Y$.
17: If (`left`) update: $\mathrm{VECL}_i \leftarrow \mathrm{VECL}_i\, X, \quad \mathrm{VECL}_j \leftarrow \mathrm{VECL}_j\, Y$.
18: Update: $(C_i, C_j) \leftarrow (C_i, C_j)\, H$.
19: If (`left`) update: $(\mathrm{VECL}_i, \mathrm{VECL}_j) \leftarrow (\mathrm{VECL}_i, \mathrm{VECL}_j)\, H$.
20: Update: $E_i \leftarrow \mathrm{U3}, \quad E_j \leftarrow \mathrm{U4}$.
21: Transpose: $C \leftarrow C^T$ (back to the original form of C).
22: **end for**

Similarly to the parallel step zero, several comments are in order also for the iterative part of

the algorithm:

1. The recursions have one important purpose—namely, to arrive at small enough matrices (or matrix blocks) which can be stored *at once* in the fast cache memory. These are the diagonal blocks of E, F and all factors of the CS decompositions. These all are square matrices of size nb, which is substantially smaller than the size n of the original upper triangular matrix provided that the blocking factor m is large enough. Perhaps more importantly, given n, the number of processors p and the blocking factor m ($p = m/2$) can be chosen in such way that all small blocks will indeed be stored in the cache memory at once. This means great time savings in computing updates by matrix multiplications.

2. The only matrix without any structure used in recursions is C. We see that C is updated in two steps, whereby the second update works with C^T. The reason is that using the transposition one can update *both* block columns and block rows by matrix multiplications *from the right*, whereby the updating matrices X and Y are small and should fit in the cache. Such updating will be very fast. If each processor contains two full block columns of C, then all updates can be computed *locally* in processors and there is no need to use the distributed matrix multiplication. The price paid for this 'comfort' is the need of two transpositions of C which can be slow on distributed parallel architectures (e.g., on a cluster of PCs). In other words, one needs some fast, parallel (distributed) algorithm for the matrix transposition.

5 Conclusions

We have shown that the updating/downdating problems in the LSI can be reduced to the computation of SVDs of upper or lower triangular matrices. For this purpose, the parallel block-Kogbetliantz algorithm was described and analyzed from the point of view of its implementation on a parallel distributed architecture. Next step should be its actual implementation on a cluster of PCs.

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), 265-287.

[3] M. Bečka and M. Vajteršic, *Block-Jacobi SVD algorithms for distributed memory systems: II. Meshes*, Parallel Algorithms Appl., 14 (1999), 37-56.

[4] M. Bečka, G. Okša and M. Vajteršic, *Dynamic ordering for a parallel block-Jacobi SVD algorithm*, Parallel Computing, 28 (2002), 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), 335-362.

[7] A. Björck, *Numerical Methods for Least Squares Problems*, First ed., SIAM, Philadelphia, PA, 1996.

[8] V. Hari and J. Matejaš, *Accuracy of the Kogbetliantz method*, preprint, University of Zagreb, 2005.

[9] V. Hari and V. Zadelj-Martič, *Parallelizing Kogbetliantz method*, accepted for publication at Int. Conf. on Numerical Analysis and Scientific Computation, Rhodos, Greece, September 2006.

[10] V. Hari, *Accelerating the SVD block-Jacobi method*, Computing, 75 (2005), 27-53.

[11] E. Kogbetliantz, *Diagonalization of general complex matrices as a new method for solution of linear equations*, Proc. Intern. Congr. Math. Amsterdam 2 (1954), 356-357.

[12] E. Kogbetliantz, *Solutions of linear equations by diagonalization of coefficient matrices*, Quart. Appl. Math. 13 (1955), 123-132.

[13] F. T. Luk and H. Park, *On parallel Jacobi orderings*, SIAM J. Sci. Statist. Comput. 10 (1989), 18-26.

[14] D. I. Witter and M. W. Berry, *Downdating the latent semantic indexing model for conceptual information retrieval*, The Computer Journal, 41 (1998), 589-601.

[15] 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, 29-42.

[16] H. Zha and H. D. Simon, *On updating problems in latent semantic indexing*, SIAM J. Sci. Comput., 21 (1999), 782-791.