Blocking and parallelization of the Hari–Zimmermann variant of the Falk–Langemeyer algorithm for the generalized SVD☆

Vedran Novaković[a], Sanja Singer[b,*], Saša Singer[c]

[a] *Science and Technology Facilities Council, Daresbury Laboratory, Sci-Tech Daresbury, Warrington WA4 4AD, United Kingdom*
[b] *University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture, Ivana Lučića 5, 10000 Zagreb, Croatia*
[c] *University of Zagreb, Faculty of Science, Department of Mathematics, Bijenička cesta 30, 10000 Zagreb, Croatia*

Abstract

The paper describes how to modify the two-sided Hari–Zimmermann algorithm for computation of the generalized eigenvalues of a matrix pair (A, B), where B is positive definite, to an implicit algorithm that computes the generalized singular values of a pair (F, G). In addition, we present blocking and parallelization techniques for speedup of the computation.

For triangular matrix pairs of a moderate size, numerical tests show that the double precision sequential pointwise algorithm is several times faster than the Lapack `DTGSJA` algorithm, while the accuracy is slightly better, especially for small generalized singular values.

Cache-aware algorithms, implemented either as the block-oriented, or as the full block algorithm, are several times faster than the pointwise algorithm. The algorithm is almost perfectly parallelizable, so parallel shared memory versions of the algorithm are perfectly scalable, and their speedup almost solely depends on the number of cores used. A hybrid shared/distributed memory algorithm is intended for huge matrices that do not fit into the shared memory.

Keywords: generalized singular value problem, blocking, parallelization, generalized eigenvalue problem
2000 MSC: 65Y05, 65Y20, 65F15

1. Introduction

The singular value decomposition (SVD) is a widely used tool in many applications. Similarly, a generalization of the SVD for a matrix pair (F, G), the generalized SVD (GSVD), has applications in many areas, such as comparative analysis of the genome-scale expression data sets [1], incomplete singular boundary element method [21], ionospheric tomography [4], and many others.

The GSVD of a pair (F, G) is closely related to the Hermitian generalized eigenvalue problem (GEP) of a pair $(A, B) := (F^*F, G^*G)$, so the methods for simultaneous diagonalization of (A, B) can be modified to compute the GSVD of (F, G). Our aim is to develop a fast and efficient parallel algorithm for the real pair (F, G).

The definition of the GSVD (see, for example, [27]), in its full generality, is as follows: for given matrices $F \in \mathbb{C}^{m\times n}$ and $G \in \mathbb{C}^{p\times n}$, where

$$K = \begin{bmatrix} F \\ G \end{bmatrix}, \quad k = \operatorname{rank}(K),$$

there exist unitary matrices $U \in \mathbb{C}^{m\times m}$, $V \in \mathbb{C}^{p\times p}$, and a matrix $X \in \mathbb{C}^{k\times n}$, such that

$$F = U\Sigma_F X, \quad G = V\Sigma_G X, \quad \Sigma_F \in \mathbb{R}^{m\times k}, \quad \Sigma_G \in \mathbb{R}^{p\times k}. \tag{1.1}$$

☆ The Version of Record is available online at `https://doi.org/10.1016/j.parco.2015.06.004`
* Corresponding author.
Email addresses: `vedran.novakovic@stfc.ac.uk` (Vedran Novaković), `ssinger@fsb.hr` (Sanja Singer), `singer@math.hr` (Saša Singer)

The elements of Σ_F and Σ_G are zeroes, except for the diagonal entries, which are real and nonnegative. Furthermore, Σ_F and Σ_G satisfy

$$\Sigma_F^T \Sigma_F + \Sigma_G^T \Sigma_G = I.$$

The ratios $(\Sigma_F)_{ii}/(\Sigma_G)_{ii}$ are called the generalized singular values of the pair (F, G). If G is of full column rank, then $\mathrm{rank}(K) = n$, and the generalized singular values are finite numbers. If the pair (F, G) is real, then all matrices in (1.1) are real. From now on, we assume that all matrices are real.

In 1960, Falk and Langemeyer published two papers [11, 12] on the computation of the GEP,

$$Ax = \lambda Bx, \quad x \neq 0, \tag{1.2}$$

where A and B are symmetric matrices and the pair is definite, i.e., there exists a real constant μ such that the matrix $A - \mu B$ is positive definite.

Note that, if B is positive definite, then the pair (A, B) is definite. This can be proved easily by using the Weyl theorem (see, for example, [19, Theorem 4.3.1, page 181]). In this case,

$$\lambda_i(A - \mu B) \geq \lambda_i(A) - \mu \lambda_{\min}(B), \tag{1.3}$$

where $\lambda_i(\cdot)$ denotes the i-th smallest eigenvalue of a matrix. If A is positive definite, any $\mu \leq 0$ will make the pair definite. If A is indefinite, or negative definite, it is sufficient to choose $\mu < 0$ in (1.3) such that $\lambda_{\min}(A) - \mu \lambda_{\min}(B) > 0$.

Almost a decade after Falk and Langemeyer, Zimmermann in her Ph.D. thesis [38] briefly sketched a new method for the problem (1.2) if B is positive definite. Gose [13] proposed some optimal non-cyclic pivot strategies and proved the global convergence of the original method. Hari in his Ph.D. thesis [14] filled in the missing details of the method of Zimmermann, and proved its global and quadratic convergence under the cyclic pivot strategies.

The quadratic convergence of the original Falk–Langemeyer method was proved in 1988, by Slapničar in his MS thesis, four years after the proof of the convergence of the Hari–Zimmermann method, and the results were extended afterwards and published in [33]. In the same paper, Slapničar and Hari also showed the following connection between the Hari–Zimmermann and the Falk–Langemeyer variants of the method. If the matrix B is scaled (from both sides) so that its diagonal elements are equal to 1, before each annihilation step in the Falk–Langemeyer method, then we get the Hari–Zimmermann method.

The GSVD was introduced by Van Loan [37] and Paige and Saunders [27]. If $k = n$ in (1.1), then the relation between the GSVD and the reduced form of the CS (cosine-sine) decomposition (see, for example, [9]) could be used for the computation of the GSVD; see the papers by Stewart [34, 35] and Sutton [36].

Drmač in [9] derived an algorithm for the computation of the GSVD of a pair (F, G), with G of full column rank. This algorithm transforms the problem to a single matrix, and then applies the ordinary Jacobi SVD algorithm. The algorithm produces the generalized singular values with small relative errors. The part of the algorithm that reduces the problem to a single matrix consists of three steps: a scaling of the columns of both matrices F and G, the QR factorization with pivoting of the already scaled G, and, finally, a solution of a triangular linear system with k right-hand sides. The last two steps are inherently sequential and, therefore, hard to parallelize.

The idea of using an implicit (i.e., one-sided) version of the Falk–Langemeyer method for the GSVD of a pair (F, G), with G of full column rank, can be found in the Ph.D. thesis of Deichmöller [7], but there is no comment on how this method performs in comparison with the other methods.

On the other hand, the state of the art algorithm `xGGSVD` for computing the GSVD in Lapack, is a Kogbetliantz-based variation of the Paige algorithm [26] by Bai and Demmel [2], with preprocessing. The preprocessing algorithm [3] transforms a given matrix pair (F_0, G_0) to a pair (F, G), such that F and G are upper triangular and G is nonsingular. If it is known in advance that G is of full column rank, both implicit methods, the Falk–Langemeyer and the Hari–Zimmermann method will work without preprocessing. But, if F and G are tall and skinny, the QR factorization of both matrices will speed up the orthogonalization. If G is not of full column rank, the same preprocessing technique (as in Lapack) should be used by our implicit algorithm, since a full column rank G guarantees that $B := G^T G$ is positive definite.

Finally, Matejaš in [23, Theorem 8] showed that one step of the Falk–Langemeyer method is backward stable for the definite matrix pair (A, B). Since the difference between the Hari–Zimmermann and the Falk–Langemeyer method is only in the two-sided scaling before each step, a similar analysis could be done for the original Hari–Zimmermann

method. The backward stability of the implicit Hari–Zimmermann algorithm for the GSVD is proved in [29], by using a different columnwise technique. Together with the results from [8], it also provides a bound for the relative accuracy of the computed generalized singular values.

Section 2 contains a description of two pointwise Jacobi-type algorithms for the GEP: the Falk–Langemeyer and the Hari–Zimmermann algorithm. The one-sided analogue of the Hari–Zimmermann algorithm for the GSVD is presented at the beginning of Section 3. Then we describe the algorithms that orthogonalize a matrix pair block-by-block, to maximize the efficiency on hierarchical memory architectures. Parallel algorithms on shared and distributed memory architectures are described in Section 4. The results of numerical testing are presented and discussed in Section 5.

2. Jacobi–type sequential algorithms for GEP

2.1. The Falk–Langemeyer algorithm

The Falk–Langemeyer method solves the GEP (1.2) for a symmetric and definite matrix pair (A, B). The method constructs a sequence of matrix pairs $(A^{(\ell)}, B^{(\ell)})$,

$$A^{(\ell+1)} = C_\ell^T A^{(\ell)} C_\ell, \quad B^{(\ell+1)} = C_\ell^T B^{(\ell)} C_\ell, \quad \ell \in \mathbb{N}, \tag{2.1}$$

where $A^{(1)} := A$, and $B^{(1)} := B$, according to some pivot strategy that selects the order in which the off-diagonal elements are annihilated. If the transformation matrix C_ℓ is nonsingular, the two consecutive pairs are congruent, so their eigenvalues are equal [28]. A matrix that diagonalizes the pair is computed by accumulating the transformations

$$C^{(1)} = I, \qquad C^{(\ell+1)} = C^{(\ell)} C_\ell, \quad \ell = 1, 2, \ldots. \tag{2.2}$$

The matrix C_ℓ resembles a scaled plane rotation: it is the identity matrix, except for its (i, j)-restriction $\widehat{C}_\ell$, where it has two parameters, α_ℓ and β_ℓ

$$\widehat{C}_\ell = \begin{bmatrix} 1 & \alpha_\ell \\ -\beta_\ell & 1 \end{bmatrix}. \tag{2.3}$$

The parameters α_ℓ and β_ℓ in (2.3) are determined so that the transformations in (2.1) diagonalize the pivot submatrices

$$\widehat{A}^{(\ell)} = \begin{bmatrix} a_{ii}^{(\ell)} & a_{ij}^{(\ell)} \\ a_{ij}^{(\ell)} & a_{jj}^{(\ell)} \end{bmatrix}, \qquad \widehat{B}^{(\ell)} = \begin{bmatrix} b_{ii}^{(\ell)} & b_{ij}^{(\ell)} \\ b_{ij}^{(\ell)} & b_{jj}^{(\ell)} \end{bmatrix}. \tag{2.4}$$

The annihilation equations are

$$\begin{bmatrix} 1 & -\beta_\ell \\ \alpha_\ell & 1 \end{bmatrix} \begin{bmatrix} a_{ii}^{(\ell)} & a_{ij}^{(\ell)} \\ a_{ij}^{(\ell)} & a_{jj}^{(\ell)} \end{bmatrix} \begin{bmatrix} 1 & \alpha_\ell \\ -\beta_\ell & 1 \end{bmatrix} = \begin{bmatrix} a_{ii}^{(\ell+1)} & 0 \\ 0 & a_{jj}^{(\ell+1)} \end{bmatrix}, \quad \begin{bmatrix} 1 & -\beta_\ell \\ \alpha_\ell & 1 \end{bmatrix} \begin{bmatrix} b_{ii}^{(\ell)} & b_{ij}^{(\ell)} \\ b_{ij}^{(\ell)} & b_{jj}^{(\ell)} \end{bmatrix} \begin{bmatrix} 1 & \alpha_\ell \\ -\beta_\ell & 1 \end{bmatrix} = \begin{bmatrix} b_{ii}^{(\ell+1)} & 0 \\ 0 & b_{jj}^{(\ell+1)} \end{bmatrix}. \tag{2.5}$$

In terms of α_ℓ and β_ℓ, the equations (2.5) reduce to

$$\alpha_\ell a_{ii}^{(\ell)} + (1 - \alpha_\ell\beta_\ell) a_{ij}^{(\ell)} - \beta_\ell a_{jj}^{(\ell)} = 0, \quad \alpha_\ell b_{ii}^{(\ell)} + (1 - \alpha_\ell\beta_\ell) b_{ij}^{(\ell)} - \beta_\ell b_{jj}^{(\ell)} = 0,$$

and the solution can be written as

$$\alpha_\ell = \frac{I_j^{(\ell)}}{\nu_\ell}, \quad \beta_\ell = \frac{I_i^{(\ell)}}{\nu_\ell},$$

where

$$I_i^{(\ell)} = a_{ii}^{(\ell)} b_{ij}^{(\ell)} - a_{ij}^{(\ell)} b_{ii}^{(\ell)}, \qquad I_{ij}^{(\ell)} = a_{ii}^{(\ell)} b_{jj}^{(\ell)} - a_{jj}^{(\ell)} b_{ii}^{(\ell)},$$
$$I_j^{(\ell)} = a_{jj}^{(\ell)} b_{ij}^{(\ell)} - a_{ij}^{(\ell)} b_{jj}^{(\ell)}, \qquad I^{(\ell)} = (I_{ij}^{(\ell)})^2 + 4 I_i^{(\ell)} I_j^{(\ell)},$$
$$\nu_\ell = \frac{1}{2} \operatorname{sign}(I_{ij}^{(\ell)}) \Big(|I_{ij}^{(\ell)}| + \sqrt{I^{(\ell)}} \Big).$$

If $I^{(\ell)} = 0$, then a special set of formulas is used (for details, see [33, Algorithm 4]).

Now suppose that the matrices in (2.1) are computed according to the IEEE 754–2008 standard [20] for floating–point arithmetic. Unfortunately, for large matrices, after a certain number of transformations, the elements of both $A^{(\ell)}$ and $B^{(\ell)}$ can become huge (represented as `Inf`). A solution to this problem is an occasional rescaling, but how often that needs to be done, depends on the dimension of the pair, the size of its elements, and the chosen pivot strategy.

2.2. The Hari–Zimmermann algorithm

If B is positive definite, the initial pair (A, B) can be scaled so that the diagonal elements of B are all equal to one. By taking

$$D = \operatorname{diag}\left(\frac{1}{\sqrt{b_{11}}}, \frac{1}{\sqrt{b_{22}}}, \ldots, \frac{1}{\sqrt{b_{kk}}}\right),$$

and making the congruence transformations

$$A^{(1)} := DAD, \quad B^{(1)} := DBD,$$

we obtain a new pair $(A^{(1)}, B^{(1)})$ which is congruent to the original pair.

The idea behind this modification of the Falk–Langemeyer algorithm is: when B is the identity matrix, the transformations are the ordinary Jacobi rotations. The method constructs a sequence of matrix pairs

$$A^{(\ell+1)} = Z_\ell^T A^{(\ell)} Z_\ell, \quad B^{(\ell+1)} = Z_\ell^T B^{(\ell)} Z_\ell, \quad \ell \in \mathbb{N}, \tag{2.6}$$

such that the diagonal elements of $B^{(\ell)}$ remain ones after each transformation. The matrices Z_ℓ in (2.6) are chosen to annihilate the elements at positions (i, j) (and (j, i)), and to keep ones as the diagonal elements of $B^{(\ell)}$. If the matrix that diagonalizes the pair is needed, the accumulation procedure is given by (2.2), with C replaced by Z.

The matrix Z_ℓ is the identity matrix, except in the plane (i, j), where its restriction $\widehat{Z}_\ell$ is equal to

$$\widehat{Z}_\ell = \frac{1}{\sqrt{1 - (b_{ij}^{(\ell)})^2}} \begin{bmatrix} \cos\varphi_\ell & \sin\varphi_\ell \\ -\sin\psi_\ell & \cos\psi_\ell \end{bmatrix}, \tag{2.7}$$

with

$$\begin{aligned} \cos\varphi_\ell &= \cos\vartheta_\ell + \xi_\ell(\sin\vartheta_\ell - \eta_\ell\cos\vartheta_\ell), & \cos\psi_\ell &= \cos\vartheta_\ell - \xi_\ell(\sin\vartheta_\ell + \eta_\ell\cos\vartheta_\ell), \\ \sin\varphi_\ell &= \sin\vartheta_\ell - \xi_\ell(\cos\vartheta_\ell + \eta_\ell\sin\vartheta_\ell), & \sin\psi_\ell &= \sin\vartheta_\ell + \xi_\ell(\cos\vartheta_\ell - \eta_\ell\sin\vartheta_\ell), \\ \xi_\ell &= \frac{b_{ij}^{(\ell)}}{\sqrt{1 + b_{ij}^{(\ell)}} + \sqrt{1 - b_{ij}^{(\ell)}}}, & \eta_\ell &= \frac{b_{ij}^{(\ell)}}{\left(1 + \sqrt{1 + b_{ij}^{(\ell)}}\right)\left(1 + \sqrt{1 - b_{ij}^{(\ell)}}\right)}, \\ \tan(2\vartheta_\ell) &= \frac{2a_{ij}^{(\ell)} - (a_{ii}^{(\ell)} + a_{jj}^{(\ell)})b_{ij}^{(\ell)}}{(a_{jj}^{(\ell)} - a_{ii}^{(\ell)})\sqrt{1 - (b_{ij}^{(\ell)})^2}}, & -\frac{\pi}{4} &< \vartheta_\ell \le \frac{\pi}{4}. \end{aligned} \tag{2.8}$$

If $a_{ij}^{(\ell)} = b_{ij}^{(\ell)} = 0$, we set $\vartheta_\ell = 0$. If $a_{ii}^{(\ell)} = a_{jj}^{(\ell)}$, and $2a_{ij}^{(\ell)} = (a_{ii}^{(\ell)} + a_{jj}^{(\ell)})b_{ij}^{(\ell)}$, then the matrices $\widehat{A}^{(\ell)}$ and $\widehat{B}^{(\ell)}$ are proportional. Hence, we set $\vartheta_\ell = \frac{\pi}{4}$, unless $a_{ij}^{(\ell)} = b_{ij}^{(\ell)} = 0$.

To justify the notation, we prove that the quantities defined in (2.8) are indeed trigonometric functions of certain angles. Note that ϑ_ℓ in the specified range is always uniquely defined by $\tan(2\vartheta_\ell)$.

Proposition 2.1. *The quantities* $\cos\varphi_\ell$, $\sin\varphi_\ell$, $\cos\psi_\ell$, *and* $\sin\psi_\ell$ *represent the stated trigonometric functions of certain angles* φ_ℓ, *and* ψ_ℓ, *respectively.*

Proof. It is sufficient to prove that $\cos^2\varphi_\ell + \sin^2\varphi_\ell = 1$, and $\cos^2\psi_\ell + \sin^2\psi_\ell = 1$. From the first two lines of (2.8) it follows that

$$\cos^2\varphi_\ell + \sin^2\varphi_\ell = \cos^2\psi_\ell + \sin^2\psi_\ell = (1 - \xi_\ell\eta_\ell)^2 + \xi_\ell^2 = 1 + \xi_\ell(-2\eta_\ell + \xi_\ell\eta_\ell^2 + \xi_\ell).$$

To prove the claim, it is sufficient to show that

$$\xi_\ell(-2\eta_\ell + \xi_\ell\eta_\ell^2 + \xi_\ell) = 0.$$

If $\xi_\ell = 0$, the conclusion is obvious. Otherwise, the result is obtained by substitution of ξ_ℓ and η_ℓ from (2.8) as the functions of $b_{ij}^{(\ell)}$, followed by simplification of this expression. □

Hari in his Ph.D. thesis proved [14, Proposition 2.4] that

$$\min\{\cos\varphi_\ell, \cos\psi_\ell\} > 0,$$

and

$$-1 < \tan\varphi_\ell \tan\psi_\ell \leq 1,$$

with $\tan\varphi_\ell \tan\psi_\ell = 1$, if and only if $\vartheta_\ell = \frac{\pi}{4}$. Hence, $\widehat{Z}_\ell$ is nonsingular. More precisely,

$$\begin{aligned}\det(\widehat{Z}_\ell) &= \frac{1}{1-(b_{ij}^{(\ell)})^2}(\cos\varphi_\ell\cos\psi_\ell + \sin\varphi_\ell\sin\psi_\ell) = \frac{1}{1-(b_{ij}^{(\ell)})^2}(1-2\xi_\ell^2)\\ &= \frac{1}{1-(b_{ij}^{(\ell)})^2}\left(1 - \frac{(b_{ij}^{(\ell)})^2}{1+\sqrt{1-(b_{ij}^{(\ell)})^2}}\right) = \frac{1}{\sqrt{1-(b_{ij}^{(\ell)})^2}}.\end{aligned}$$

The transformations applied on B are congruences, and consequently, they preserve the inertia of B. Therefore, all matrices $B^{(\ell)}$ are positive definite. Since their diagonal elements are ones, the absolute value of all the other elements is smaller than one, i.e., the elements of $B^{(\ell)}$ cannot overflow. For the elements of $A^{(\ell)}$, the situation is slightly more complicated.

Proposition 2.2. *The elements of the matrix $A^{(\ell+1)} = Z_\ell^T A^{(\ell)} Z_\ell$ are bounded in terms of the elements of $A^{(\ell)}$, and the pivot element $b_{ij}^{(\ell)}$ of $B^{(\ell)}$. They are equal to the elements of $A^{(\ell)}$, except in the columns (and, due to symmetry, rows) with indices i and j, where $a_{ij}^{(\ell+1)} = 0$, and*

$$\max\left\{|a_{ii}^{(\ell+1)}|, |a_{jj}^{(\ell+1)}|\right\} \leq \frac{4\max\left\{|a_{ii}^{(\ell)}|, |a_{ij}^{(\ell)}|, |a_{jj}^{(\ell)}|\right\}}{1-(b_{ij}^{(\ell)})^2}, \quad \max\left\{|a_{pi}^{(\ell+1)}|, |a_{pj}^{(\ell+1)}|\right\} \leq \frac{2\max\left\{|a_{pi}^{(\ell)}|, |a_{pj}^{(\ell)}|\right\}}{\sqrt{1-(b_{ij}^{(\ell)})^2}}, \quad p = 1, \ldots, k.$$

Moreover, if A is positive definite, the bound for the diagonal elements of $A^{(\ell+1)}$ can be written as

$$\max\left\{a_{ii}^{(\ell+1)}, a_{jj}^{(\ell+1)}\right\} \leq \frac{4\max\left\{a_{ii}^{(\ell)}, a_{jj}^{(\ell)}\right\}}{1-(b_{ij}^{(\ell)})^2}.$$

Proof. From the structure of Z_ℓ it follows that the elements of $A^{(\ell+1)}$ are equal to the elements of $A^{(\ell)}$, except in the pivot rows and columns, i.e., the rows and columns with indices i and j. We have

$$\begin{aligned}
&a_{pi}^{(\ell+1)} = a_{ip}^{(\ell+1)} = \frac{a_{pi}^{(\ell)}\cos\varphi_\ell - a_{pj}^{(\ell)}\sin\psi_\ell}{\sqrt{1-(b_{ij}^{(\ell)})^2}}, \quad p \neq i, j, \qquad a_{pj}^{(\ell+1)} = a_{jp}^{(\ell+1)} = \frac{a_{pi}^{(\ell)}\sin\varphi_\ell + a_{pj}^{(\ell)}\cos\psi_\ell}{\sqrt{1-(b_{ij}^{(\ell)})^2}}, \quad p \neq i, j,\\
&a_{ii}^{(\ell+1)} = \frac{\cos^2\varphi_\ell a_{ii}^{(\ell)} - 2\cos\varphi_\ell\sin\psi_\ell a_{ij}^{(\ell)} + \sin^2\psi_\ell a_{jj}^{(\ell)}}{1-(b_{ij}^{(\ell)})^2}, \quad a_{jj}^{(\ell+1)} = \frac{\sin^2\varphi_\ell a_{ii}^{(\ell)} + 2\sin\varphi_\ell\cos\psi_\ell a_{ij}^{(\ell)} + \cos^2\psi_\ell a_{jj}^{(\ell)}}{1-(b_{ij}^{(\ell)})^2},
\end{aligned} \tag{2.9}$$

and $a_{ij}^{(\ell+1)} = a_{ji}^{(\ell+1)} = 0$. From (2.9), for $s = i, j$ and $p \neq s$, it follows

$$|a_{ps}^{(\ell+1)}| = |a_{sp}^{(\ell+1)}| \leq \frac{|a_{pi}^{(\ell)}| + |a_{pj}^{(\ell)}|}{\sqrt{1-(b_{ij}^{(\ell)})^2}} \leq \frac{2\max\left\{|a_{pi}^{(\ell)}|, |a_{pj}^{(\ell)}|\right\}}{\sqrt{1-(b_{ij}^{(\ell)})^2}},$$

and

$$|a_{ss}^{(\ell+1)}| \leq \frac{|a_{ii}^{(\ell)}| + 2|a_{ij}^{(\ell)}| + |a_{jj}^{(\ell)}|}{1-(b_{ij}^{(\ell)})^2} \leq \frac{4\max\left\{|a_{ii}^{(\ell)}|, |a_{ij}^{(\ell)}|, |a_{jj}^{(\ell)}|\right\}}{1-(b_{ij}^{(\ell)})^2}.$$

The bound for positive definite matrices A follows from the fact that all matrices $A^{(\ell)}$ are positive definite, and all principal minors of order 2 in $A^{(\ell)}$ are positive, i.e., $|a_{ij}^{(\ell)}| \leq \sqrt{a_{ii}^{(\ell)} a_{jj}^{(\ell)}} \leq \max\{a_{ii}^{(\ell)}, a_{jj}^{(\ell)}\}$. □

Note that if $B = I$, then $\xi_\ell = \eta_\ell = 0$ for all ℓ in (2.8), and ϑ_ℓ is the angle from the ordinary Jacobi method for a single symmetric matrix A. In this case, $\widehat{Z}_\ell$ in (2.7) is the ordinary plane rotation for the angle ϑ_ℓ. A similar situation occurs near the end of the diagonalization process, when B is close to I, and the matrices $\widehat{Z}_\ell$ defined by (2.7) tend to the ordinary rotations. This reasoning is justified by the following convergence theorem, proved in [14].

Theorem 2.3 (Hari)**.** *If B is positive definite, the row- and the column-cyclic Hari–Zimmermann method is globally convergent. There exists a permutation π of the generalized eigenvalues $\lambda_1 \le \cdots \le \lambda_k$ of the pair (A, B), such that*

$$\lim_{\ell\to\infty} A^{(\ell)} = \operatorname{diag}(\lambda_{\pi(k)}, \ldots, \lambda_{\pi(1)}), \qquad \lim_{\ell\to\infty} B^{(\ell)} = I.$$

Since the elements of $A^{(\ell+1)}$ are bounded, and the algorithm is convergent, we expect that the elements of $A^{(\ell)}$, for all ℓ, will not overflow, provided that the initial matrix B is not severely ill-conditioned.

3. Jacobi–type algorithms for GSVD

3.1. The implicit Hari–Zimmermann algorithm

In the GSVD problem, two matrices $F_0 \in \mathbb{R}^{m\times n}$ and $G_0 \in \mathbb{R}^{p\times n}$ are given. If G_0 is not of full column rank, then after the preprocessing step from [3], we obtain a matrix pair of two square matrices (F, G), with G of full rank k.

For such F and G, since G^TG is a positive definite matrix, the pair (F^TF, G^TG) in the corresponding GEP is symmetric and definite. There exist many nonsingular matrices Z that simultaneously diagonalize the pair (F^TF, G^TG) by congruences [28, Theorem 15.3.2, p. 344],

$$Z^TF^TFZ = \Lambda_F, \quad Z^TG^TGZ = \Lambda_G, \tag{3.1}$$

where Λ_F and Λ_G are diagonal matrices such that $(\Lambda_F)_{ii} \ge 0$ and $(\Lambda_G)_{ii} > 0$, for $i = 1, \ldots, k$. Since $Z^TG^TGZ\Lambda_G^{-1} = I$, the problem (3.1) can be rewritten as

$$(F^TF)Z = Z^{-T}\cdot I\cdot\Lambda_F = Z^{-T}(Z^TG^TGZ\Lambda_G^{-1})\Lambda_F = (G^TG)Z(\Lambda_G^{-1}\Lambda_F),$$

and the eigenvalues of the pair (F^TF, G^TG) are the diagonal elements of the matrix $\Lambda_G^{-1}\Lambda_F$.

First note that the congruence transformations by Z in (3.1) can be written as one-sided transformations from the right-hand side on F and G, with transformation parameters computed from F^TF and G^TG. Moreover, the final matrices Λ_F and Λ_G are diagonal, so the columns of FZ and GZ are orthogonal (not orthonormal), with the column norms equal to the square roots of the diagonal elements of Λ_F and Λ_G, respectively. Hence, (3.1) can be written as

$$FZ = U\Lambda_F^{1/2}, \quad GZ = V\Lambda_G^{1/2}, \tag{3.2}$$

where U and V are orthogonal matrices, and $(\,)^{1/2}$ denotes the principal square root of a matrix (see [18, Section 1.7]). Finally, if $\Lambda_F + \Lambda_G \ne I$, then define the diagonal scaling

$$S = (\Lambda_F + \Lambda_G)^{1/2}. \tag{3.3}$$

By setting

$$X := SZ^{-1}, \qquad \Sigma_F := \Lambda_F^{1/2}S^{-1}, \qquad \Sigma_G := \Lambda_G^{1/2}S^{-1}, \tag{3.4}$$

the relation (3.2) becomes (1.1), i.e., the GSVD of the pair (F, G). If only the generalized singular values are sought for, this rescaling is not necessary, and the generalized singular values are simply $\sigma_i = (\Lambda_G^{-1/2}\Lambda_F^{1/2})_{ii}$, for $i = 1, \ldots, k$.

Now it is easy to establish a connection between the two-sided GEP algorithm and the one-sided GSVD algorithm. Suppose that the algorithm works in sweeps. Each sweep annihilates all pivot pairs (i, j), for $1 \le i < j \le k$, only once, in some prescribed cyclic order (for example, row- or column-cyclic). To determine the annihilation parameters from (2.7)–(2.8), the elements of the matrices $\widehat{A}^{(\ell)}$ and $\widehat{B}^{(\ell)}$ from (2.4) should be computed. If the pivot columns of $F^{(\ell)}$ and $G^{(\ell)}$ are denoted by $f_i^{(\ell)}$, $f_j^{(\ell)}$, $g_i^{(\ell)}$, and $g_j^{(\ell)}$, then the elements of $\widehat{A}^{(\ell)}$ and $\widehat{B}^{(\ell)}$ are the inner products of pivot columns of $F^{(\ell)}$ and $G^{(\ell)}$, respectively,

$$\begin{aligned} a_{ii}^{(\ell)} &= (f_i^{(\ell)})^T f_i^{(\ell)}, \qquad a_{ij}^{(\ell)} = (f_i^{(\ell)})^T f_j^{(\ell)}, \qquad a_{jj}^{(\ell)} = (f_j^{(\ell)})^T f_j^{(\ell)}, \\ b_{ii}^{(\ell)} &= (g_i^{(\ell)})^T g_i^{(\ell)}, \qquad b_{ij}^{(\ell)} = (g_i^{(\ell)})^T g_j^{(\ell)}, \qquad b_{jj}^{(\ell)} = (g_j^{(\ell)})^T g_j^{(\ell)}. \end{aligned} \tag{3.5}$$

The rest of the one-sided or the implicit algorithm consists of computing the elements of the transformation matrix from (2.8), and updating of the corresponding columns in $F^{(\ell)}$ and $G^{(\ell)}$, as described in Algorithm 3.1.

Algorithm 3.1: Implicit cyclic Hari–Zimmermann algorithm for the GSVD.

Description: Algorithm `HZ_Orthog` works in sweeps, where *max_sw* is the maximal number of allowed sweeps. The step index ℓ is omitted, and the pivot submatrices are denoted by hat. After completion of the algorithm, `Finalize` computes all the matrices in the GSVD of the pair (F, G). The matrices U and V are stored in F and G, respectively. If *acc* is set to true in `HZ_GSVD`, the transformation matrix $Z = X^{-1}S$ is accumulated, and X is obtained by solving the linear system $Z^T X^T = \mathrm{diag}(S_{11}, \ldots, S_{kk})$. The maximal value of 50 sweeps in `HZ_GSVD` is set provisionally, because numerical testing shows that `HZ_Orthog` terminates much sooner than that. No transformations in this sweep means that all the transformations have been computed as the identity matrices.

```
HZ_Orthog(inout :: F, G, Z,  in :: k, acc, max_sw);
begin
    if acc then Z = I;
    it = 0;
    repeat   // sweep loop
        it = it + 1;
        for all pairs (i, j), 1 ≤ i < j ≤ k do
            compute Â and B̂ from (3.5);
            compute the elements of Ẑ by using (2.8);
            // transform F and G
            [f_i, f_j] = [f_i, f_j] · Ẑ;
            [g_i, g_j] = [g_i, g_j] · Ẑ;
            // if needed, accumulate Z
            if acc then [z_i, z_j] = [z_i, z_j] · Ẑ;
        end for
    until (no transf. in this sweep) or (it ≥ max_sw);
end
```

```
Finalize(inout :: F, G,  in :: Z, k, acc,
         out :: Σ_F, Σ_G, X);
begin
    for i = 1 to k do
        S_ii = sqrt(||f_i||_2^2 + ||g_i||_2^2);
        (Σ_F)_ii = ||f_i||_2 / S_ii;      (Σ_G)_ii = ||g_i||_2 / S_ii;
        f_i = f_i / ||f_i||_2;      g_i = g_i / ||g_i||_2;
    end for
    if acc then
        solve the linear system Z^T X^T = S to obtain X;
    end if
end

HZ_GSVD(inout :: F, G,  in :: k,  out :: Z, Σ_F, Σ_G, X);
begin
    set acc;      max_sw = 50;
    call HZ_Orthog(F, G, Z, k, acc, max_sw);
    call Finalize(F, G, Z, k, acc, Σ_F, Σ_G, X);
end
```

In principle, the GSVD of a given pair (F, G) can be computed by solving the GEP for the pair $(F^T F, G^T G)$. The first step is to compute the products $A = F^T F$ and $B = G^T G$. Unfortunately, the conditions of A and B are squares of the original conditions of F and G. For example, since A is symmetric and positive definite,

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} = \frac{\sigma^2_{\max}(F)}{\sigma^2_{\min}(F)} = \kappa^2(F),$$

and, similarly, for B and G. As a consequence, the computed generalized eigenvalues may suffer a significant loss of accuracy with respect to the computed solution of the original GSVD problem. In addition, the GEP algorithm uses two-sided transformations, which are not cache-aware and are hard to parallelize. The one-sided transformations do not suffer from any of these drawbacks, so the GSVD algorithm turns out to be much faster and more stable.

From the practical point of view, it is better to solve the GEP by reducing it to the GSVD, than the other way around. The original two-sided Hari–Zimmermann algorithm for the GEP can be replaced by a two-step algorithm. When A is positive definite, the first step is the Cholesky factorization of both matrices A and B, resulting in two triangular matrices F and G, to which we apply the implicit Hari–Zimmermann algorithm. Since the iterative part of the algorithm is one-sided, we perform only the transformations of columns, which are much faster than the two-sided transformations in the original algorithm.

When A is symmetric and indefinite, and B is symmetric positive definite, an algorithm similar to Algorithm 3.1 can also be used as the second step in solving the GEP for the pair (A, B). In such an algorithm, the first step should be a slightly modified symmetric indefinite factorization [32] of A (preferably, with complete pivoting [5]), that produces the signs of the eigenvalues,

$$A = P^T F^T J F P,$$

where P is a permutation, F is block upper triangular with diagonal blocks of order 1 or 2, and J is a diagonal matrix with ± 1 on its diagonal. Then, G is computed from the Cholesky factorization of the matrix

$$PBP^T = G^T G.$$

After establishing the connection between one-sided and two-sided Hari–Zimmermann algorithms, the global convergence of the one-sided algorithm follows directly from Theorem 2.3.

Theorem 3.1. *If G is of full column rank, the row- and the column-cyclic implicit Hari–Zimmermann method for the GSVD of the pair (F, G) is globally convergent.*

Proof. Let $A = F^T F$ and $B = G^T G$. Since G is of full column rank, the matrix B is positive definite. According to Theorem 2.3, the two-sided Hari–Zimmermann algorithm for the GEP of the pair (A, B) is globally convergent. There exists a permutation π of the generalized eigenvalues $\lambda_1 \leq \cdots \leq \lambda_k$ of the pair (A, B), i.e., a permutation of the generalized singular values $\sigma_i = \sqrt{\lambda_i}$, $i = 1, \ldots, k$, of the pair (F, G), such that

$$\lim_{\ell \to \infty} A^{(\ell)} = \operatorname{diag}(\lambda_{\pi(k)}, \ldots, \lambda_{\pi(1)}) = \operatorname{diag}(\sigma^2_{\pi(k)}, \ldots, \sigma^2_{\pi(1)}), \qquad \lim_{\ell \to \infty} B^{(\ell)} = \operatorname{diag}(1, \ldots, 1) = I.$$

The relationship (3.5) between the elements of A and B, and the columns of F and G, shows that in the limit, when $\ell \to \infty$, the matrices $F^{(\ell)}$ tend to a matrix with orthogonal (but not orthonormal) columns, while the matrices $G^{(\ell)}$ tend to an orthogonal matrix. Finally, the postprocessing by the one-sided scaling S^{-1} in (3.3)–(3.4) has no influence on the convergence. □

3.2. Blocking in the one-sided algorithm

The main motivation for blocking of the algorithm is an efficient utilization of the cache memory. To this end, the columns of F and G are grouped into nb block-columns, denoted by F_i and G_i, respectively, with an almost equal number of columns $k_i \approx k/nb$ in each block

$$F = [F_1, F_2, \ldots, F_{nb}], \quad G = [G_1, G_2, \ldots, G_{nb}]. \tag{3.6}$$

A blocked algorithm works with the block-columns in a similar way as the non-blocked algorithm works with the individual columns. The notation remains the same, but now it is used at the level of blocks, instead of columns.

A chosen block-pivot strategy in block-step ℓ selects a pair of block-columns (i, j), where $1 \leq i < j \leq nb$, as the pivot pair. That generates the following pair of pivot submatrices

$$(\widehat{A}^{(\ell)}_{ij}, \widehat{B}^{(\ell)}_{ij}) := ((F^{(\ell)}_{ij})^T F^{(\ell)}_{ij}, (G^{(\ell)}_{ij})^T G^{(\ell)}_{ij}),$$

where

$$F^{(\ell)}_{ij} = [F^{(\ell)}_i, F^{(\ell)}_j], \quad G^{(\ell)}_{ij} = [G^{(\ell)}_i, G^{(\ell)}_j], \tag{3.7}$$

and

$$\widehat{A}^{(\ell)}_{ij} := (F^{(\ell)}_{ij})^T F^{(\ell)}_{ij} = \begin{bmatrix} (F^{(\ell)}_i)^T F^{(\ell)}_i & (F^{(\ell)}_i)^T F^{(\ell)}_j \\ (F^{(\ell)}_j)^T F^{(\ell)}_i & (F^{(\ell)}_j)^T F^{(\ell)}_j \end{bmatrix}, \qquad \widehat{B}^{(\ell)}_{ij} := (G^{(\ell)}_{ij})^T G^{(\ell)}_{ij} = \begin{bmatrix} (G^{(\ell)}_i)^T G^{(\ell)}_i & (G^{(\ell)}_i)^T G^{(\ell)}_j \\ (G^{(\ell)}_j)^T G^{(\ell)}_i & (G^{(\ell)}_j)^T G^{(\ell)}_j \end{bmatrix}. \tag{3.8}$$

The order of pivot matrices in (3.8), i.e., the number of columns in the so-called pivot blocks $F^{(\ell)}_{ij}$ and $G^{(\ell)}_{ij}$ in (3.7), is $k_i + k_j$. The blocked algorithm then performs a one-sided transformation of both pivot blocks to obtain a new pair of blocks

$$F^{(\ell+1)}_{ij} = F^{(\ell)}_{ij} \widehat{Z}^{(\ell)}_{ij}, \quad G^{(\ell+1)}_{ij} = G^{(\ell)}_{ij} \widehat{Z}^{(\ell)}_{ij}, \tag{3.9}$$

where $\widehat{Z}^{(\ell)}_{ij}$ is a nonsingular matrix of order $k_i + k_j$. This transformation matrix is a blocked analogue of the pointwise Hari–Zimmermann transformation $\widehat{Z}_\ell$ from (2.7). If these transformations have to be accumulated as in (2.2), the global transformation matrix $Z^{(\ell)}$ of order k is partitioned in the same way as the columns of F and G, and only the block-columns $Z^{(\ell)}_i$ and $Z^{(\ell)}_j$ need to be updated. By putting

$$Z^{(\ell)}_{ij} = [Z^{(\ell)}_i, Z^{(\ell)}_j],$$

the block-column update can be written as

$$Z^{(\ell+1)}_{ij} = Z^{(\ell)}_{ij} \cdot \widehat{Z}^{(\ell)}_{ij}. \tag{3.10}$$

The matrix $\widehat{Z}_{ij}^{(\ell)}$ in (3.9) is taken as a product, or a sequence of a certain number of pointwise transformations, all of which have the same form as in (2.7). Each pointwise transformation orthogonalizes a pair of pivot columns, and the whole sequence of such transformations in a blocked algorithm is chosen to accomplish a similar task on the block-level, i.e., to make all the columns in both pivot blocks more orthogonal than before. Since $k_i + k_j$ is, generally, greater than 2, the non-blocked (pointwise) algorithm can now be mimicked in two different ways—we can "fully" orthogonalize both pivot blocks, or apply just a single sweep of pointwise transformations.

1. If the block-sizes are chosen to be sufficiently small, most of the required data resides in cache, and the pivot pair $(\widehat{A}_{ij}^{(\ell)}, \widehat{B}_{ij}^{(\ell)})$ can be diagonalized efficiently by the non-blocked two-sided Hari–Zimmermann algorithm. In the one-sided terminology, the columns of pivot blocks $F_{ij}^{(\ell)}$ and $G_{ij}^{(\ell)}$ are orthogonalized. This approach is called the *full block* algorithm.
2. The offdiagonal elements of the pair $(\widehat{A}_{ij}^{(\ell)}, \widehat{B}_{ij}^{(\ell)})$ can be annihilated exactly once, i.e., we apply only one sweep of the transformations to the right-hand side of $F_{ij}^{(\ell)}$ and $G_{ij}^{(\ell)}$. If the block-sizes are chosen to be moderate, this, the so-called *block-oriented* algorithm, is a powerful alternative to the full block algorithm.

Note that the one-sided transformations of block-columns can be applied from the right-hand side on $(F_{ij}^{(\ell)}, G_{ij}^{(\ell)})$ in (3.7), or from the left-hand side on $((F_{ij}^{(\ell)})^T, (G_{ij}^{(\ell)})^T)$. The choice of the side depends on the programming language. For example, Fortran stores matrices by columns, and the right-handed transformations are suitable, while C keeps matrices by rows, and the left-handed transformations are more appropriate.

Both blocked versions of the Hari–Zimmermann algorithm are recursive in nature, as they use the pointwise algorithm at the bottom (or the inner) level, to compute the transformations at the block-level. It is a well-known fact that the inner one-sided Jacobi-type algorithms are most efficient when applied to square matrices, because the columns that appear in inner products (3.5) and column updates (Algorithm 3.1) are as short as possible. Because the block-columns $F_{ij}^{(\ell)}$ and $G_{ij}^{(\ell)}$ are "tall-and-skinny" matrices, suitable square matrices $R_F^{(\ell)}$ and $R_G^{(\ell)}$ (pivot indices ij are omitted to keep the notation readable) can be computed either

1. from (3.7), by the QR factorization of $F_{ij}^{(\ell)}$ and $G_{ij}^{(\ell)}$, or
2. from (3.8), by the Cholesky factorization of $\widehat{A}_{ij}^{(\ell)}$ and $\widehat{B}_{ij}^{(\ell)}$.

Since the original $F_{ij}^{(\ell)}$ and $G_{ij}^{(\ell)}$ are needed in the algorithm for the block-column updates (3.9), they should be copied and saved before the QR factorization, because the QR factorization of a given matrix is performed in-place, i.e., it destroys the original matrix. In the second alternative, this extra copying of tall blocks is not needed. An additional storage is required only for the small square pivot submatrices to store the matrix multiply results, and the Cholesky factorization is performed in-place on these results. Besides, the "multiply + in-place Cholesky" approach turns out to be much faster in practice, so we use it in all blocked algorithms to compute $R_F^{(\ell)}$ and $R_G^{(\ell)}$.

Regardless of the choice, the use of pivoting is desirable in both approaches, to improve the numerical stability and the speed of convergence of the blocked algorithm. Since the pivoting must be the same for both matrices, a direct application of the Lapack routines for the pivoted QR or the pivoted Cholesky factorization of each matrix, as in [31], is not possible. Instead of using pivoting in the factorization algorithms, which becomes quite complicated when multiple levels of blocking are present, the pivoting is incorporated directly into the (pointwise) orthogonalization at the bottom level of the algorithm, i.e., into the inner loop of `HZ_Orthog` from Algorithm 3.1.

One way to do this is similarly as de Rijk [6] incorporated pivoting in the ordinary Jacobi SVD computation. In that algorithm, each sweep orthogonalizes k columns of a given matrix R in a row-cyclic manner. To ensure that all pairs of the original columns are indeed transformed, a sweep is implemented like the selection sort algorithm over the columns of the current working matrix (initially R), in the decreasing order of their norms (see Algorithm 3.2 (left), where the sweep with pivoting is implemented in-place—on the working matrix denoted by R). The sweep is divided into $k - 1$ subsweeps. In each subsweep, the first pivot column is the one with the largest norm among the remaining columns, and then all the remaining pairs are transformed, which changes the norms of the transformed columns. The number of necessary column interchanges in each sweep is at most $k - 1$.

Explicit interchanges of columns can cause a significant slowdown of the inner orthogonalization algorithm, so they should be avoided, if possible. Such a trick is used by Novaković [24] for the Jacobi SVD on graphics processing units (see Algorithm 3.2 (right), again written in-place, on the working matrix R). Since this strategy is strictly local, i.e., it compares only a pair of columns to be transformed, it is suitable for parallel algorithms—any prescribed ordering of pairs (i, j) in a sweep can be used, not just a cyclic one.

Algorithm 3.2: Pivoting strategies for the SVD: sequential de Rijk [6] (left), and local Novaković [24] (right)

```
for i = 1 to k − 1 do
    find column r_max such that ||r_max||_2 = max_{j=i,...,k} ||r_j||_2;
    interchange columns r_i and r_max;
    for j = i + 1 to k do
        orthogonalize columns r_i and r_j;
    end for
end for
```

```
for each pair (i, j) in a sweep do
    compute 2 × 2 rotation that orthogonalizes r_i, r_j;
    compute ||r'_i||_2 and ||r'_j||_2 after the transformation;
    if ||r'_i||_2 < ||r'_j||_2 then
        swap columns of the computed 2 × 2 rotation;
    end if
    apply the rotation to r_i and r_j;
end for
```

More importantly, in the local pivoting strategy there are no column interchanges before the transformation, and the actual ordering is achieved by permuting the pointwise transformation itself, when required. In contrast to de Rijk's sequential strategy, which begins with a "sorted" pair of columns (with respect to their norms), the local reordering strategy ends with a "sorted" pair of columns after each pointwise transformation.

In the GSVD algorithm, the same pivoting has to be applied on two matrices $R_F^{(\ell)}$ and $R_G^{(\ell)}$, instead of only to one matrix R in the ordinary SVD algorithm. To describe the modified reordering strategy, that is suitable for the GSVD computation, it is sufficient to consider the state before and after one pointwise transformation. For simplicity, we will omit the pointwise transformation index in the discussion below, and adopt the following convention: all quantities without a prime refer to the state before the transformation, while all primed quantities refer to the state after the transformation.

The GSVD pivoting is designed in such a fashion that the ratio between the norms of the pivot columns from R_F and R_G is decreasing after the transformation, i.e.,

$$\frac{\|(R'_F)_i\|_2}{\|(R'_G)_i\|_2} \geq \frac{\|(R'_F)_j\|_2}{\|(R'_G)_j\|_2}, \quad i < j.$$

Additionally, in the one-sided Hari–Zimmermann algorithm, the norms of all computed columns in R'_G are equal to 1, so it is sufficient to compare the norms of the computed pivot columns in R'_F.

To accomplish the desired ordering, the sweep phase of Algorithm 3.1 should be modified so that the pivot submatrices $[f_i, f_j]$, $[g_i, g_j]$, and $[z_i, z_j]$ are multiplied either by $\widehat{Z}$ or by $\widehat{Z}P$, where

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

depending on the sizes of $\|f'_s\|_2 = \|(R'_F)_s\|_2$, for $s = i, j$. These two norms can be computed very quickly (with a reasonable accuracy) before the actual transformation. The elements of pivot submatrices from (2.4) are already at our disposal, as they are needed in the computation of $\tan(2\vartheta)$ in (2.8). Then it is easy to compare the scaled squares of norms

$$a''_{ss} := (1 - b_{ij}^2) a'_{ss} = (1 - b_{ij}^2)\, \|(R'_F)_s\|_2^2, \quad s = i, j,$$

where a''_{ii} and a''_{jj} are computed as

$$\begin{aligned} a''_{ii} &= \cos^2\varphi\, a_{ii} - 2\cos\varphi \sin\psi\, a_{ij} + \sin^2\psi\, a_{jj}, \\ a''_{jj} &= \sin^2\varphi\, a_{ii} + 2\sin\varphi \cos\psi\, a_{ij} + \cos^2\psi\, a_{jj}. \end{aligned}$$

In the later stages of the orthogonalization, it can be expected that the columns are already sorted in such a way, so that the permutations do not occur any more, and the algorithm will produce the generalized singular values in the decreasing order.

Subsequently, we assume that `HZ_Orthog` in Algorithm 3.1 incorporates the local reordering of pivot columns as in Algorithm 3.2 (right), combined with any chosen pointwise pivoting strategy. Such a modification brings an additional speedup of up to 20%.

In any blocked version of the algorithm, the matrix $\widehat{Z}_{ij}^{(\ell)}$ that transforms the columns of $R_F^{(\ell)}$ and $R_G^{(\ell)}$ can be computed in several different ways (see [16, 17]). Numerical tests show that the explicit accumulation of all used transformations is the best option regarding the accuracy, while the solution of one of the linear systems

$$R_F^{(\ell)}\widehat{Z}_{ij}^{(\ell)} = R_F^{(\ell+1)}, \qquad R_G^{(\ell)}\widehat{Z}_{ij}^{(\ell)} = R_G^{(\ell+1)}, \tag{3.11}$$

after all transformations, is the best option regarding the speed (about 20% faster for sequential execution of blocked algorithms), with a negligible loss of accuracy (see [10] for details). The first system in (3.11) can be singular, while the second one is certainly not, since G is of full column rank.

When multiple levels of blocking are present, the decision which option is better may vary from level to level. For instance, if the blocks are spread over a distributed memory, then local matrix multiplication in each process (executed in parallel), is much faster than the parallel solution of a distributed linear system.

Both blocked versions of the one-sided Hari–Zimmermann algorithm are given in Algorithm 3.3.

Algorithm 3.3: Implicit blocked Hari–Zimmermann algorithm for the GSVD.

Description: Algorithm HZ_BOrthog_Lv$_n$ transforms F and G by one of the blocked algorithms. The step index ℓ is omitted. To accommodate multiple levels of blocking, the algorithm is recursive in nature—each pivot block is transformed by the calling essentially the same algorithm HZ_BOrthog_Lv$_{n-1}$ at the next level of blocking. At the bottom level, HZ_BOrthog_Lv$_0$ is the pointwise algorithm HZ_Orthog.

HZ_BOrthog_Lv$_n$(*inout :: F, G, Z, in :: k, acc$_n$, max_sw$_n$*);

begin

partition matrices F, G, and (if needed) Z, as in (3.6);

set acc_{n-1} to true, if $\widehat{Z}_{ij}$ will be accumulated, or to false, if $\widehat{Z}_{ij}$ will be obtained by solving the linear system;

$it = 0$;

if algorithm is *full block* **then**

$max\_sw_{n-1} = 50$;

else if algorithm is *block-oriented* **then**

$max\_sw_{n-1} = 1$;

end if

repeat // sweep loop

$it = it + 1$;

for each pair (i, j) in a block-sweep, according to some block-pivot strategy **(parallel) do**

// block-transformation: in parallel implementation, a thread or a process job

compute the factors R_F and R_G of the submatrices $\widehat{A}_{ij}$ and $\widehat{B}_{ij}$, respectively;

if acc_{n-1} **then**

$\widehat{Z}_{ij} = I$;

else

save R_F to T_F, or R_G to T_G;

end if

call HZ_BOrthog_Lv$_{n-1}$(R_F, R_G, $\widehat{Z}_{ij}$, $k_i + k_j$, acc_{n-1}, max_sw_{n-1});

if not acc_{n-1} **then** solve the linear system $T_F\widehat{Z}_{ij} = R_F$, or $T_G\widehat{Z}_{ij} = R_G$;

// transform the pivot blocks of F and G

$F_{ij} = F_{ij} \cdot \widehat{Z}_{ij}$; $\quad G_{ij} = G_{ij} \cdot \widehat{Z}_{ij}$;

// if needed, accumulate Z

if acc_n **then** $Z_{ij} = Z_{ij} \cdot \widehat{Z}_{ij}$;

end for

until (no transf. in this sweep) **or** ($it \geq max\_sw_n$);

end

The driver routine HZ_GSVD is the same as in Algorithm 3.1, except that it now calls the topmost blocked algorithm HZ_BOrthog_Lv$_n$, instead of the pointwise algorithm HZ_Orthog.

The general blocked algorithm HZ_BOrthog_Lv$_n$, at a certain level n, can be implemented either sequentially or in parallel, depending on the choice of the block-pivot strategy.

The hierarchy of blocking levels in Algorithm 3.3 corresponds to the memory hierarchy in modern computer architectures. In that respect, level 1 (the bottom level of true blocking) is, usually, the sequential blocked algorithm, aimed at exploiting the local cache hierarchy of a single processing unit. The next two levels of blocking correspond to the shared and distributed memory parallelism, respectively. The choice of the topmost level depends on the size of the pair and the available hardware.

In a multi-level blocked algorithm, only the topmost level "sees" the block-columns of the original square matrices F and G, while the lower levels "see" only the shortened square matrices $R_F^{(\ell)}$ and $R_G^{(\ell)}$, resulting from some pivot blocks at the level above.

Note that the bottom pointwise level HZ_BOrthog_Lv$_0$ = HZ_Orthog can be implemented in a similar fashion as all the blocked levels. However, for pivot blocks with only two columns, it is much faster to implement the transformations directly (as in Algorithm 3.1), without a reduction to square matrices of order 2.

The most time-consuming part of the blocked algorithm is the update of both pivot blocks (3.9), and the update of the transformation matrix (3.10), if it is accumulated. All updates are done by the BLAS 3 routine xGEMM. Since the original matrices cannot be overwritten by xGEMM, we need a temporary $k \times 2k_0$ matrix, where

$$k_0 = \max_{i=1,\ldots,nb} k_i. \tag{3.12}$$

This is sufficient for the sequential blocked algorithms (level 1). If these two or three updates are to be performed in parallel (at higher levels), then the required temporary storage is $k \times 2k_0$ per update.

At parallel blocked levels of the algorithm, the number of blocks nb is determined by the number of cores or processes available (see Section 4), which determines the value of k_0 in (3.12).

For the sequential blocked algorithms, the optimal value of k_0 very much depends on the cache hierarchy, but also on the fine implementation details of the algorithm itself. Therefore, the optimal k_0 in (3.12) can be determined only by extensive numerical testing. Generally, the performance of each algorithm is almost constant for a reasonably wide range of nearly optimal block-sizes. In the full block algorithm, this range is narrower and covers smaller values of k_0, than in the block-oriented algorithm. Moreover, the range width decreases as the matrix order k grows. For example, for moderately sized matrices of order 5000, the nearly optimal range is 16–40 for the full block, and 32–128 for the block-oriented algorithm. On the other hand, for smaller matrices of order 2000, the difference between the ranges of nearly optimal block-sizes is minimal (see Figure 3.1).

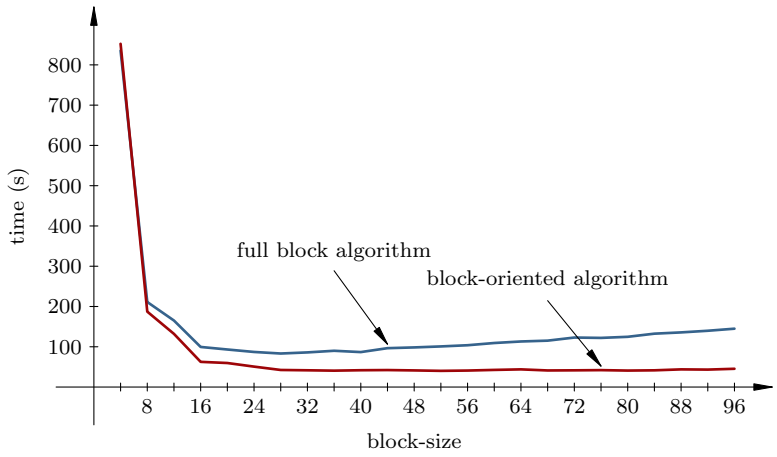

Figure 3.1: Effect of the chosen block-size k_0 for the sequential block-oriented and full block algorithms on a matrix of order 2000, with threaded MKL routines on 12 cores.

4. Parallel algorithms

It is well known that the one-sided Jacobi type algorithms are easy to parallelize, because independent (disjoint) pivot blocks can be transformed simultaneously. This is implemented by choosing one of the so-called parallel pivoting strategies at the appropriate level of blocking, or even at the pointwise level of the algorithm.

4.1. Shared memory algorithm

To maximize the efficiency on multicore computers, the blocked algorithms should be parallelized adequately. Our aim here is to exploit the shared memory parallelism—available on almost any modern computing architecture. To this end, we use the OpenMP interface in our Fortran routines.

The basic principle of a parallel blocked algorithm is to divide the columns of F and G into blocks, as in (3.6), such that $nb = 2p$, where p is the number of available cores. For simplicity, we assume that each core runs (at most) one computational OpenMP thread. In each step, every thread transforms two block-columns of F, and two block-columns of G, with the same indices for both pairs of block-columns. These block-columns are regarded as "pivots" in each thread (as in (3.7)), and the sequential algorithm can be used to transform them.

However, if we have, for example, 6 threads, and two matrices of order 12000, then each thread would work on two 12000×2000 pivot blocks $F_{ij}^{(\ell)}$ and $G_{ij}^{(\ell)}$, and they would be shortened to square matrices $R_F^{(\ell)}$ and $R_G^{(\ell)}$ of order 2000. Depending on the cache memory available for each thread, these blocks can be too large for an efficient execution of the pointwise Hari–Zimmermann algorithm `HZ_Orthog`. Therefore, the matrices $R_F^{(\ell)}$ and $R_G^{(\ell)}$ should be subdivided into a second level of blocking and transformed by the sequential blocked algorithm $\texttt{HZ\_BOrthog\_Lv}_1$. Finally, the transformation matrix is applied from the right-hand side on both pivot blocks. So, the topmost parallel shared memory algorithm is actually given by $\texttt{HZ\_BOrthog\_Lv}_2$.

The main difference between the sequential blocked algorithms and the parallel shared memory algorithms is the choice of pivot blocks. In the parallel case, each thread (among p of them) transforms a block assigned to it. By choosing an appropriate parallel pivot strategy, that maps disjoint pivot blocks to the computational threads in each parallel step, the block-transformation in Algorithm 3.3 is the task performed by each thread. If $max\_sw_1 = 1$, then $\texttt{HZ\_BOrthog\_Lv}_2$ is the block-oriented algorithm, otherwise, for sufficiently large max_sw_1, it is the full block algorithm.

For $\texttt{HZ\_BOrthog\_Lv}_1$ in each thread, again it is sufficient to use a nearly optimal block-size k_0 in (3.12), because the differences in performance are not essential for any block-size in the nearly optimal range. Figure 3.1 shows that for matrices of order 12000, and 6 available threads, the speeds of the block-oriented and the full block parallel algorithm are almost constant if k_0 is chosen in the common range 24–40 for both algorithms. On machines with different cache configurations, the conclusion about the flexible range of (nearly) optimal k_0 values remains the same, even though there may be a significantly different speed ratio between these two variants of the algorithm.

Up until now we have omitted the discussion of pivot strategies, i.e., the order in which cores choose the pivot pairs. The ideal parallel algorithm would simultaneously transform $nb = 2p$ pivot blocks by using some parallel block-strategy, while performing the transformation of columns inside the block by using some sequential strategy, usually, row- or column-wise.

Parallel strategies can be classified as either pointwise or blockwise. As of now, the convergence results mostly exist for the pointwise strategies, and they are based on proving their weak equivalence (see [15] for the definition) to the ordinary row- or column-cyclic strategies. Formal proofs of convergence for blockwise strategies have started to appear very recently [17], but only for the standard Jacobi algorithm that diagonalizes a single symmetric matrix.

The pointwise modulus strategy, described in [22], simultaneously annihilates the elements of A and B on each antidiagonal. This parallel strategy is known to be globally convergent.

Theorem 4.1. *If G is of full column rank, the implicit Hari–Zimmermann method for the pair (F, G), under the modulus strategy, is globally convergent.*

Proof. It is sufficient to prove that the modulus strategy is weakly equivalent to the row-cyclic strategy, and thus convergent. This proof can be found in [25, Appendix]. □

If the order k of matrices A and B is even, the pointwise modulus strategy is not optimal, in the sense that it annihilates the maximal possible number of elements $(k/2)$ only on one half of the antidiagonals, and one element

less $(k/2-1)$ on the other half of the antidiagonals. This nonoptimality is illustrated in Figure 4.1, left, where an additional element that can be annihilated is presented in a lighter hue. The strategy that annihilates these additional elements, as well, is called the modified modulus strategy.

The same principle is used for the corresponding block-strategy that operates on blocks, with a slight modification to include the diagonal blocks (see Figure 4.1, right), thus providing a pivot pair for all cores at every parallel step. A complete description of the block-layout per core in each parallel step, and all the necessary data transfer (when needed) is given in [31, Algorithm 3.1]. The `Send_Receive` part of Algorithm 3.1 from [31] is not needed for the shared memory algorithms, but will be used in the MPI version of the algorithm.

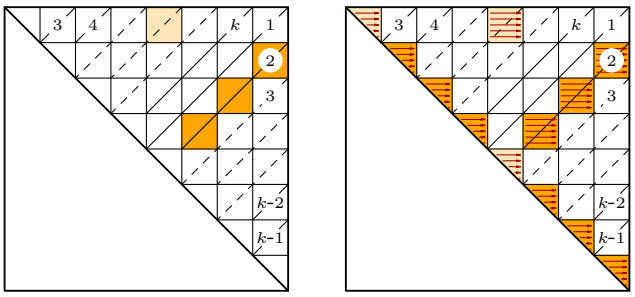

Figure 4.1: Left: modified pointwise modulus strategy on A (and B), right: modified block-modulus strategy on A (and B).

This block-strategy, combined with the row-cyclic inner strategy, is weakly equivalent to the row-cyclic block-strategy (with the same inner strategy). Therefore, we expect that the proof of convergence of these blocked algorithms is similar to the proofs for the ordinary block-Jacobi method [16, 17]. A mathematical technique suitable for the proofs of convergence of block-processes is developed in [15].

A halting criterion for the sequential algorithms is that the algorithm stops if no transformations have been performed in any block during the sweep, i.e., the off-diagonal elements in every pivot submatrix (2.4) are negligible, relative to the diagonal ones. This strong criterion can lead to unnecessary block-sweeps in the final phase of the parallel blocked algorithm, where only a few blocks are acted upon in the concluding block-sweeps, with only a few transformations per block. These transformations are caused by small rounding errors in factorizations and postmultiplications of the pivot blocks, but they do not alter the computed generalized singular values.

To prevent unnecessary transformations to incur extraneous sweeps, the shared memory and distributed memory parallel algorithms are terminated when a sweep (at the highest level of blocking) has had all its transformations with their cosines computed as 1. A similar heuristics is applied in [24] for the multi-level blocked SVD. Note that the corresponding sines might still be different from 0 in such a case. In Table 4.1 the total number of pointwise transformations in block-sweeps of the full block parallel algorithm is compared to the number of transformations with their cosines different from 1.

sweep number	number of all transformations	number of transformations with cosines different from 1
1	480628807	294997040
2	358157606	207835477
3	263922745	143114173
4	154679964	71401218
5	61192932	20656585
6	15891242	2317820
7	1139311	0

Table 4.1: Number of transformations per sweep for the full block parallel algorithm on a matrix of order 5000 with 4 cores.

The number of block-sweeps depends on the size of the pivot blocks $\widehat{A}_{ij}^{(\ell)}$ and $\widehat{B}_{ij}^{(\ell)}$ in (3.8), as well as the cuts between the generalized singular values induced by the block-partition (3.6). This causes a slight variation in the number of block-sweeps, when the block-size is changed.

4.2. Distributed memory algorithm

The parallel shared memory algorithms presented here can also be used as the building blocks in a hybrid MPI/OpenMP algorithm for the GSVD, that would target machines with the distributed, multi-level memory hierarchy and many CPU cores per machine.

The main principle is similar to the shared memory algorithm, i.e., the columns of F, G, and Z (if needed) are divided into blocks, as in (3.6), such that $nb = 2p_m$, where p_m is the number of available MPI processes. Therefore, each MPI process, as in all blocked variants of the algorithm so far (sequential and thread-parallel ones), requires memory storage for four block-columns per matrix (F, G, and Z), and the auxiliary storage for the block-factors $R_F^{(\ell)}$, $R_G^{(\ell)}$, and the transformation matrix $\widehat{Z}_{ij}^{(\ell)}$. Two block-columns hold the input data, and also serve as a receive buffer in the communication phase of the algorithm, and the other two block-columns store the postmultiplied (updated) data, and also serve as a send buffer in the communication phase.

The only conceptual difference between the distributed memory algorithm `HZ_BOrthog_Lv`$_3$ and the shared memory algorithm `HZ_BOrthog_Lv`$_2$ is that the updated block-columns have to be exchanged among the MPI processes after completing one step of the algorithm, according to the chosen parallel pivot strategy, while with the threaded algorithm the new pairs of block-columns are readily available in the shared memory. The exchange is achieved by a single `MPI_SENDRECV` operation per block-column on the Cartesian topology of the one-dimensional torus of processes, according to the modified modulus pivot strategy. The communication pattern is the same as in the three-level MPI-parallel Jacobi-type SVD computation, described in [30, 31], so we omit it here for brevity.

To eliminate the need for communication among the processes residing on the same computational node, there should be only one process allocated per node—either a physical one, or a NUMA domain. We have opted for the latter. Typically, in either case, an MPI process has several CPU cores available for execution, and we assume there is an even number p of them. In our setup, there is a single CPU with $p = 12$ cores per process available.

Note that the block-factors $R_F^{(\ell)}$ and $R_G^{(\ell)}$ may be computed concurrently, each in its own thread, under each MPI process. Such a thread may further employ the threaded BLAS interface with $p/2$ threads, if nested parallelism is available, to utilize fully all the cores available.

When the block-factors have been computed, the thread-parallel blocked algorithm `HZ_BOrthog_Lv`$_2$ with p threads is called—either the block-oriented or the full block one. In order to have a balanced execution time among the processes, the block-oriented variant is preferable. After that, the block-columns $F_{ij}^{(\ell)}$, $G_{ij}^{(\ell)}$, and $Z_{ij}^{(\ell)}$ are postmultiplied by $\widehat{Z}_{ij}^{(\ell)}$, using the threaded BLAS with p threads. Finally, a single block-column of each matrix is exchanged with the neighboring processes in the communication ring, creating an implicit synchronization point of the (otherwise concurrent) MPI processes, which concludes the actions of a single step of the distributed memory algorithm.

Table 4.2 shows the actual levels of blocking used in our implementations of the one-sided Hari–Zimmermann algorithm, as described in Algorithm 3.3. The table also specifies the pivoting strategy used at each level, and the corresponding value of acc_n that selects the method of computing the transformation matrix (accumulation or solution of a linear system). The choice acc_n = true at all levels is motivated by the fact that the Jacobi-type methods are known for their high accuracy of the computed results, so we opted for accuracy, rather than speed.

blocking level n	type of the algorithm	pivoting strategy	communication	transformations acc_n
3	distributed (MPI)	modified modulus	yes	true
2	shared memory parallel	modified modulus	no	true
1	sequential blocked	row-cyclic	–	true
0	pointwise	row-cyclic	–	true

Table 4.2: Levels of blocking for the one-sided Hari–Zimmermann algorithm.

5. Numerical testing

Numerical testing has been performed on a cluster with InfiniBand interconnect, and with the nodes consisting of two Intel Xeon E5-2697 v2 CPUs, running at 2.7 GHz. Each CPU is equipped with 12 cores. The cores share 30 MB of level-3 cache. Each core has 256 kB of private instruction/data level-2 cache, and 32 + 32 kB of private level-1 data and instruction caches. Each CPU belongs to a separate NUMA domain with 64 GB of RAM.

The software stack consists of Intel Composer XE 2015.1.133 with the Fortran compiler under 64-bit Linux, and the BLAS and LAPACK routines from Intel MKL library, in the sequential and thread-parallel modes.

Matrices were generated in the following manner: the diagonal matrices Σ_F and Σ_G were generated with uniformly distributed values from $[10^{-5}, 10^3]$, giving the generalized singular values from 10^{-5} to 10^4. Afterwards, these diagonal matrices were multiplied by random orthogonal matrices U and V, and a non-singular, reasonably well-conditioned matrix X, as in (1.1), resulting in relatively ill-conditioned matrices F_0 and G_0.

To alleviate possible round-off errors, all multiplications of matrices were carried out in Intel's `extended` floating-point type with 80 bits, and only the final matrices F_0 and G_0 were rounded to `double` precision.

Since Lapack's `DTGSJA` routine requires triangular starting matrices, F_0 and G_0 are preprocessed by Lapack's `DGGSVP` to obtain the starting matrices F and G for all algorithms. The time needed for this step is not measured.

First, we compare the non-blocked and blocked sequential versions of the Hari–Zimmermann algorithm with `DTGSJA` (see Table 5.1, left). In all cases, we used the threaded BLAS interface with 12 computational threads. Since the differences between the execution times of our routines and `DTGSJA` are already evident for matrices of order 5000, further comparisons with `DTGSJA` were not performed, because `DTGSJA` is too slow for larger matrices.

	with threaded MKL (12 cores)					with sequential MKL	
k	`DTGSJA`	`pointwise HZ`	`HZ-FB-32`	`HZ-BO-32`	k	`P-HZ-FB-32`	`P-HZ-BO-32`
500	16.16	3.17	4.36	2.03	500	1.41	0.88
1000	128.56	26.89	18.50	7.65	1000	4.78	2.02
1500	466.11	105.31	42.38	19.31	1500	14.57	5.99
2000	1092.39	273.48	86.01	41.60	2000	30.02	12.13
2500	2186.39	547.84	139.53	73.07	2500	53.13	22.34
3000	3726.76	1652.14	203.00	109.46	3000	86.78	36.08
3500	6062.03	2480.14	294.58	186.40	3500	129.37	55.20
4000	8976.99	3568.00	411.71	239.89	4000	180.32	86.36
4500	12805.27	4910.09	553.67	343.58	4500	249.92	119.74
5000	20110.39	6599.68	711.86	426.76	5000	320.39	159.59

Table 5.1: The running times of Lapack `DTGSJA` algorithm and various sequential and parallel implementations of the Hari–Zimmermann algorithm: pointwise (`pointwise HZ`), full block with block-size 32 (`HZ-FB-32`), and block-oriented with block-size 32 (`HZ-BO-32`). Shared memory parallel versions of the algorithms running on 12 cores are denoted by prefix `P-`.

The threaded BLAS routines, however, do not guarantee the maximum performance on the multi-core hardware. Therefore, we developed and tested the parallel versions of the algorithms, to see if further speedup is possible. The right-hand side of Table 5.1 shows the execution times for the parallel (shared memory) versions of blocked routines.

To demonstrate the value of explicit parallelization, we compare the fastest sequential routine `HZ-BO-32` with threaded MKL on 12 cores, and shared memory parallel routines on $p = 2, \ldots, 12$ cores (with p even). Figure 5.1 shows that the block-oriented parallel algorithm outperforms the fastest sequential routine when $p \geq 6$, and the full block algorithm does the same when $p \geq 10$.

It is interesting to observe that the fastest sequential routine is more than 15 times faster than the pointwise routine, both running in the same environment (with threaded BLAS on 12 cores). Likewise, the $2p$-core parallel algorithm is more than 2 times faster than the p-core algorithm. These superlinear speedups are caused by more efficient exploitation of the available memory hierarchy. The exact relationship between various versions of the algorithm is extremely machine-dependent, due to many intricacies of modern computer architectures, and thus it is very hard to predict without the actual testing.

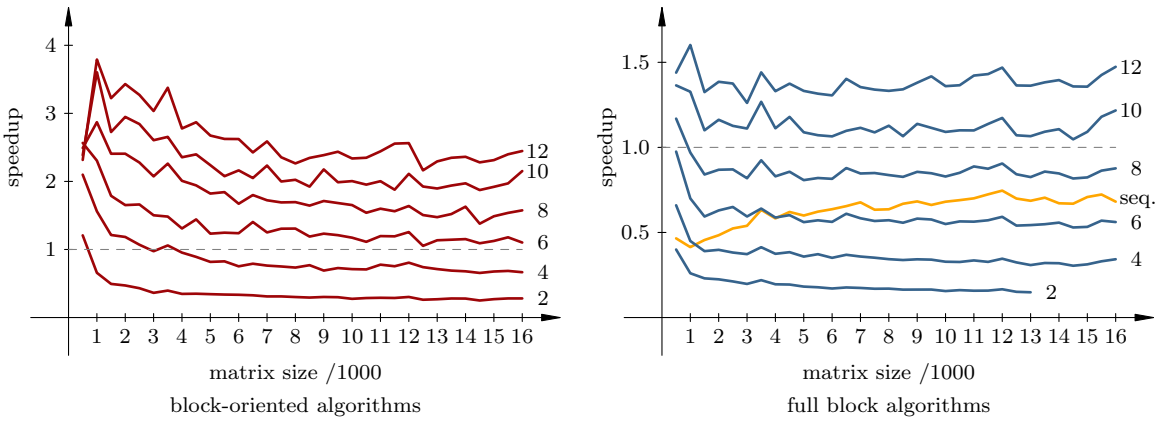

Figure 5.1: Speedup of the shared memory parallel algorithms on 2–12 cores vs. the sequential block-oriented Hari–Zimmermann algorithm (with MKL threaded on 12 cores). On the right-hand side, an additional curve (denoted by seq.) shows the "speedup" of the sequential full block algorithm vs. the block-oriented algorithm. A few missing values for 2 cores on the right-hand side are caused by the limit of execution time on the cluster (12 hours).

The final speed test compares the distributed memory MPI algorithm to the shared memory algorithms. Based on the results of earlier tests, the topmost (distributed memory) level of the algorithm executes the block-oriented algorithm, with the modified modulus strategy, on a collection of MPI processes. On the second (shared memory) level, each MPI process executes the block-oriented thread-parallel algorithm, with the same strategy, on the given block-columns of F and G.

This comparison was carried out only for the largest matrix size of 16000, and the results are given in Table 5.2. Because the communication between the MPI processes is relatively slow (compared to everything else), the MPI algorithm becomes faster than the shared memory block-oriented algorithm when the number of processes p_m is at least 8. At this point the number of cores used is 96, which is huge in comparison to only 12 cores used by the shared memory algorithm. Therefore, the MPI algorithm is designed for really huge matrices, that cannot fit into the shared memory, or the parts of matrices given to a single thread are too big for efficient orthogonalization.

number of		time
MPI processes	cores	`MPI-HZ-BO-32`
2	24	15323.72
4	48	8229.32
6	72	6049.77
8	96	4276.65
10	120	3448.90
12	144	3003.39
14	168	2565.29
16	192	2231.71

number of	time	
cores	`P-HZ-FB-32`	`P-HZ-BO-32`
2	–	42906.93
4	35168.73	18096.72
6	21473.00	10936.10
8	13745.17	7651.86
10	9901.96	5599.25
12	8177.90	4925.56

Table 5.2: Left table: the running times of the hybrid MPI/OpenMP version of the Hari–Zimmermann algorithm for a matrix pair of order 16000. Each MPI process executes the OpenMP algorithm with 12 threads. Each thread performs the block-oriented algorithm with the inner block-size of 32. Right table: the running times for the full block and block-oriented shared memory algorithms for the same matrix.

Finally, we measured the accuracy of algorithms, by comparing the computed generalized singular values with the original ones, set at the beginning of the matrix generation. The test was carried out on a matrix pair of order 5000, which is the largest size that we could test on all algorithms, including `DTGSJA`. This particular pair is moderately ill-conditioned, with a condition measure $\max \sigma_i / \min \sigma_i \approx 6.32 \cdot 10^5$. The relative errors (see Figure 5.2) are U-shaped

over the ordering of values (displayed in the logarithmic scale)—the largest errors occur at both ends of the range, which is understandable, since either $(\Sigma_F)_{ii}$ or $(\Sigma_G)_{ii}$ is small there.

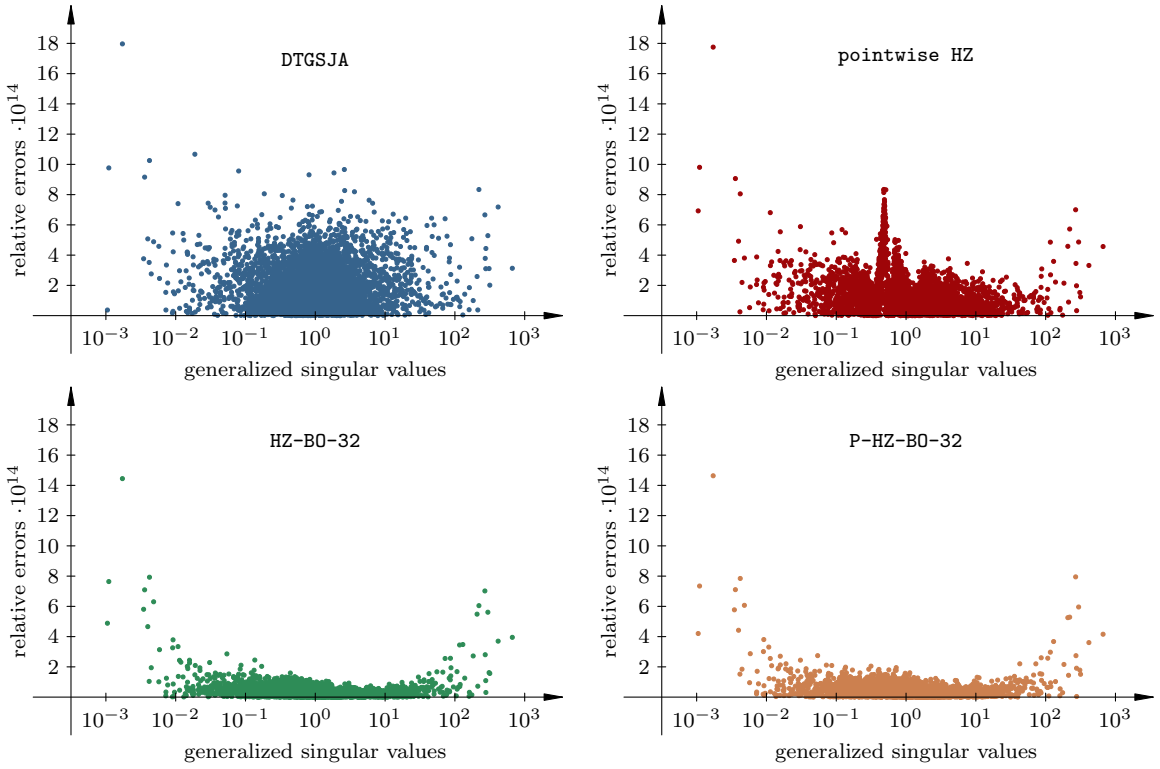

Figure 5.2: Relative errors in the computed generalized singular values for a matrix pair of order 5000 by `DTGSJA`, `pointwise HZ`, `HZ-BO-32`, and `P-HZ-BO-32`.

Quite unexpectedly for the Jacobi-type methods, the pointwise algorithm, which is usually more accurate than any of the blocked algorithms, turns out to be slightly less accurate here. The sequential blocked algorithms are a bit more accurate than the parallel ones, as expected, while the differences in accuracy between the block-oriented and the full block versions are negligible. Maximal relative errors, and average relative errors for different routines are given in Table 5.3. Note that the average relative error for `DTGSJA` is about 5 times larger than for the blocked Hari–Zimmermann algorithms.

	`DTGSJA`	`pointwise HZ`	`HZ-BO-32`	`P-HZ-BO-32`
max_rel	$1.79666 \cdot 10^{-13}$	$1.77529 \cdot 10^{-13}$	$1.44462 \cdot 10^{-13}$	$1.46348 \cdot 10^{-13}$
avg_rel	$1.92500 \cdot 10^{-14}$	$1.25585 \cdot 10^{-14}$	$3.50042 \cdot 10^{-15}$	$4.07475 \cdot 10^{-15}$

Table 5.3: Maximal relative errors, and average relative errors for a matrix pair of order 5000.

The maximal relative errors grow with the condition of the problem, and very slightly with the matrix size. For a matrix pair of order 16000 with $\max \sigma_i / \min \sigma_i \approx 4.51 \cdot 10^8$, the maximal and average relative errors are given in Table 5.4. Only the first few and the last few generalized singular values are responsible for the maximal relative error in all algorithms (the average values are much smaller). If we ignore the 5 smallest and the 5 largest generalized singular values, the ratio between the largest and the smallest σ_i drops to $2.41 \cdot 10^6$, and the obtained errors are similar to those in Table 5.3.

	HZ-BO-32	P-HZ-BO-32	MPI-HZ-BO-32
max_rel	$1.45480 \cdot 10^{-11}$	$1.45499 \cdot 10^{-11}$	$1.45576 \cdot 10^{-11}$
avg_rel	$7.91808 \cdot 10^{-15}$	$7.30987 \cdot 10^{-15}$	$7.62689 \cdot 10^{-15}$
max_rel (6 : 15995)	$2.70628 \cdot 10^{-13}$	$2.64252 \cdot 10^{-13}$	$2.65734 \cdot 10^{-13}$
avg_rel (6 : 15995)	$6.35910 \cdot 10^{-15}$	$5.75219 \cdot 10^{-15}$	$6.06843 \cdot 10^{-15}$

Table 5.4: Maximal relative errors, and average relative errors for a matrix pair of order 16000. If the 5 smallest and the 5 largest generalized singular values are ignored, the respective values are denoted by (6 : 15995).

Note that DTGSJA is unable to handle such large matrices in any reasonable time. On the largest matrices that can be used for comparison, our fastest sequential threaded routine HZ-BO-32 is more than 47 times faster than Lapack's (on this particular architecture), and more accurate, as well. For the fastest explicitly parallel shared memory algorithm P-HZ-BO-32, the speedup factor is 126!

Bear in mind that, these speedup factors do not include the time required for triangularization of both matrices, which is mandatory for DTGSJA, but not necessary for the Hari–Zimmermann method, when G is of full column rank.

The tests conducted on several different architectures give similar results, only the speedup factors are somewhat different, due to differences in architecture.

6. Conclusion

In this paper we developed the implicit Hari–Zimmermann method for computation of the generalized singular values of matrix pairs, where one of the matrices is of full column rank. The method is backward stable, and, if the matrices permit, computes the generalized singular values with small relative errors. Moreover, it is easy to parallelize. A part of the work is motivated by the fact that the current alternatives for the GSVD computation—the best known is DGGSVD from Lapack, are extremely slow for larger matrices.

Unlike the triangular routine DTGSJA from Lapack, which is Kogbetliantz-based, the Hari–Zimmermann method needs no preprocessing to make both matrices triangular. Even when the matrices are preprocessed to a triangular form, the sequential pointwise Hari–Zimmermann method is, for matrices of a moderate size, significantly faster than DTGSJA. On a particular hardware, where threaded MKL (on 12 cores) is used, the pointwise Hari–Zimmermann method is about 3 times faster than DTGSJA, and the computed generalized singular values have, on average, smaller relative errors.

A significant speedup is obtained by blocking of the algorithm, to exploit the efficiency of BLAS-3 operations. For example, the sequential block-oriented version of the algorithm, in the same hardware/software environment, is 15 times faster and even more accurate than the pointwise version, for matrices of order 5000.

Further gains in speed are obtained by explicit shared memory parallelization of blocked algorithms, now with the sequential MKL. The shared memory block-oriented algorithm gives an additional speedup factor of about 2.5, even for much larger matrices. We also developed a distributed memory MPI version of the algorithm, designed for really huge matrices that do not fit into the available shared memory.

The paper contains several theoretical results about the implicit Hari–Zimmermann method. The convergence of the pointwise algorithm with the row-cyclic, column-cyclic, and modulus strategies follows from the results in [14] for the two-sided GEP algorithm. An extension of these results for blocked algorithms is an open problem. The proofs of convergence for various blocked versions of the algorithm are under development, by using the techniques from [15].

Acknowledgment

The authors are indebted to Neven Krajina, for his valuable help with preparing the final draft of the manuscript. Also, we would like to express our gratitude to Prof. Enrique S. Quintana–Ortí, for the computing time provided at Universitat Jaume I, and to the STFC Daresbury Laboratory, for additional computing time provided for the revision of the manuscript.

References

[1] O. Alter, P. O. Brown, D. Botstein, Generalized singular value decomposition for comparative analysis of genome-scale expression data sets of two different organisms, P. Natl. Acad. Sci. USA 100 (6) (2003) 3351–3356.

[2] Z. Bai, J. W. Demmel, Computing the generalized singular value decomposition, SIAM J. Sci. Comput. 14 (6) (1993) 1464–1486.

[3] Z. Bai, H. Zha, A new preprocessing algorithm for the computation of the generalized singular value decomposition, SIAM J. Sci. Comput. 14 (4) (1993) 1007–1012.

[4] K. Bhuyan, S. B. Singh, P. K. Bhuyan, Application of generalized singular value decomposition to ionospheric tomography, Ann. Geophys. 22 (10) (2004) 3437–3444.

[5] J. R. Bunch, B. N. Parlett, Direct methods for solving symmetric indefinite systems of linear equations, SIAM J. Numer. Anal. 8 (4) (1971) 639–655.

[6] P. P. M. de Rijk, A one–sided Jacobi algorithm for computing the singular value decomposition on a vector computer, SIAM J. Sci. Statist. Comput. 10 (2) (1989) 359–371.

[7] A. Deichmöller, Über die Berechnung verallgemeinerter singularer Werte mittels Jacobi-ähnlicher Verfahren, Ph.D. thesis, FernUniversität–Gesamthochschule, Hagen (1990).

[8] Z. Drmač, Computing the singular and the generalized singular values, Ph.D. thesis, FernUniversität–Gesamthochschule, Hagen (1994).

[9] Z. Drmač, A tangent algorithm for computing the generalized singular value decomposition, SIAM J. Numer. Anal. 35 (5) (1998) 1804–1832.

[10] Z. Drmač, A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm, IMA J. Numer. Anal. 19 (2) (1999) 191–213.

[11] S. Falk, P. Langemeyer, Das Jacobische Rotationsverfahren fur reellsymmetrische Matrizenpaare I, in: H. K. Schuff (ed.), Elektronische Datenverarbeitung Folge 7, Friedr. Vieweg & Sohn, Braunschweig, 1960, pp. 30–34.

[12] S. Falk, P. Langemeyer, Das Jacobische Rotationsverfahren fur reellsymmetrische Matrizenpaare II, in: H. K. Schuff (ed.), Elektronische Datenverarbeitung Folge 8, Friedr. Vieweg & Sohn, Braunschweig, 1960, pp. 35–43.

[13] G. Gose, Das Jacobi–Verfahren für $Ax = \lambda Bx$, Z. Angew. Math. Mech. 59 (1979) 93–101.

[14] V. Hari, On cyclic Jacobi methods for the positive definite generalized eigenvalue problem, Ph.D. thesis, FernUniversität–Gesamthochschule, Hagen (1984).

[15] V. Hari, Convergence to diagonal form of block Jacobi-type methods, Numer. Math. 129 (2) (2015) 449–481.

[16] V. Hari, S. Singer, S. Singer, Block-oriented J-Jacobi methods for Hermitian matrices, Linear Algebra Appl. 433 (2010) 1491–1512.

[17] V. Hari, S. Singer, S. Singer, Full block J-Jacobi methods for Hermitian matrices, Linear Algebra Appl. 444 (2014) 1–27.

[18] N. J. Higham, Functions of Matrices: Theory and Computation, SIAM, Philadelphia, 2008.

[19] R. A. Horn, C. R. Johnson, Matrix Analysis, Cambridge University Press, Cambridge, 1985.

[20] IEEE 754-2008, Standard for Floating-Point Arithmetic, New York, NY, USA (Aug. 2008).

[21] S. R. Kuo, W. Yeih, Y. C. Wu, Applications of the generalized singular-value decomposition method on the eigenproblem using the incomplete boundary element formulation, J. Sound. Vib. 235 (5) (2000) 813–845.

[22] F. T. Luk, H. Park, A proof of convergence for two parallel Jacobi SVD algorithms, IEEE Trans. Comput. C–38 (6) (1989) 806–811.

[23] J. Matejaš, Accuracy of one step of the Falk–Langemeyer method, Numer. Algor. 68 (4) (2015) 645–670.

[24] V. Novaković, A hierarchically blocked Jacobi SVD algorithm for single and multiple graphics processing units, SIAM J. Sci. Comput. 37 (2015) C1–C30.

[25] V. Novaković, S. Singer, A GPU-based hyperbolic SVD algorithm, BIT 51 (2011) 1009–1030.

[26] C. C. Paige, Computing the generalized singular value decomposition, SIAM J. Sci. Statist. Comput. 7 (1986) 1126–1146.

[27] C. C. Paige, M. A. Saunders, Towards a generalized singular value decomposition, SIAM J. Numer. Anal. 18 (3) (1981) 398–405.

[28] B. N. Parlett, The Symmetric Eigenvalue Problem, No. 20 in Classics in Applied Mathematics, SIAM, Philadelphia, 1998.

[29] S. Singer, S. Singer, Accuracy of the Hari–Zimmermann method for the generalized singular value decomposition, in preparation (2015).

[30] S. Singer, S. Singer, V. Novaković, D. Davidović, K. Bokulić, A. Ušćumlić, Three-level parallel J-Jacobi algorithms for Hermitian matrices, Appl. Math. Comput. 218 (2012) 5704–5725.

[31] S. Singer, S. Singer, V. Novaković, A. Ušćumlić, V. Dunjko, Novel modifications of parallel Jacobi algorithms, Numer. Alg. 59 (2012) 1–27.

[32] I. Slapničar, Componentwise analysis of direct factorization of real symmetric and Hermitian matrices, Linear Algebra Appl. 272 (1998) 227–275.

[33] I. Slapničar, V. Hari, On the quadratic convergence of the Falk–Langemeyer method, SIAM J. Math. Anal. 12 (1) (1991) 84–114.

[34] G. W. Stewart, Computing the CS decomposition of a partitioned orthogonal matrix, Numer. Math. 40 (3) (1982) 297–306.

[35] G. W. Stewart, Computing the CS and the generalized singular value decompositions, Numer. Math. 46 (4) (1985) 479–491.

[36] B. D. Sutton, Stable computation of the CS decomposition: Simultaneous bidiagonalization, SIAM J. Matrix Anal. Appl. 33 (1) (2012) 1–21.

[37] C. F. Van Loan, Generalizing the singular value decomposition, SIAM J. Numer. Anal. 13 (1) (1976) 76–83.

[38] K. Zimmermann, Zur Konvergenz eines Jacobiverfahren für gewönliche und verallgemeinerte Eigenwertprobleme, Dissertation no. 4305, ETH, Zürich (1969).