ISSN 1360-1725

Parallel Singular Value Decomposition via the Polar Decomposition

N. J. Higham and P. Papadimitriou

Numerical Analysis Report No. 239

October 1993

Manchester Centre for Computational Mathematics
Numerical Analysis Reports

DEPARTMENTS OF MATHEMATICS

Reports available from:
Department of Mathematics
University of Manchester
Manchester M13 9PL
England

And over the World-Wide Web from URLs
`http://www.ma.man.ac.uk/MCCM`
`http://www.ma.man.ac.uk/~nareports`

Parallel Singular Value Decomposition via the Polar Decomposition

Nicholas J. Higham* Pythagoras Papadimitriou†

October 30, 1993

Abstract

A new method is described for computing the singular value decomposition (SVD). It begins by computing the polar decomposition and then computes the spectral decomposition of the Hermitian polar factor. The method is particularly attractive for shared memory parallel computers with a relatively small number of processors, because the polar decomposition can be computed efficiently on such machines using an iterative method developed recently by the authors. This iterative polar decomposition method requires only matrix multiplication and matrix inversion kernels for its implementation and is designed for full rank matrices; thus the proposed SVD method is intended for matrices that are not too close to being rank-deficient. On the Kendall Square KSR1 virtual shared memory computer the new method is up to six times faster than a parallelized version of the LAPACK SVD routine, depending on the condition number of the matrix.

Key words. singular value decomposition, polar decomposition, numerical stability, LAPACK, level 3 BLAS, Kendall Square Research KSR1 computer

AMS subject classifications. primary 65F15

*Department of Mathematics, University of Manchester, Manchester, M13 9PL, England (na.nhigham@na-net.ornl.gov). The work of this author was supported by Science and Engineering Research Council grant GR/H52139, and by the EEC Esprit Basic Research Action Programme, Project 6634 (APPARC).

†Department of Mathematics, University of Manchester, Manchester, M13 9PL, England (na.papadimitriou@na-net.ornl.gov). Current address: Data Information Systems PLC, 125 Thessalonikis, N. Philadelphia, Athens 142 43, Greece. This author was supported by an SERC Research Studentship.

1 Background and Motivation

A modern theme in matrix computations is to build algorithms out of existing kernels. It is standard practice to express algorithms in terms of BLAS operations, since efficient implementations of the three levels of BLAS are available on a wide range of machines; LAPACK, for example, uses the BLAS throughout [1]. Higher level building blocks can also be used. Bai and Demmel [2] are building a toolbox from which methods for solving the nonsymmetric eigenvalue problem can be constructed; their building blocks include matrix multiplication, matrix inversion, LU and QR factorization, and condition number estimation.

In this work we propose an algorithm for computing the singular value decomposition (SVD) that requires just three building blocks: matrix multiplication, matrix inversion, and solution of the Hermitian eigenproblem. Our approach is motivated by the fact that for the Kendall Square KSR1 virtual shared memory computer there is not available, at the time of writing, a library of matrix computation software that takes full advantage of the machine. The manufacturer does, however, supply a KSRlib/BLAS library that contains a highly optimized level-3 BLAS routine `xGEMM`[1] [12], together with an implementation of LAPACK, KSRlib/LAPACK [13], that calls the KSRlib/BLAS library. Our aim was to produce an SVD routine that is faster on the KSR1 than the KSRlib/LAPACK routine `xGESVD` (which implements the Golub–Reinsch algorithm), but without sacrificing numerical stability. Rather than code any of the existing parallel SVD algorithms [3], [4], [16], we wanted to make use of parallel codes that we had already developed for computing the polar decomposition and solving the Hermitian eigenproblem. Our aim has been achieved and we think that our new algorithm merits consideration for other parallel machines that lack an optimized SVD routine.

Unlike standard techniques for computing the SVD, our method is influenced by the singular value distribution. Specifically, the time required to compute the SVD of A is, very roughly, proportional to $\log_2 \kappa_2(A)$, where $\kappa_2(A)$ is the 2-norm condition number. Since, furthermore, the iteration underlying our method can fail to converge when A is rank-deficient, the method is intended for matrices that are not too nearly rank-deficient. There are various applications in which this requirement is met. When the iteration does converge the method is almost always numerically stable, and in any case it is easy to test the stability a posteriori.

We briefly mention two other well-known ways to compute the SVD of $A \in \mathbb{C}^{m\times n}$ given a Hermitian eigensolver. One involves computing the eigensystem of A^*A; we reject

[1] The '`x`' in routine names stands for the Fortran data type: in this case, `S` or `C`.

this method on the grounds that it is numerically unstable unless A is well-conditioned [6, Sec. 8.3.2].

The second approach is to compute the eigensystem of the $(m+n) \times (m+n)$ matrix $\left[\begin{smallmatrix} 0 & A \\ A^* & 0 \end{smallmatrix}\right]$, from which A's SVD can be "read off" [6, p. 427]. Because of the expanded dimension this approach is substantially slower for large dimensions on the KSR1 than the method we propose (this can be seen from the results in Section 3), and the storage requirements may in any case be prohibitive.

We describe the new algorithm in Section 2 and give results of its implementation on the KSR1 in Section 3. Our conclusions are given in Section 4.

2 The SVD Algorithm

Any matrix $A \in \mathbb{C}^{m \times n}$ $(m \geq n)$ has a polar decomposition $A = UH$, where $U \in \mathbb{C}^{m \times n}$ has orthonormal columns and $H = (A^*A)^{1/2} \in \mathbb{C}^{n \times n}$ is Hermitian positive semidefinite (see [11, Section 7.3], for example). If A is real then the polar factors are real. Let H have the spectral decomposition $H = VDV^*$, where V is unitary and $D = \mathrm{diag}(d_i)$ with $d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$. Then $A = P\Sigma Q^*$ is an SVD where $P = UV$, $\Sigma = D$ and $Q = V$. Note that this is an "economy size" SVD with an $m \times n$ P, which is all that is needed to solve the least squares problem, for example. If the full SVD is required, we could first compute a QR factorization $A = Q\left[\begin{smallmatrix} R \\ 0 \end{smallmatrix}\right]$ and then find the SVD of R. The polar decomposition is often expressed in terms of the SVD, but we are turning the tables to regard the SVD as a combination of a polar decomposition and a spectral decomposition. In floating point arithmetic, rounding errors can cause some of the computed $\widehat{d_i}$ to be negative when H is nearly singular. The following algorithm takes account of this possibility. Although we state our algorithms for complex A, they require only real arithmetic when A is real.

Algorithm SVD.
Given $A \in \mathbb{C}^{m \times n}$ with $m \geq n$ this algorithm computes an SVD $A = P\Sigma Q^*$.

(1) Compute the polar decomposition $A = UH$.

(2) Compute the spectral decomposition $H = VDV^*$.

(3) $P = UVD_S$, $\Sigma = |D|$, $Q = V$, where $D_S = \mathrm{diag}(\mathrm{sign}(d_i))$.

For the algorithm to be successful we need efficient and stable ways to carry out steps (1) and (2). And for a particular implementation we need either a proof that the algorithm is stable or an a posteriori stability test that can be used as a fourth stage of the algorithm.

An operation count for the case $m = n$ gives insight into the efficiency. If we compute the polar decomposition using the Newton iteration of Higham [7],

$$Y_{k+1} = \tfrac{1}{2}(\gamma_k Y_k + \gamma_k^{-1} Y_k^{-*}) \quad Y_0 = A \in \mathbb{C}^{n\times n} \tag{2.1}$$

(for which $Y_k \to U$ as $k \to \infty$ with a quadratic rate of convergence, and where γ_k is an acceleration parameter), the cost is in practice at most $20n^3$ flops, where a flop is an addition, multiplication or division; the precise cost depends on the condition number of A. The eigendecomposition of H can be obtained using the QR algorithm in approximately $9n^3$ flops [6, p. 424]. Accounting for the matrix multiplication in step (3) we have a total of $31n^3$ flops, which is somewhat more than the $22n^3$ flops required by the Golub–Reinsch SVD algorithm [6, p. 239]. Thus, on a machine where flops are a good measure of cost, this particular implementation of Algorithm SVD is unlikely to be more efficient than the Golub–Reinsch SVD algorithm.

For parallel computation the picture is different because matrix multiplication, and to a lesser extent matrix inversion, can usually be accomplished much more quickly than the flop counts indicate. Therefore the Newton iteration (2.1) could be attractive. However, even better for the KSR1 is the following algorithm of Higham and Papadimitriou [9].

Algorithm Polar.
Given $A \in \mathbb{C}^{m\times n}$ $(m \ge n)$ of full rank, a convergence tolerance tol and a positive integer p, this algorithm computes the polar decomposition $A = UH$.

$\xi_i = \frac{1}{2}\left(1 + \cos\frac{(2i-1)\pi}{2p}\right)$, $\alpha_i^2 = 1/\xi_i - 1$, $i = 1\colon p$.
$X_0 = A/\|A\|_F$; $k = 0$
repeat
    $C_k = X_k^* X_k$
    $\rho = \|I - C_k\|_F$
    if $\rho \le$ tol, goto (1), end
    Compute the acceleration parameter μ_k.
(∗) $X_{k+1} = \dfrac{1}{p}\mu_k X_k \displaystyle\sum_{i=1}^{p} \frac{1}{\xi_i}(\mu_k^2 C_k + \alpha_i^2 I)^{-1}$,
    $k = k + 1$
end
(1) $U = X_k$
$H_1 = U^* A$
$H = \frac{1}{2}(H_1 + H_1^*)$ (to ensure the computed H is Hermitian)

The iteration $(*)$ in Algorithm Polar is globally convergent to the polar factor U, with order $2p$. The iteration is well suited to shared memory parallel computers because each of the p inversions in $(*)$ can be done in parallel. On the KSR1, the implementation of Algorithm Polar described in [9], which has p equal to the number of processors ($p \leq 16$ in [9]), runs an order of magnitude faster than the KSRlib/LAPACK SVD routine `xGESVD`. We emphasize, however, that there is little point in choosing p bigger than 16, because the increased order of convergence brings little or no reduction in the number of iterations. When more than 16 processors are available they can be employed to share the work of forming each of the p matrix inverses.

Algorithm Polar requires that A have full rank. If A is rank deficient then so are all the iterates X_k. The iteration still converges, but to a matrix with singular values 0 and 1; we obtain the correct H, but a rank-deficient U. In practice, when A is rank-deficient rounding errors usually cause the iteration to converge to a matrix with orthonormal columns (to within the convergence tolerance) and we may still obtain a satisfactory computed polar decomposition, but rounding errors can also cause failure to converge. Note that U is not unique when A is rank deficient, so there are many possible matrices to which the iteration could converge in the presence of rounding errors.

Following the approach of Higham and Schreiber [10] we can add an initial complete orthogonal decomposition step to Algorithm SVD and apply Algorithm Polar to the resulting square nonsingular upper triangular matrix. We thereby obtain an SVD algorithm that works for *any* $A \in \mathbb{C}^{m\times n}$. However, efficient computation of the complete orthogonal decomposition on a parallel machine is difficult and requires building blocks in addition to those that we are assuming are available, so we do not explore this idea here.

We now investigate the numerical stability of Algorithm SVD. We will use the 2-norm $\|A\|_2 = (\rho(A^*A))^{1/2}$, where ρ is the spectral radius, and the Frobenius norm $\|A\|_F = (\sum_{i,j} |a_{ij}|^2)^{1/2}$. Let the computed polar factors $\widehat{U}$ and $\widehat{H}$ satisfy

$$A + E = \widehat{U}\widehat{H},$$

and let the computed matrices $\widehat{V}$ and $\widehat{D}$ of eigenvectors and eigenvalues of $\widehat{H}$ satisfy

$$\widehat{H} + F = \widehat{V}\widehat{D}\widehat{V}^*.$$

Then, ignoring the rounding errors in forming $\widehat{P}$ (which have a negligible effect on the final error bound), we have

$$\begin{aligned}\widehat{P}\widehat{\Sigma}\widehat{Q}^* &= \widehat{U}\widehat{V}\widehat{D}\widehat{V}^* \\ &= \widehat{U}(\widehat{H} + F)\end{aligned}$$

$$= A + E + \widehat{U}F$$
$$\equiv A + \Delta A.$$

We will assume that the eigensolver is stable to precision $\mathrm{tol} \ll 1$, so that

$$\|F\|_2 \le c_n \mathrm{tol} \|\widehat{H}\|_2, \qquad \|\widehat{V}^*\widehat{V} - I\|_2 \le c_n \mathrm{tol},$$

where c_n is a constant, and that the polar factors are computed from Algorithm Polar. The convergence test in Algorithm Polar ensures that $\widehat{U}^*\widehat{U} = I + K$ with $\|K\|_F \le \mathrm{tol}$, which implies that $\|\widehat{U}\|_2 \le 1 + \mathrm{tol}$. From the above analysis it follows that the computed $\widehat{P}$ and $\widehat{Q}$ satisfy $\|\widehat{P}^*\widehat{P} - I\|_2 = O(\mathrm{tol})$ and $\|\widehat{Q}^*\widehat{Q} - I\|_2 = O(\mathrm{tol})$, and that

$$\begin{aligned}\|\Delta A\|_2 &\le \|E\|_2 + \|\widehat{U}\|_2 \|F\|_2 \\ &\le \|E\|_2 + c_n \mathrm{tol} \|A\|_2 + O(\mathrm{tol}^2).\end{aligned}$$

Thus the algorithm is stable provided that

$$\|A - \widehat{U}\widehat{H}\|_2 = \|E\|_2 = O(\mathrm{tol}\|A\|_2). \tag{2.2}$$

Ignoring the rounding errors in the formation of $\widehat{H}_1$ and $\widehat{H}$ (which again are negligible), we have $A - \widehat{U}\widehat{H} = A - \frac{1}{2}\widehat{U}(\widehat{H}_1 + \widehat{H}_1^*)$, so that, since $\widehat{H}_1 = \widehat{U}^*A$,

$$\begin{aligned}\widehat{U}^*(A - \widehat{U}\widehat{H}) &= \widehat{U}^*A - \tfrac{1}{2}(I + K)(\widehat{H}_1 + \widehat{H}_1^*) \\ &= \tfrac{1}{2}(\widehat{H}_1 - \widehat{H}_1^*) + O(\mathrm{tol}\|A\|_2).\end{aligned}$$

Hence we see that when A is square, so that $\|\widehat{U}^*(A - \widehat{U}\widehat{H})\|_2 = \|A - \widehat{U}\widehat{H}\|_2(1 + O(\mathrm{tol}))$, condition (2.2) is equivalent to (converting to the more computationally convenient Frobenius norm)

$$\tfrac{1}{2}\|\widehat{H}_1 - \widehat{H}_1^*\|_F = O(\mathrm{tol}\|A\|_F), \tag{2.3}$$

which is trivial to check at the end of Algorithm Polar and avoids the matrix multiplication necessary to form $A - \widehat{U}\widehat{H}$.

Is condition (2.2) satisfied? If the "optimal" acceleration parameter μ_k, as defined in [9], is used in Algorithm Polar then the answer is no, because $\|A - \widehat{U}\widehat{H}\|_2$ is typically of order $\kappa_2(A)\mathrm{tol}\|A\|_2$; this is the observed behaviour and it can be partly explained theoretically [9]. If we use the unaccelerated iteration ($\mu_k \equiv 1$) then more iterations may be required, but $\|A - \widehat{U}\widehat{H}\|_2$ is now usually of order $\mathrm{tol}\|A\|_2$ (indeed it is of this order in all our experiments) and again there is some supporting theory [9].

Numerical experiments provide further insight into the stability of Algorithm SVD when it is used in conjunction used with Algorithm Polar. We have experimented in

Matrix	Iters	$\|A - \widehat{U}\widehat{H}\|_2/\|A\|_2$	$\|A - \widehat{P}\widehat{\Sigma}\widehat{Q}\|_2/\|A\|_2$
`vand(25)`	14	5.42e-14	5.43e-14
`cycol(16)`	15	1.34e-14	1.34e-14
`randsvd([200,100],1.01e0)`	1	8.40e-16	3.61e-15
`randsvd([200,100],1e1)`	2	1.02e-15	4.83e-15
`randsvd([200,100],1e4)`	4	1.11e-14	1.17e-14
`randsvd([200,100],1e8)`	7	3.60e-14	3.62e-14
`randsvd([200,100],1e12)`	9	4.03e-14	4.05e-14
`randsvd([200,100],1e16)`	12	4.96e-14	4.97e-14

Table 2.1: Results for Algorithm SVD with Algorithm Polar.

Matlab, for which the unit roundoff $u \approx 1.1 \times 10^{-16}$. We took $\mu_k \equiv 1$, $p = 16$ and $\mathrm{tol} = mu$ in Algorithm Polar, and used the QR algorithm to compute the eigensystem of H (that is, we used Matlab's `eig` function). Results are shown in Table 2.1 for three types of matrix from the test collection in [8]. `Vand(25)` is the Vandermonde matrix of order 25 with (i,j) element $((j-1)/24)^{i-1}$; it has rank 21 to working precision. `Cycol(16)` is a matrix of the form $[B\ B\ B\ B]$, where B is a random 16×4 matrix, and so it has rank 4 to working precision. `Randsvd([m,n],kappa))` is a random $m \times n$ matrix with 2-norm condition number `kappa` and exponentially distributed singular values.

The key features of the results are as follows.

1. The more ill-conditioned A is, the more iterations are required by Algorithm Polar.

2. The maximum value of $\|X^*X - I\|_F$ for $X = \widehat{P}$ or $X = \widehat{Q}$ over all the matrices in the table is 3.05e-14, so $\widehat{P}$ and $\widehat{Q}$ are unitary to within the tolerance in each case, as the analysis above shows they must be. The relative residuals for the SVD are all less than $m\mathrm{tol}$. Therefore the algorithm is performing stably in all these examples, even though some of the matrices are rank-deficient to working precision.

3. The third and fourth columns of Table 2.1 confirm that the polar decomposition relative residual is an excellent predictor of the SVD relative residual. Furthermore, the values of $\frac{1}{2}\|\widehat{H}_1 - \widehat{H}_1^*\|_F$ (not shown in the table) are all within a factor 4 of $\|A - \widehat{U}\widehat{H}\|_2$, showing that the stability test (2.3) is quite reliable, even for rectangular matrices.

It is possible to find examples of rank deficient matrices for which Algorithm Polar does not converge. An example is a singular Jordan block. We have not found any examples where the algorithm converges but gives an unstable computed polar decomposition.

Order	Standard SSYEV	Modified SSYEV
64	0.71	0.29
128	5.58	1.78
256	40.51	17.01
512	369.56	148.53
1024	2534.84	1150.37

Table 3.1: Times in seconds for SSEYV.

3 Experiments on the KSR1

We have implemented Algorithm SVD on the Kendall Square KSR1 virtual shared memory computer. Each node of the KSR1 is a superscalar 64-bit processor with a 40 Mflop peak performance and 32 Mbytes of local cache memory. We had access to 16 processors of the 32-processor KSR1 at the University of Manchester. Our codes are written in KSR Fortran, which is Fortran 77 with some parallel and Fortran 90 extensions. We used single precision arithmetic, for which the unit roundoff $u \approx 1.1 \times 10^{-16}$.

Although the KSRlib/BLAS library contains a highly optimized SGEMM routine, the other level 3 BLAS routines do not appear to be highly optimized for the KSR1 [14]. The SGEMM routine supports four optional parameters in addition to those in the standard level 3 BLAS specification; they permit controlled exploitation of parallelism.

To compute the polar decomposition $A = UH$ we used the KSR1 implementation of Algorithm Polar from [9], with tol $= mu$ and with no acceleration ($\mu_k \equiv 1$). We considered two approaches to computing the spectral decomposition of H. The first is to use the LAPACK driver routine SSYEV, which carries out the symmetric QR algorithm, involving a reduction to tridiagonal form followed by an iterative QR phase. To obtain better performance than that provided by the KSRlib/LAPACK SSYEV we modified the SGEMM calls to include the four KSR1-specific additional parameters; these SGEMM calls occur only in the auxiliary routine SLARFB, which applies a product of Householder transformations (stored in "compact WY" form) to a rectangular matrix.

Table 3.1 reports run times for standard SSYEV on one processor and for the modified SSYEV on 16 processors, for matrices of several dimensions. In both cases we used a block size of 16, which we found to give the best performance. Each timing is for one particular matrix, but the times vary only slightly with the matrix.

The second approach is to use the block Jacobi method (see [6, Section 8.5.11] for details of this method.) Papadimitriou [15] has implemented several versions of the block Jacobi method on the KSR1. We report results for two implementations: Algorithm

Order	Algorithm BJ1	Algorithm BJ2
64	24.58	29.74
128	39.51	52.15
256	96.47	136.02
512	145.35	236.78
1024	518.20	1014.07

Table 3.2: Times in seconds for block Jacobi Algorithms BJ1 and BJ2.

BJ1 and Algorithm BJ2. Algorithm BJ1 uses a parallel rotation ordering and solves the spectral decomposition subproblems using `SSYEV`. The block size is chosen as $n/(2p)$, where p is the number of processors. A block rotation is skipped whenever the off-diagonal elements of the subproblem are below a dynamically varying threshold. A technique suggested by Davies and Modi [5] is used to terminate the sweeps prematurely, once the matrix is reasonably close to diagonal form, and then to transform directly to diagonal form (to within the convergence tolerance) using perturbation formulae. The Davies–Modi scheme is numerically stable only if the matrix to which it is applied has "sufficiently distinct" eigenvalues λ_i: stability degrades as $\lambda_i - \lambda_j \to 0$ for $i \neq j$, since the algorithm involves division by $\lambda_i - \lambda_j$ for all $i \neq j$. Algorithm BJ2 omits the Davies–Modi scheme and uses the threshold strategy only on the first 6 block sweeps. Full details of these algorithms are given in [15]. Timings for Algorithms BJ1 and BJ2 are given in Table 3.2.

We see from the tables that modified `SSYEV` is the fastest method for $n \leq 256$, but Algorithms BJ1 and BJ2 are significantly faster for $n = 1024$. Although Algorithm BJ1 is nearly twice as fast as Algorithm BJ2 for $n = 1024$, we will use Algorithm BJ2 for the remaining experiments because of its guaranteed stability.

We now compare our new method with the KSRlib/LAPACK driver routine `SGESVD`. By modifying the routine in the same way as for `SSYEV` we were able to reduce the run time. Timings for square matrices of five different dimensions are given in Table 3.3. These timings are largely unaffected by the singular value distribution.

In Tables 3.4 and 3.5 we give timings for Algorithm SVD with both Algorithm BJ1 and modified `SSYEV` as the eigensolver. The timings are for a random matrix with exponentially distributed singular values and 2-norm condition number 1.01 (Table 3.4) or 10^{12} (Table 3.5). Algorithm SVD is up to 6.2 times faster than modified `SGESVD` for $\kappa_2(A) = 1.01$, and up to 2.6 times faster for $\kappa_2(A) = 10^{12}$. For $n = 1024$ with Algorithm BJ2, the percentage of the run time taken by Algorithm Polar is 15% for $\kappa_2(A) = 1.01$

Order	Standard SGESVD	Modified SGESVD
64	1.35	1.35
128	9.77	9.12
256	122.39	101.03
512	980.10	880.02
1024	9500.00	7500.00

Table 3.3: Times in seconds for SGESVD.

Order	With Algorithm BJ2	With modified SSYEV
64	30.29	0.84
128	53.19	2.82
256	140.24	21.23
512	262.42	174.17
1024	1203.83	1340.13

Table 3.4: Times in seconds for Algorithm SVD, $\kappa_2(A) = 1.01$.

and 65% for $\kappa_2(A) = 10^{12}$.

For $n = 1024$ in Table 3.5, Algorithm SVD with Algorithm BJ2 is running at about 23 megaflops (compared with the peak of 640 megaflops for 16 processors of the KSR1), while modified SGESVD for the same dimension runs at about 3 megaflops. For comparison, the highly optimized SGEMM runs at 384 megaflops.

Order	With Algorithm BJ2	With modified SSYEV
64	31.66	2.21
128	59.59	9.22
256	172.10	53.09
512	490.80	402.55
1024	2887.70	3024.00

Table 3.5: Times in seconds for Algorithm SVD, $\kappa_2(A) = 10^{12}$.

4 Conclusions

Algorithm SVD is the fastest stable method we know for computing the SVD of large, full rank matrices on the KSR1. The algorithm is worth considering for other types of parallel machine for which a fast Hermitian eigensolver, but not a fast SVD solver, is available. For some architectures, the Newton iteration (2.1) could be a faster way to compute the polar decomposition than Algorithm Polar. Further experimentation on different architectures is needed.

We emphasize that our modified version of `SGESVD` is not the fastest possible implementation of the Golub–Reinsch algorithm on the KSR1. With a major rewrite to exploit the machine a faster code could be obtained—but, of course, with a complete loss of portability.

Finally, we note that the SVD can be computed by (block) one-sided Jacobi or two-sided Jacobi methods (which may require a preliminary QR factorization) [3], [4], [16]. We have not tried any of these methods on the KSR1. A major advantage of our approach is that coding is straightforward, given the required matrix multiplication, matrix inversion and Hermitian eigenproblem building blocks.

5 Acknowledgements

We thank Zhaojun Bai and Des Higham for suggesting improvements to the manuscript.

References

[1] E. Anderson, Z. Bai, C. H. Bischof, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, S. Ostrouchov, and D. C. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. xv+118+listings pp. ISBN 0-89871-294-7.

[2] Zhaojun Bai and James W. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, Part I. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Volume I*, Richard F. Sincovec, David E. Keyes, Michael R. Leuze, Linda R. Petzold, and Daniel A. Reed, editors, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1993, pages 391–398.

[3] Michael Berry and Ahmed H. Sameh. An overview of parallel algorithms for the singular value and symmetric eigenvalue problems. *J. Comp. Appl. Math.*, 27:191–213, 1989.

[4] Richard P. Brent, Franklin T. Luk, and Charles F. Van Loan. Computation of the singular value decomposition using mesh-connected processors. *J. VLSI and Computer Systems*, 1:242–270, 1985.

[5] Roy O. Davies and J. J. Modi. A direct method for completing eigenproblem solutions on a parallel computer. *Linear Algebra and Appl.*, 77:61–74, 1986.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Second edition, Johns Hopkins University Press, Baltimore, MD, USA, 1989. xix+642 pp. ISBN 0-8018-3772-3 (hardback), 0-8018-3739-1 (paperback).

[7] Nicholas J. Higham. Computing the polar decomposition—with applications. *SIAM J. Sci. Stat. Comput.*, 7(4):1160–1174, October 1986.

[8] Nicholas J. Higham. The Test Matrix Toolbox for Matlab. Numerical Analysis Report No. 237, Manchester Centre for Computational Mathematics, Manchester, England, December 1993. 76 pp.

[9] Nicholas J. Higham and Pythagoras Papadimitriou. A parallel algorithm for computing the polar decomposition. *Parallel Computing*, 20(8):1161–1173, August 1994.

[10] Nicholas J. Higham and Robert S. Schreiber. Fast polar decomposition of an arbitrary matrix. *SIAM J. Sci. Stat. Comput.*, 11(4):648–655, July 1990.

[11] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985. xiii+561 pp. ISBN 0-521-30586-1.

[12] Kendall Square Research Corporation. *KSRlib/BLAS Library, Version 1.0, Installation Guide and Release Notes*. Waltham, MA, 1993.

[13] Kendall Square Research Corporation. *KSRlib/LAPACK Library, Version 1.0b BETA, Installation Guide and Release Notes*. Waltham, MA, 1993.

[14] Pythagoras Papadimitriou. The KSR1—A numerical analyst's perspective. Numerical Analysis Report No. 242, University of Manchester, Manchester, England, December 1993.

[15] Pythagoras Papadimitriou. *Parallel Solution of SVD-Related Problems, With Applications*. PhD thesis, University of Manchester, Manchester, England, October 1993.

[16] Charles F. Van Loan. The block Jacobi method for computing the singular value decomposition. In *Computational and Combinatorial Methods in Systems Theory*, C. I. Byrnes and A. Lindquist, editors, Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1986, pages 245–255.