Chapter 1
A New Parallel Algorithm for Computing the Singular Value Decomposition[*]

Nicholas J. Higham[†] Pythagoras Papadimitriou[‡]

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.

1 Background and Motivation

A modern theme in matrix computations is to build algorithms out of existing kernels. 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] [6], together with an implementation of LAPACK, KSRlib/LAPACK [7], 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

[*]Appears as: N. J. Higham and P. Papadimitriou. A new parallel algorithm for computing the singular value decomposition. In J. G. Lewis, editor, *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 80–84. Society for Industrial and Applied Mathematics, Philadelphia, 1994. This paper is a shortened version of [4].

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

xGESVD (which implements the Golub–Reinsch algorithm), but without sacrificing numerical stability. Rather than code any of the existing parallel SVD algorithms [1], [2], [10], 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.

2 The SVD Algorithm

Any matrix $A \in \mathbf{C}^{m\times n}$ $(m \ge n)$ has a polar decomposition $A = UH$, where $U \in \mathbf{C}^{m\times n}$ has orthonormal columns and $H = (A^*A)^{1/2} \in \mathbf{C}^{n\times n}$ is Hermitian positive semidefinite. 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 $d_1 \ge d_2 \ge \cdots \ge d_n \ge 0$. 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 $A \in \mathbf{C}^{m\times n}$ $(m \ge n)$ 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). For step (1) we consider the following algorithm of Higham and Papadimitriou [5].

```
**Algorithm Polar**.
Given $A \in \mathbf{C}^{m\times n}$ $(m \ge n)$ of full rank, a convergence tolerance tol and
a positive integer $p$, this algorithm computes the polar decomposition
$A = UH$.
$\xi_i = \frac{1}{2}(1 + \cos\frac{(2i-1)\pi}{2p})$, $\alpha_i^2 = 1/\xi_i - 1$, $i = 1\colon p$.
$X_0 = A/\|A\|_F$; $k = 0$
repeat
        $C_k = X_k^* X_k$
        $\rho = \|I - C_k\|_F$
        if $\rho \le$ tol, goto (1), end
        Compute the acceleration parameter $\mu_k$.
```

```
(*)         $X_{k+1} = \frac{1}{p}\mu_k X_k \sum_{i=1}^{p} \frac{1}{\xi_i}(\mu_k^2 C_k + \alpha_i^2 I)^{-1}$,
            $k = k + 1$
        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 [5], which has p equal to the number of processors ($p \leq 16$ in [5]), 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.

The numerical stability of Algorithm SVD is investigated in [4]. The conclusion is that Algorithm SVD is stable if a stable eigensolver is used and if Algorithm Polar produces a stable computed polar decomposition $A \approx \widehat{U}\widehat{H}$. If the "optimal" acceleration parameter μ_k, as defined in [5], is used in Algorithm Polar, then $\|A - \widehat{U}\widehat{H}\|_2$ is typically of order $\kappa_2(A)\mathrm{tol}\|A\|_2$; this is the observed behaviour and it can be partly explained theoretically [5]. For the unaccelerated iteration ($\mu_k \equiv 1$) 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 [5]. There is an inexpensive a posteriori test for stability of Algorithm Polar; see [4].

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 [8]. 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 [5], with $\mathrm{tol} = mu$ and with no acceleration ($\mu_k \equiv 1$). We

Table 1
Times in seconds for eigensolvers.

Order	Standard `SSYEV`	Modified `SSYEV`	Algorithm BJ
64	0.71	0.29	29.74
128	5.58	1.78	52.15
256	40.51	17.01	136.02
512	369.56	148.53	236.78
1024	2534.84	1150.37	1014.07

Table 2
Times in seconds for `SGESVD`.

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

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.

Table 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 [3, Section 8.5.11] for details of this method.) Papadimitriou [9] has implemented several versions of the block Jacobi method on the KSR1. The version used here, Algorithm BJ, 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. We see from Table 1 that modified `SSYEV` is the fastest method for $n \leq 512$, but Algorithm BJ is the fastest for $n = 1024$.

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 2. These timings are largely unaffected by the singular value distribution.

In Tables 3 and 4 we give timings for Algorithm SVD with both Algorithm BJ 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) or 10^{12} (Table 4). 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 BJ, the percentage of the run time taken by Algorithm Polar is 15% for $\kappa_2(A) = 1.01$ and 65% for $\kappa_2(A) = 10^{12}$.

Table 3
Times in seconds for Algorithm SVD, $\kappa_2(A) = 1.01$.

Order	With Algorithm BJ	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 4
Times in seconds for Algorithm SVD, $\kappa_2(A) = 10^{12}$.

Order	With Algorithm BJ	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

For $n = 1024$ in Table 4, Algorithm SVD with Algorithm BJ 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.

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.

References

[1] M. Berry and A. H. Sameh, *An overview of parallel algorithms for the singular value and symmetric eigenvalue problems*, J. Comp. Appl. Math., 27 (1989), pp. 191–213.

[2] R. P. Brent, F. T. Luk, and C. F. Van Loan, *Computation of the singular value decomposition using mesh-connected processors*, J. VLSI and Computer Systems, 1 (1985), pp. 242–270.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, second ed., 1989.

[4] N. J. Higham and P. Papadimitriou, *Parallel singular value decomposition via the polar decomposition*, Numerical Analysis Report No. 239, University of Manchester, England, Oct. 1993.

[5] ——, *A parallel algorithm for computing the polar decomposition*, Parallel Computing, 20 (1994), pp. 1161–1173.

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

[7] ——, *KSRlib/LAPACK Library, Version 1.0b BETA, Installation Guide and Release Notes*, Waltham, MA, 1993.

[8] P. Papadimitriou, *The KSR1—A numerical analyst's perspective*, Numerical Analysis Report No. 242, University of Manchester, England, Dec. 1993.

[9] ——, *Parallel Solution of SVD-Related Problems, With Applications*, PhD thesis, University of Manchester, England, Oct. 1993.

[10] C. 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, eds., Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1986, pp. 245–255.