A SYSTOLIC ARCHITECTURE FOR
THE SINGULAR VALUE DECOMPOSITION

Richard P. Brent*
Franklin T. Luk

TR 82-522
September 1982

Department of Computer Science
Upson Hall
Cornell University
Ithaca, New York 14853

*Department of Computer Science, Australian National University, Canberra, A.C.T. 2600, Australia

Supported in part by the U.S. Army Office under grant DAAG 29-79-C0124 and in part by the Mathematical Sciences Research Centre and the Centre for Mathematical Analysis, Australian National University.
A Systolic Architecture for
the Singular Value Decomposition

Richard P. Brent
Department of Computer Science
Australian National University
Canberra, A.C.T. 2600
Australia

Franklin T. Luk*
Department of Computer Science
Cornell University
Ithaca, N.Y. 14853
U.S.A.

TR-CS-82-09
August, 1982

* Supported in part by the U.S. Army Office under grant DAAG 29-79-C0124 and in part by the Mathematical Sciences Research Centre and the Centre for Mathematical Analysis, Australian National University.
Abstract

We propose a systolic architecture for computing a singular value decomposition of an \( m \times n \) matrix, where \( m \geq n \). Our algorithm is stable and requires only \( O(mn) \) time on a linear array of \( O(n) \) processors. Extensions to algorithms for two-dimensional arrays are also discussed.

Key Words and Phrases: Systolic arrays, singular value decomposition, Hestenes method, threshold Jacobi method, real-time computation.
1. **Introduction**

A singular value decomposition (SVD) of an \( m \times n \) \((m \geq n)\) matrix \( A \) is its factorization into the product of three matrices:

\[
A = U \Sigma V^T,
\]

where \( U \) is an \( m \times n \) matrix with orthonormal columns, \( \Sigma \) is an \( n \times n \) nonnegative diagonal matrix and the \( n \times n \) matrix \( V \) is orthogonal. This decomposition has many important scientific and engineering applications (cf. [1], [5] and [11]), for some of which real-time computation is desirable. With the advent of VLSI technology, there is interest in a systolic network for computing the decomposition.

In [3] Finn et al. describe one such architecture and two related algorithms. Unfortunately, the convergence of their algorithms is only an empirical observation and has not been proved. Heller and Ipsen [7] consider the problem of computing only the singular values of a banded matrix with bandwidth \( w \). They present a systolic array of \( O(w) \) processors and an \( O(wn^2) \) algorithm.

In this paper we present a systolic array of \( O(n) \) linearly-connected processors which computes the SVD in time \( O(mn) \).

Alternatively, a two-dimensional systolic array of \( O(mn) \) processors with a more complicated interconnection pattern can compute the SVD in time \( O(n \log m) \). Our systolic architectures implement a one-sided orthogonalization method due to Hestenes [8].

The Hestenes method is essentially the serial Jacobi procedure for finding an eigenvalue decomposition of the matrix \( A^T A \), and has been used successfully by the second author [9] on the ILLIAC IV computer. A complication arises here in that, for the sake of parallel computing, we discard the classical scheme of rotating column pairs
in the order \((1,2),(1,3), \ldots, (1,n),(2,3), \ldots, (2,n),(3,4), \ldots, (n-1,n)\). To enforce convergence, we choose a threshold approach (see Wilkinson [13, pp. 277-278]).

2. Hestenes method

The idea is to generate an orthogonal matrix \(V\) such that the transformed matrix \(AV\) has orthogonal columns. Normalizing the euclidean length of each nonnull column to unity, we get the relation

\[
(2.1) \quad AV = \tilde{U} \Sigma ,
\]

where \(\tilde{U}\) is a matrix whose nonnull columns form an orthonormal set of vectors and \(\Sigma\) is a nonnegative diagonal matrix. A SVD of \(A\) is thus given by

\[
(1.1') \quad A = \tilde{U} \Sigma V^T.
\]

As a null column of \(\tilde{U}\) is always associated with a zero diagonal element of \(\Sigma\), there is no essential difference between (1.1) and (1.1').

Hestenes [8] suggests that the orthogonal matrix \(V\) should be constructed as a sequence of plane rotations, say \(Q_1,Q_2, \ldots\). Let

\[
(2.2) \quad A^c \equiv A Q_1 Q_2 \ldots Q_k.
\]

Without loss of generality, we may assume that the next rotation acts on the \(i\)-th and \(j\)-th columns (\(a_i^c\) and \(a_j^c\), \(i < j\)) of the current matrix \(A^c\). Let us consider

\[
(2.3) \quad (a_i^c, a_j^c) \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \equiv (\hat{a}_i^N, \hat{a}_j^N).
\]

Our problem is therefore choosing a rotation angle \(\theta\) to generate an orthogonal pair of column vectors \(\hat{a}_i^N\) and \(\hat{a}_j^N\). We use the formulas given by Rutishauser [10]. Defining
\[ (2.4) \quad \alpha \equiv \|a_i^c\|_2^2, \quad \beta \equiv \|a_j^c\|_2^2 \quad \text{and} \quad \gamma \equiv (a_i^c)^T(a_j^c), \]

we set \( \theta = 0 \) if \( \gamma = 0 \), and otherwise compute

\[ \xi = \frac{\beta - \alpha}{2\gamma}, \]

\[ t = \frac{\text{sign} (\xi)}{|\xi| + \sqrt{1 + \xi^2}}, \]

\[ (2.5) \quad \cos \theta = \frac{1}{\sqrt{1 + t^2}}, \]

and \( \sin \theta = t \cos \theta \).

The rotation angle always satisfies

\[ (2.6) \quad |\theta| \leq \frac{\pi}{4}, \]

which would guarantee convergence had we rotated the columns in the classical order of \((1,2),(1,3), \ldots, (1,n),(2,3), \ldots, (2,n),(3,4), \ldots, (n-1,n)\) (Forsythe and Henrici [4]). However, in order to perform the rotations in parallel, we choose a scheme (to be described in Section 4) in which all column pairs are rotated just once in any sequence (called a "sweep") of \( n(n-1)/2 \) rotations, but not in the above order. Very little is known about the convergence properties of this alternate scheme (see Hansen [6]), although we note that it has an asymptotically optimal "preference factor" [6, eqn. (23)].

To enforce convergence, we adopt a threshold approach (Wilkinson [13, pp. 277-278]). Let us associate with each sweep a threshold value, and when making the transformations of that sweep, we omit any rotation based on a normalized inner product

\[ \frac{(a_i^c)^T(a_j^c)}{\|a_i^c\|_2 \|a_j^c\|_2} \]
which is below the threshold value. Our method enjoys ultimate quadratic convergence (Wilkinson [12]) and numerical experience suggests that only six to ten sweeps are required (Rutishauser [10]).

We may compute the matrix $V$ by accumulating the plane rotations. If we let

$$ V^c = Q_1 Q_2 \cdots Q_k \equiv (y_1^c, \ldots, y_n^c), $$

then the $(k+1)$-st rotation will affect only the columns $y_1^c$ and $y_j^c$. We can therefore determine the new columns using

$$
\begin{pmatrix}
\begin{bmatrix}
A_1^c & A_j^c \\
y_1^c & y_j^c
\end{bmatrix} 
\begin{pmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{pmatrix}
= \begin{bmatrix}
A_1^c & A_j^c \\
y_1^c & y_j^c
\end{bmatrix}
\end{pmatrix},
$$

i.e., we update both $A^c$ and $V^c$ simultaneously.

3. **Generation of all pairs $(i,j)$**

In this section we show how $O(n)$ linearly-connected processors can generate all pairs $(i,j)$, $1 \leq i < j \leq n$, in $O(n)$ steps. The application to the computation of the SVD is described in Sections 4 and 5.

First suppose $n$ is even. We use $n/2$ processors $P_1, \ldots, P_{n/2}$, where $P_k$ and $P_{k+1}$ communicate ($k=1,2,\ldots,n/2-1$). Each processor $P_k$ has registers $L_k$ and $R_k$, output lines $\text{out}_{L_k}$ and $\text{out}_{R_k}$, and input lines $\text{in}_{L_k}$ and $\text{in}_{R_k}$, except that $\text{out}_{L_1}$, $\text{in}_{L_1}$, $\text{out}_{R_{n/2}}$ and $\text{in}_{R_{n/2}}$ are omitted. The output $\text{out}_{R_k}$ is connected to the input $\text{in}_{L_{k+1}}$ and the output $\text{out}_{L_{k+1}}$ is connected to the input $\text{in}_{R_k}$, as shown in Figure 1.
Initially $L_k = 2k - 1$ and $R_k = 2k$. At each time step processor $P_k$ executes the following program:

1. If $L_k < R_k$ then process $(L_k, R_k)$ else process $(R_k, L_k)$;
2. If $k=1$ then $\text{out}_{L_k} := R_k$
   else if $k < n/2$ then $\text{out}_{R_k} := L_k$;
3. If $k > 1$ then $\text{out}_{L_k} := R_k$;
4. (wait for outputs to propagate to inputs of adjacent processors)
   If $k < n/2$ then $R_k := \text{in}_{R_k}$ else $R_k := L_k$;
5. If $k > 1$ then $L_k := \text{in}_{L_k}$;

Here process $(i,j)$ means perform whatever operations are required on the pair $(i,j)$, $1 \leq i < j \leq n$. The operation of the systolic array is illustrated in Figure 2.

We see that the index 1 stays in the register $L_1$ of processor $P_1$. Indices 2, ..., $n$ travel through a cycle of length $n-1$ consisting of the registers $L_2, L_3, \ldots, L_{n/2}, R_{n/2}, R_{n/2-1}, \ldots, R_1$. During any $n-1$ consecutive steps a pair $(i,j)$ or $(j,i)$ can appear in a register pair $(L_k, R_k)$ at most once. A parity argument shows that $(i,j)$ and $(j,i)$ cannot both occur (see Figure 2). Since there are $n/2$ register pairs at each of $n-1$ time steps, each possible pair $(i,j)$, $1 \leq i < j \leq n$, is processed exactly once during a cycle of $n-1$ consecutive steps.
Figure 2: Full cycle of the systolic array for $n = 8$

If $n$ is odd, we use $\lceil n/2 \rceil$ processors but initialize

$L_k = 2k - 2$, $R_k = 2k - 1$ for $k = 1, \ldots, \lceil n/2 \rceil$ and omit any "process" calls from processor $P_1$.

4. A one-dimensional systolic array for SVD computation

Assume that $n$ is even (else we can add a zero column to $A$ or modify the algorithm as described at the end of Section 3). We use $n/2$ processors $P_1, \ldots, P_{n/2}$, as described in Section 3, except that $L_k$ and $R_k$ are now local memories large enough to store a column of $A$ (i.e. $L_k$ and $R_k$ each have at least $m$ floating-point
words). Shift registers or other sequential access memories are sufficient as we do not need random access to the elements of each row.

Suppose processor $P_k$ contains column $A^c_i$ in $L_k$ and column $A^c_j$ in $R_k$. It is clear that $P_k$ can implement the transformation defined by (2.3)-(2.6) in time $O(m)$ by making one pass through $A^c_i$ and $A^c_j$ to compute the inner products (2.4), and another pass to perform the transformations (2.3) or (2.8). Adjacent processors can then exchange columns in the same way that the processors of Section 3 exchange indices. This takes time $O(m)$ if the bandwidth between adjacent processors is one floating-point word. (Alternatively, exchanges can be combined with the transformations (2.3) or (2.8).)

Consequently, we see that $n/2$ processors can perform a full sweep of the Hestenes method in $n - 1$ steps of time $O(m)$, i.e. in total time $O(mn)$. Initialization requires that the $(2k-1)$-th and $2k$-th columns of $A$ be stored in the local memory of processor $P_k$ for $k=1, \ldots, n/2$; clearly this can also be performed in time $O(mn)$.

Although the process is iterative, in practice only 6-10 sweeps are required to orthogonalize the columns to full machine accuracy (see Section 2). Hence, we have a systolic array of $n/2$ processors which computes the SVD in time $O(mn)$. By comparison, the serial Hestenes algorithm takes time $O(mn^2)$.

After an integral number of sweeps the columns of the matrix $AV = \tilde{U}\Sigma$ (see (2.1)) will be stored in the systolic array (two per processor). If $V$ is required, it can be accumulated at the same time that $AV$ is accumulated, at the expense of increasing each processor's local memory (but the computation time remains $O(mn)$): see (2.8).
5. **Two-dimensional systolic arrays for SVD computation**

It is natural to ask if a rectangular array of $m$ by $n/2$ processors can be used to obtain a greater speedup than a linear array of $n/2$ processors. The only difficulty (but a major one) is in the fast accumulation of the inner products (2.4). If the processors have a binary tree interconnection pattern superimposed on each column of the rectangular array, so that the inner products (2.4) can be accumulated in time $O(\log m)$, then an array of $mn/2$ processors can compute the SVD of $A$ in time $O(n \log m)$, by an obvious modification of the method of Section 4.

As observed in Section 1, Hestenes method is theoretically equivalent (with exact arithmetic) to the Jacobi method applied to $A^T A$. In [2] we show that Jacobi's method can be implemented in time $O(n)$ on a square array of $n/2$ by $n/2$ systolic processors with nearest-neighbour connections. Hence, the factor $\log m$ above (and the non-local connection pattern) can be avoided, at the expense of some loss of numerical accuracy.
References


