Large Scale Eigenvalue Problems
J. Cullum and R.A. Willoughby (Editors)

COMPUTING THE SINGULAR VALUE DECOMPOSITION ON A RING OF ARRAY PROCESSORS

Christian Bischof
Charles Van Loan

Department of Computer Science
Cornell University
Ithaca, New York 14853

The parallel matrix computation community has lately devoted considerable attention to the Jacobi family of methods for computing eigenvalues and singular values. These methods "map" rather neatly onto various nearest neighbor architectures. If the processors are reasonably powerful and have significant local memory then block Jacobi procedures are appealing because they render a more favorable computation/communication ratio.

We examined the scope of these claims by implementing a block Jacobi SVD procedure on the IBM Kingston Loosely Coupled Array Processor (LCAP) system. The LCAP system consists of ten FPS-164/MAX array processors connected in a ring via some large bulk memories. Our basic finding is that the algorithm is well-suited to the architecture but that its advantage over a single processor implementation of the Golub-Reinsch procedure is rather unclear.

1. BLOCK JACOBI SVD

The singular value decomposition (SVD) of a matrix $A \in R^{m \times n}$ $(m \geq n)$ has many important applications. In the SVD, real orthogonal matrices U $(m \times m)$ and V $(n \times n)$ are sought such that $U^T A V = \text{diag}(\sigma_1, ..., \sigma_n)$. See Golub and Van Loan [1983]. In this paper we report on an implementation of a block generalization of the parallel Jacobi scheme in Brent, Luk, and Van Loan [1985].

Assume for simplicity that we have a partitioning of the form

$$(1.1) \qquad A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ A_{21} & A_{22} & \cdots & A_{2k} \\ \vdots & \vdots & & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{bmatrix}, \qquad A_{ij} \in R^{m_i \times n_j}$$

wher $m_i \geq n_j$ for all i and j.

Jacobi procedures proceed by making A increasingly diagonal by solving a judicious sequence of (i,j) subproblems. Solving the (i,j) subproblem means finding orthogonal U_0 and V_0 (of appropriate dimension) so that if

$$(1.2) \qquad U_0^T \begin{bmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{bmatrix} V_0 = \begin{bmatrix} B_{ii} & B_{ij} \\ B_{ji} & B_{jj} \end{bmatrix}$$

then

$$\| B_{ij} \|_F^2 + \| B_{ji} \|_F^2 \leq \mu^2 [\| A_{ij} \|_F^2 + \| A_{ji} \|_F^2]$$

where $\mu \in [0, 1)$ is called the "subproblem parameter". It dictates how much the subproblem is diagonalized. When $\mu = 0$, a full SVD is computed.

Another aspect of the subproblem solution concerns the "threshold". Subproblem (i,j) is "skipped" if A_{ij} and A_{ji} are too small, according to a threshold parameter $\tau > 0$, i.e., if

$$\| A_{ij} \|_F^2 + \| A_{ji} \|_F^2 < \tau^2 \| A \|_F^2 .$$

This manuever is critical for formal convergence.

Once the (i,j) subproblem is solved, the matrices U_0 and V_0 are appropriately applied to A:

$$(1.3) \qquad A \leftarrow J(i, j, U_0)^T A \, J(i, j, V_0)$$

The J matrices are block Jacobi rotations in block planes i and j. To illustrate the notation suppose that

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}$$

is orthogonal, and Q_{11} and Q_{22} are square. J(i,j,Q) designates the k-by-k block orthogonal matrix that is the identity with Q inserted in block positions (i,i), (i,j), (j,i), and (j,j), e.q.,

$$J(2,4,Q) = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & Q_{11} & 0 & Q_{12} \\ 0 & 0 & I & 0 \\ 0 & Q_{21} & 0 & Q_{22} \end{bmatrix} \qquad (k = 4).$$

With each update the matrix A becomes more block diagonal in the sense that the Frobenius norm of the off-diagonal blocks,

$$OFF(A)^2 = \sum_{i \neq j} \| A_{ij} \|_F^2.$$

is reduced. In particular, it can be shown that

$$OFF(J(i,j, U_0)^T A\, J(i,j, V_0))^2 \leq OFF(A)^2 - (1 - \mu^2) [\, \| A_{ij} \|_F^2 + \| A_{ji} \|_F^2 \,].$$

An important feature of any Jacobi procedure concerns the order in which the subproblems are solved. One approach is to specify an ordering

$$\Omega : \ (i_1, j_1), \ \ldots, (i_K, j_K) \qquad K = k(k-1)/2$$

of the set $\{(i,j) \mid 1 \leq i < j \leq k\}$ and choose the subproblems by cycling through Ω repeatedly until OFF(A) is small enough. Well-known is the row ordering, i.e., (1,2),(1,3),...,(1,k),(2,3),...,(2,k),...,(k-1,k). We will be more concerned with something called the "parallel" ordering which we discuss in the next section.

With these concepts and notations we have

Algorithm 1.1 (Generic Block Jacobi SVD)

Suppose $A \in R^{m\times n}$ is partitioned according to (1.1) and that $(i_1, j_1), \ldots, (i_K, j_K)$ is an enumeration of the set $\{(i,j) \mid 1 \le i < j \le k\}$. Given termination criteria $\varepsilon > 0$, positive threshold parameter τ that satisfies $\tau < \varepsilon \| A \|_F / \sqrt{K}$, and subproblem parameter $\mu \in [0, 1)$, the following algorithm computes orthogonal matrices U $(m \times m)$ and V $(n \times n)$ such that $OFF(U^T AV) \le \varepsilon \| A \|_F$. A is overwritten by $U^T AV$.

Algorithm 1.1

$U \leftarrow I_m$, $V \leftarrow I_n$
While $(OFF(A) > \varepsilon \| A \|_F)$
  For $r = 1 : K$
    $(i,j) \leftarrow (i_r, j_r)$
    If $(\| A_{ij} \|_F^2 + \| A_{ji} \|_F^2 \ge \tau^2 \| A \|_F^2)$ **then**
      Generate U_0 and V_0 by solving subproblem (i,j).
    else
      $U_0 \leftarrow I$, $V_0 \leftarrow I$.
    endif
    Set $J_1 = J(i,j, U_0)$ and $J_2 = J(i,j, V_0)$ and performs updates
      $A \leftarrow J_1^T A$, $A \leftarrow AJ_2$, $U \leftarrow UJ_1$, and $V \leftarrow VJ_2$
  end
end

The body of the while-loop is called a ***block sweep***. Normally, a few block sweeps are required for termination. An easy proof of convergence may be found in Van Loan [1985].

Jacobi methods, particularly for the symmetric eigenvalue problem, have a long history. See Jacobi [1846], Henrici [1958], Hansen [1962], Schonhage [1964], and Rutishauser [1966]. Further discussion and references appear in Golub and Van Loan [1983, p.295ff]. Block Jacobi methods are analyzed in Hansen [1960].

In Section 2 we develop a parallel version of Algorithm 1.1. The practical details associated with this parallel scheme and our implementation experiences on the IBM Kingston LCAP system are detailed in Sections 3 and 4. Some general remarks are offered in Section 5.

We mention in passing that just about everything in this paper applies to the symmetric eigenvalue problem -- just assume in the sequel that $A = A^T$ and identify the matrix U with V.

2. BLOCK JACOBI SVD WITH PARALLEL ORDERING

A parallel version of Algorithm 1.1 can be obtained if "nonconflicting" subproblems are solved concurrently. To illustrate what we mean by nonconflicting, assume that k = 8 and that we have four processors P_1, ..., P_4 with A apportioned among them as follows:

P_1	P_2	P_3	P_4
A_{11} A_{12}	A_{13} A_{14}	A_{15} A_{16}	A_{13} A_{14}
A_{21} A_{22}	A_{23} A_{24}	A_{25} A_{26}	A_{23} A_{28}
A_{31} A_{32}	A_{33} A_{34}	A_{35} A_{36}	A_{37} A_{38}
A_{41} A_{42}	A_{43} A_{44}	A_{45} A_{46}	A_{47} A_{48}
A_{51} A_{52}	A_{53} A_{54}	A_{55} A_{56}	A_{57} A_{58}
A_{61} A_{62}	A_{63} A_{64}	A_{65} A_{66}	A_{67} A_{68}
A_{71} A_{72}	A_{73} A_{74}	A_{75} A_{76}	A_{77} A_{78}
A_{81} A_{82}	A_{83} A_{84}	A_{85} A_{86}	A_{87} A_{88}

Note that processor i can generate the U_0 and V_0 associated with subproblem (2i-1,2i). Ignoring communication details for the moment, assume that A is updated by these 4 pairs of orthogonal matrices. What do we do next?

This is where the parallel ordering comes in. Partition the set of 28 off-diagonal index pairs into seven ***rotation sets*** as follows:

I.	(1,2)	(3,4)	(5,6)	(7,8)
II.	(1,4)	(2,6)	(3,8)	(5,7)
III.	(1,6)	(4,8)	(2,7)	(3,5)
IV.	(1,8)	(6,7)	(4,5)	(2,3)
V.	(1,7)	(5,8)	(3,6)	(2,4)
VI.	(1,5)	(3,7)	(2,8)	(4,6)
VII.	(1,3)	(2,5)	(4,7)	(6,8)

Read left to right, top to bottom, the above is an instance of the parallel ordering. This ordering can be easily derived by imagining a chess tournament among 8 players in which each players plays every other player exactly once. In between rounds (rotation sets) the players (block columns) move to adjacent tables (processors) in musical chair fashion:

Round I :	1	3	5	7
	2	4	6	8
Round II :	1	2	3	5
	4	6	8	7
Round III:	1	4	2	3
	6	8	7	5

etc.

To ready the processor array for subproblems (1,4), (2,6), (3,8), and (5,7) , we must reapportion A as follows:

P_1	P_2	P_3	P_4
A_{11} A_{14}	A_{12} A_{16}	A_{13} A_{18}	A_{15} A_{17}
A_{41} A_{44}	A_{42} A_{46}	A_{43} A_{48}	A_{45} A_{47}
A_{21} A_{24}	A_{22} A_{26}	A_{23} A_{28}	A_{25} A_{27}
A_{61} A_{64}	A_{62} A_{66}	A_{63} A_{68}	A_{65} A_{67}
A_{31} A_{34}	A_{32} A_{36}	A_{33} A_{38}	A_{35} A_{37}
A_{81} A_{84}	A_{82} A_{86}	A_{83} A_{88}	A_{85} A_{87}
A_{51} A_{54}	A_{52} A_{56}	A_{53} A_{58}	A_{55} A_{57}
A_{71} A_{74}	A_{72} A_{76}	A_{73} A_{78}	A_{75} A_{77}

Notice that solving the "current" (1,2), (3,4), (5,6), and (7,8) problems is equivalent to solving the (1,4), (2,6), (3,8), and (5,7) subproblems of the unpermuted A -- precisely what we are supposed to do.

Now for some generality. Suppose k is even and that we have $N = k/2$ processors $P_1, \ldots, P_N$. Let

$$A = [A_1, \ldots, A_k] \qquad A_i \in R^{m \times n_i}$$

be a column partitioning of A. If

$$I_n = [E_1, E_2, \ldots, E_k] \qquad E_i \in R^{n \times p}, \quad n = kp$$

is a conformable block column partitioning of the n-by-n identity, and we define the permutation

$$\Pi_n = [E_1, E_4, E_2, E_6, E_3, E_8, \ldots, E_{k-5}, E_k, E_{k-3}, E_{k-1}]$$

then $A\Pi_n$ permutes the block columns of A according to the parallel ordering. Let Π_m be the corresponding m-by-m permutation with the property that $\Pi_m^T A$ permutes the block rows of A according to the parallel ordering. With this notation we see that we can proceed from rotation set to rotation set on our processor array by performing the update

$$A \Leftarrow \Pi_m^T A \Pi_n$$

where the symbol $\Leftarrow$ means "assign block columns 2i-1 and 2i of the matrix prescribed by the right hand side to processor P_i.

We can now specify a parallel block Jacobi procedure. The algorithm is " column oriented" and it will be handy to let $U = [U_1, ..., U_k]$ and $V = [V_1, ..., V_k]$ be block column partitionings of U and V with $U_i \in R^{m \times m_i}$ and $V_i \in R^{n \times n_i}$ for $i = 1$.

Suppose $A = [A_1, ..., A_k]$, $U = [U_1, ..., U_k] = I_n$, and $V = [V_1, ..., V_k] = I_n$. Assume that we have $N = k/2$ processors and that processor i contains block columns 2i-1 and 2i of A, U, and V. Given $\varepsilon > 0$ the following algorithm computes orthogonal U and V such that $OFF(U^TAV) \leq \varepsilon \| A \|_F$.

Algorithm 2.1

While $(OFF(A) > \varepsilon \| A \|_F)$.

For rotation.set $= 1 : k-1$

Processor P_i $(i = 1:N)$ does the following:

Solves subproblem (2i-1,2i) thereby generating orthogonal U(i) and V(i).

Broadcasts U(i) to $P_1, ..., P_{i-1}, P_{i+1}, ..., P_N$.

Receives $U(1), ..., U(i-1), U(i+1), ..., U(N)$.

Performs the updates:

$$[A_{2i-1}, A_{2i}] \leftarrow \text{diag}(U(1),...,U(N))^T [A_{2i-1}, A_{2i}]V(i)$$

$$[V_{2i-1}, V_{2i}] \leftarrow [V_{2i-1}, V_{2i}]V(i)$$

$$[U_{2i-1}, U_{2i}] \leftarrow [U_{2i-1}, U_{2i}]U(i)$$

$$[A_{2i-1}, A_{2i}] \leftarrow \Pi_m^T [A_{2i-1}, A_{2i}]$$

Global communications:

$$A \Leftarrow AP$$

$$U \Leftarrow UP$$

$$V \Leftarrow VP$$

end

end

It is assumed that a threshold criteria τ and a subproblem parameter μ are part of the subproblem solution procedure.

With this breezy development of the parallel block Jacobi SVD algorithm, we are ready to look at some important practical details.

3. SOLVING THE SUBPROBLEMS

In the typical subproblem we are presented with a submatrix

$$A_0 = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} m_1 \\ m_2 \end{matrix}$$

$$\begin{matrix} n_1 & n_2 \end{matrix}$$

and must choose orthogonal U_0 and V_0 such that

$$U_0^T A_0 V_0 = B_0 = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

satisfies

$$\|B_{12}\|_F^2 + \|B_{21}\|_F^2 \leq \mu^2 [\|A_{12}\|_F^2 + \|A_{21}\|_F^2] \tag{3.1}$$

for some fixed $\mu < 1$. We mention two distinct approaches to this problem.

Method I. (Partial SVD via Row-Cyclic Jacobi)

Use the row cyclic Jacobi procedure (Algorithm 1.1) to compute U_0 and V_0 such that (3.1) holds. That is, keep sweeping until A_0 is sufficiently close to 2-by-2 block diagonal form. What is nice about this approach is that it can exploit the fact that the subproblems are increasingly diagonal as the overall iteration progresses. A_0 more diagonal implies fewer sweeps needed to satisfy (3.1). On the other hand, Jacobi requires square matrices and so A_0 may have to be padded with zero columns. There are some subtlties associated with this, see Brent, Luk, and Van Loan [1985].

Method 2. (Golub-Reinsch SVD with Bidiagonalization Pause)

Recall that the Golub-Reinsch algorithm begins with a bidiagonalization. After n_1 steps of this initial reduction we have

$$U_B^T A_0 V_B = \begin{bmatrix} x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \end{bmatrix} \quad (n_1 = m_1 = n_2 = m_2 = 4)$$

where U_B and V_B are products of Householder transformations. Note that the "b" entry is all that prevents the reduced matrix from being block diagonal. This suggests that if $| b |$ is small enough, then A_0 is sufficiently close to block diagonal form and we set $U_0 = U_B$ and $V_0 = V_B$. If $| b |$ is too large, we complete the bidiagonalization and proceed with the iterative portion of the Golub-Reinsch algorithm terminating as soon as the absolute value of the current $(n_1, n_1 + 1)$ entry is sufficiently small.

In contrast to Method 1, Method 2 can handle rectangular problems more gracefully. But note that in the rectangular case, diagonalization of A_0 does not correspond to block diagonalization:

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & x & 0 \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (m_1 = m_2 = 3, = n_1 = n_2 = 2)$$

The situation is remedied by some obvious permutations of U_0 and V_0.

After some experimentation we settled on the following subproblem procedure (illustrated by the $m_1 = m_2 = 4, n_1 = n_2 = 3$ case):

Subproblem Procedure

Step 1. Compute orthogonal Q so that

$$Q^T A_0 = \begin{bmatrix} x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This is a slight variation of the LINPACK QR factorization. For long rectangular problems Chan [1982] has demonstrated the advisability of preceding the SVD with a QR factorization. The reason for the "split" R has to do with obtaining close-to-identity U_0 and V_0 in the final stages of the iteration. We also mention that after the first rotation set is performed all subsequent subproblems have diagonal blocks, a fact that our split QR routine exploits.

Step 2. Glue the pieces of R together and compute the SVD.

$$U^T \begin{bmatrix} x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \end{bmatrix} V = \Sigma$$

Steps are taken to insure that U_0 and V_0 are close to the identity whenever that is possible through permutation.

Step 3. Form U_0 out of Q and U and form V_0 out of V.

It is perhaps worth dwelling on the need for close-to-identity transformations. This corresponds to the choosing of small rotation angles in the scalar case. A brief example serves to il-

lustrate why this is important. Suppose k=4, and that each block in A is 2-by-2. Suppose that A had the following form

$$\begin{bmatrix} X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon \\ X & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon \\ X & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X \end{bmatrix}$$

where ε denotes a small entry. Suppose that the U_0 and V_0 for subproblem (1,2) are close to the 4-by-4 identity but that in subproblem (3,4) $U_0 \approx V_0 \approx [e_3, e_4, e_1, e_2]$. It then follows that A gets updated to

$$\begin{bmatrix} X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ X & X & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon \\ X & X & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X \end{bmatrix}$$

To get ready for the next rotation set, which involves rotations (1,4) and (2,3), the block columns and rows of A are shuffled according to the parallel ordering $(A \leftarrow [E_1, E_3, E_4, E_2]^T A [E_1, E_3, E_4, E_2])$ giving

$$\begin{bmatrix} X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon & \varepsilon \\ X & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X & \varepsilon \\ X & X & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & X \end{bmatrix}$$

Thus the position of the non-neglible off-diagonal block remains fixed thereby slowing convergence immeasurably. (We stumbled into these observations when we encountered a fairly small example that required about 40 block sweeps to converge.)

To guard against this we have incorporated a low-overhead heuristic procedure in Step 3 that permutes all large entries in U and V to the diagonal.

4. IMPLEMENTATION ON THE IBM KINGSTON LCAP SYSTEM

The IBM Kingston Loosely Coupled Array Processor (LCAP) system consists of ten FPS-164/MAX array processors connected in a ring via five bulk memories manufactured by Scientific Computing Associates (SCA). Each bulk memory is attached to four array processors (AP's) and each AP is attached to a pair of bulk memories. This allows for considerable flexibility. For example, six AP's can be allocated to one user and four to another. Communication is via some message passing primitives provided by the SCA system. To the user these primitives look like calls to Fortran subroutines and are processed by a precompiler.

There are two types of communication required by Algorithm 2.1. Associated with each rotation set is a broadcast. Each AP must send an orthogonal matrix to every other AP in the ring. This is accomplished in merry-go-round fashion. The U(i) are passed around the ring (clockwise for example) and are applied to the housed block columns at each "stop". After the update, the A matrix must be redistributed around the ring. Here, the nearest neighbor topology is particularly convenient.

By "piggybacking" information on the U(i) as they circulate around the ring important global information such as OFF(A) and $\varepsilon \| A \|_F$ can be made available to each AP. This, of course, is critical in order to automate termination and the threshold testing.

We have run several examples and have gathered as much timing information as the LCAP system permits. Our results for a 636 × 96 example run on a 6-processor ring are fairly typical. Approximately 5 percent of the execution time was devoted to communication. Seven block sweeps were required and the overall performance rate was in the neighborhood of 3.6 Mflops. If is impossible to compare this with a single processor run as the problem would not fit into the fast memory of a single AP. However, it is clear that we are not achieving significant speed-up as the FPS-164 (without MAX boards) has a peak performance rating of 11 Mflops. (The AP's in the LCAP system are each scheduled to have two MAX boards, but these were not available at the time of our benchmarks.)

Since the block Jacobi procedure is rich in matrix-matrix multiplication, we expect more favorable performance when the LCAP MAX boards are in place. It is clear that communication costs diminish as problem size grows. When the m and n are several hundred, communication costs are quite insignificant.

5. CONCLUSIONS

The problem, as we see it, concerns the algorithm itself. In terms of the amount of arithmetic, two block sweeps is equivalent to one complete Golub-Reinsch algorithm. Thus, it is critical that the number of block sweeps be kept to a minimum. The problems we ran require between 6 and 10 block sweeps in order to reduce OFF(A) to a small multiple of $\varepsilon \| A \|_F$ where ε is the machine precision. Thus, we suspect that it will be difficult to implement a parallel block Jacobi SVD procedure that has a decided advantage over a single processor LINPACK SVD routine. The situation is even worse for scalar Jacobi procedures where the communication/computation ratio is less favorable.

The only way to rescue block Jacobi is with an ultrafast update procedure. Fortunately, updating is much more rich in matrix multiplication than the subproblem solution procedure. Thus,

our subsequent work on the LCAP system will involve making optimum use of the MAX boards which are tailored to the matrix multiplication problem.

Acknowledgements. We wish to thank Dr. Enrico Clementi of IBM Kingston who invited us to use the LCAP system. We are also indebted to his research group whose cooperation and friendliness made our experiments possible.

6. REFERENCES

[1] R. Brent and F. Luk (1985), The solution of singular value and symmetric eigenproblems on multiprocessor arrays, SIAM J. Scientific and Statistical Computing, **6**, 69-84.

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

[3] T. Chan (1982), An improved algorithm for computing the singular value decomposition, ACM Trans. Math. Software, **8**, 72-83.

[4] G. Forsythe and P. Henrici (1960), The cyclic Jacobi method for computing the principal values of a complex matrix, Trans. Amer. Math. Soc., **94**, 1-23.

[5] G. H. Golub and C. Van Loan (1983), ***Matrix Computations***, Johns Hopkins University Press, Baltimore, Md.

[6] E. Hansen (1960), On Jacobi methods and block-Jacobi methods for computing marix eigenvalues, Ph. D. Thesis, Stanford University, Stanford, Calif.

[7] E. Hansen (1962), On quasicyclic Jacobi methods, ACM J., **9**, 118-135.

[8] P. Henrici (1958), On the speed of convergence of cyclic and quasicyclic Jacobi methods for computing the eigenvalues of Hermitian matrices, SIAM J. Applied Math., 6, 144-162.

[9] C.G.J. Jacobi (1846), Uber ein leiches vehfahren die in der theorie der sacular-storungen vorkommendern gleichungen numerisch aufzulosen, Crelle's Journal, **30**, 51-94.

[10] H. Rutishauser (1966), The Jacobi method for real symmetric matrices, Numer. Math., **16**, 205-223.

[11] A. Schonhage (1964), On the quadratic convergence of the Jacobi process, Numer. Math., **6**, 410-412.

[12] C. Van Loan (1985), The block Jacobi method for computing the singluar value decomposition, Technical Report TR85-680, Department of Computer Science, Cornell University, Ithaca, NY 14853.