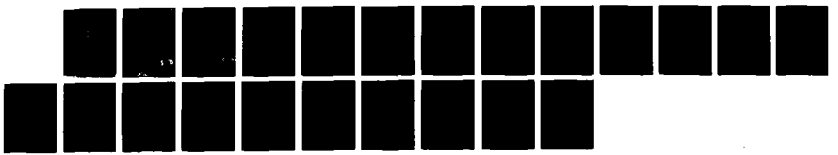
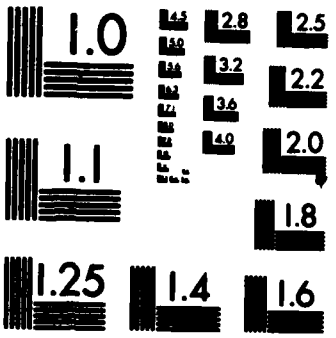
AD-A181 612 A PROCESSOR FOR TWO-DIMENSIONAL SYMMETRIC EIGENVALUE AND SINGULAR VALUE ARRAYS(U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE J DELOSME MAY 87 1/1

UNCLASSIFIED YALEU/DCS/RR-540 DAAL03-86-K-0158 F/G 12/2 NL





A processor for two-dimensional symmetric eigenvalue and singular value arrays

Jean-Marc Delosme

Research Report YALEU/DCS/RR-540
May 1987

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract. A dedicated processor architecture is here introduced for multiprocessor array implementations of Jacobi methods. Our new processing element factors arbitrary rotations into products of elementary rotations whose angles are exactly twice the CORDIC elementary angles. This characteristic permits complete concurrency between the evaluation of the Jacobi rotations and their application. Thus the processors in the resulting arrays are almost never idle.

Keywords: Symmetric eigenvalues, singular value decomposition, signal processing.

A processor for two-dimensional symmetric eigenvalue and singular value arrays

Jean-Marc Delosme

Research Report YALEU/DCS/RR-540
May 1987

The work presented in this paper was supported by the Army Research Office under Contract DAAL03-86-K-0158.

I. INTRODUCTION

Symmetric eigenvalue and singular value decompositions (SVD) are often at the core of today's signal processing techniques. Because these decompositions demand a large number of computations, and because of the stringent throughput requirements of adaptive beamforming, direction finding and other real-time signal processing applications that employ them, several parallel architectures tailored to their implementation have elsewhere been proposed. These architectures are briefly reviewed in [9]. A consequence of this burgeoning interest in simple dedicated parallel implementations has been the revival of the Jacobi methods, eclipsed for about twenty years by the computationally less demanding QR-factorization methods [7]. The Jacobi methods have the advantage of leading to highly regular implementations with local communications and simple control.

The Jacobi method is applied to a square $n \times n$ matrix, either the original matrix for the eigenvalue problem or the square factor computed in a preparatory QR-factorization step for the SVD problem [2]. In this paper the matrix is assumed to be real. (Note however that the Jacobi approach is very general and can also be used for parallel computation when the matrix is complex [6].) A particular ordering scheme for the rotations in the Jacobi approach allows for a high degree of parallelism, enabling the use of two-dimensional arrays of processors to implement a Jacobi sweep in $O(n)$ time [1], [2]. Experiments indicate that only $O(\log n)$ sweeps are needed for convergence, giving an $O(n \log n)$ total time for the diagonalization of the $n \times n$ matrix.

The multiprocessor implementation developed in [1] and [2] consists of a square array of $\frac{n}{2} \times \frac{n}{2}$ processors with nearest neighbor interconnections. Each sweep comprises a cyclic sequence of $n-1$ steps. At the start of a step the current matrix to be operated upon is already stored in a distributed fashion in

the processors of the array. The matrix is partitioned into 2×2 blocks, and adjacent blocks are assigned to adjacent processors. The step starts with the parallel evaluation by the diagonal processors of the rotations θ and θ' that diagonalize the 2×2 block stored in each of them

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} \cos\theta' & -\sin\theta' \\ \sin\theta' & \cos\theta' \end{pmatrix} = \begin{bmatrix} \bar{a} & 0 \\ 0 & \bar{d} \end{bmatrix} .$$

The angles θ and θ' are propagated to all processors in the same row and the same column, respectively, as the diagonal processor that has evaluated them. Every processor in the array pre- and post-multiplies the 2×2 block it currently has in storage by the rotations θ and θ' that it has received. Then, to complete the step, matrix elements are interchanged between adjacent processors in such a way, discovered in [1], as to enable a complete sweep in only $n-1$ steps.

The angles θ and θ' are easily seen to satisfy [2]

$$\tan(\theta + \theta') = \frac{b + c}{a - d} \quad \text{and} \quad \tan(\theta - \theta') = \frac{b - c}{a + d} .$$

The main role of the diagonal processors is to extract θ and θ' from these equations, while the function of the off-diagonal processors is simply to apply the rotations defined by these angles. In order to use the array efficiently, the diagonal processors should be as fast as the off-diagonal processors, even though they must perform a more complex task. The array efficiency will be further increased if the evaluation of the angles by the diagonal processors and the application of the rotations overlap significantly in time. The CORDIC concept is here fully exploited in order to meet these goals. In fact, according to the implementation here proposed, the evaluation and application of the rotations of a given step start and finish almost simultaneously. As a result the efficiency of the array is almost one, which is a significant improvement over the work presented in [3].

In the symmetric case the computational burden of the diagonal processors is reduced, since then c always equals b, which implies $\theta = \theta'$ and

$$\tan 2\theta = \frac{2b}{a-d} .$$

The array itself may furthermore be reduced to a triangle: in the square array, the blocks pertaining to a pair of processors which are positioned symmetrically with respect to the diagonal are always the transpose of each other. Since it is conceptually simpler, the implementation of this important special case will be presented first.

II. SYMMETRIC EIGENVALUE PROBLEM

Some elementary facts from CORDIC arithmetic [10], [11] will first be reviewed, as we shall build upon them in order to derive our implementations.

CORDIC arithmetic hinges on a fundamental property: if a matrix function $R(\theta)$ can be expressed as a matrix exponential $R(\theta) = e^{A\theta}$, an additive decomposition of θ yields a multiplicative decomposition of $R(\theta)$ [4], [5]. The off-diagonal processors in the symmetric eigenvalue array must apply plane rotations,

$$R(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} ,$$

which satisfy the above property, since

$$R(\theta) = e^{A\theta} \text{ with } A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} .$$

Consequently, assuming the availability of the encoding of θ as a set of signs $\gamma_i = \pm 1$ such that $\theta = \sum_{i=1}^{p} \gamma_i \theta_i$, where the positive angles θ_i form a preset sequence, the rotation of a vector by $R(\theta)$ can be performed by applying the sequence of elementary rotations $R(\gamma_i \theta_i)$ to that vector. It remains to specify

how the angles θ_i are selected to make the elementary rotations easy to perform.

Expressing the rotation $R(\gamma_i\,\theta_i)$ in terms of $t_i = \tan\theta_i$,

$$R(\gamma_i\,\theta_i) = (I + A\,\gamma_i\,t_i)/\sqrt{\det(I + A\,\gamma_i\,t_i)}\,,$$

and noting that the scaling factor is independent of γ_i,

$$\sqrt{\det(I + A\,\gamma_i\,t_i)} = \sqrt{1 + t_i^2} \triangleq S(t_i),$$

we see that the rotation $R(\theta) = \prod_{i=1}^{p} R(\gamma_i\,\theta_i)$ may be implemented by performing sequentially the product by the p factors $I + A\,\gamma_i\,t_i$ and then dividing the result by the global scaling factor $S = \prod_{i=1}^{p} S(t_i)$, [4], [11]. (Alternatively the scaling could be done first, followed by the p products by $I + A\,\gamma_i\,t_i$.) If the elementary angles θ_i are selected such that the tangents t_i are *integer powers of two*, multiplication of a vector by $I + A\,\gamma_i\,t_i$ is simply a pair of *coupled* addition/subtractions,

$$x_{i+1} = x_i + \gamma_i\,t_i\,y_i\,, \quad y_{i+1} = y_i - \gamma_i\,t_i\,x_i\,,$$

where the vector components x_i and y_i are shifted by t_i and then added or subtracted to y_i or x_i, respectively.

A high performance implementation of that operation will use two parallel adder/subtractors, one for each equation, and two barrel shifters, for quick multiplication by arbitrary integer powers of two. That architecture, with a minor modification, can also perform the division by the global scaling factor S. Indeed S^{-1} can be decomposed into a product $S^{-1} = \prod_{i=1}^{q}(1 + \delta_i\,s_i)$, where $\delta_i = \pm 1$ and each s_i is an integer power of two. Scaling of a vector by $1 + \delta_i\,s_i$ is simply a pair of *decoupled* addition/subtractions,

$$x_{i+1} = x_i + \delta_i\,s_i\,x_i\,, \quad y_{i+1} = y_i + \delta_i\,s_i\,y_i\,,$$

where the components x_i and y_i are shifted by s_i and then added or subtracted to x_i or y_i, respectively. Hence only a multiplexer has to be added to the basic architecture, to direct the shifted x_i towards the y-adder and the shifted y_i towards the x-adder (coupled mode) or to direct the shifted x_i towards the x-adder and the shifted y_i towards the y-adder (decoupled mode). (Note that, for compactness and speed, the multiplexer and the two barrel shifters could be merged into a single structure.)

The total time to perform a rotation is independent of the rotation angle and equals the time to perform p coupled pairs of addition/subtractions and q decoupled pairs of addition/subtractions. Since in the above implementation a pair of addition/subtractions, whether it is coupled or decoupled, requires the same amount of time - one cycle - the total time to perform a rotation is $p + q$ cycles. The tangents t_i should therefore be selected to minimize $p + q$ [4]. Assuming, without loss of generality, that the tangents t_i are non-increasing, that is $t_i \geq t_j$ for $1 \leq i \leq j \leq p$, any rotation within a range $(-\theta_{max}, +\theta_{max})$ can be implemented with accuracy ϵ if and only if $\theta_{max} \leq \sum_{j=1}^{p} \theta_j + \epsilon$ and

$$\theta_i \leq \sum_{j=i+1}^{p} \theta_j + \epsilon , \quad \text{for} \quad 1 \leq i \leq p .$$

These conditions, derived in [10], are easily seen to be satisfied if $t_{i+1} \geq t_i / 2$ for $1 \leq i < p-1$ and $t_p = \epsilon$. The freedom in the choice for t_{i+1} of either t_i or $t_i / 2$ may be exploited in order to minimize $p + q$: by properly choosing the tangent values to be repeated, we may modify the scaling factor S and reduce the number q of factors $1 + \delta_i s_i$ in the expansion of its reciprocal [4]. Given the desired range and accuracy, we may achieve that minimization via a search procedure such as simulated annealing [8]. For $\theta_{max} = \pi/4$, the range for the symmetric eigenvalue problem [1], and for the resolution $\epsilon = 2^{-32}$, our simulated

annealing program generated the sequences

$$t_i = 2^{-2}, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-4}, 2^{-5}, 2^{-6}, \ldots, 2^{-31}, 2^{-32},$$

$$\delta_i \, e_i = -2^{-3}, +2^{-4}, -2^{-15}, +2^{-16}, -2^{-18}, -2^{-22}, -2^{-24},$$

giving a rotation time of just 40 cycles. The control unit of each processor is programmed to cycle through such a sequence.

In the proposed architecture for the symmetric eigenvalue problem, the off-diagonal processors will each have two processing units that execute in parallel the standard CORDIC algorithm presented above, with minimal number of cycles $p+q$. In the first phase of each step, each processing unit rotates a column of the 2×2 block stored in the off-diagonal processor by the angle θ whose encoding is generated by the diagonal processor in the same row. In the second phase of the step, each processing unit now operates on a row of the modified 2×2 block and rotates it by the angle θ whose encoding has been generated in the first phase by the diagonal processor in the same column.

While the off-diagonal processors use the above standard CORDIC algorithm to apply rotations by angles θ encoded as sequences of signs γ_i, $1 \leq i \leq p$, the diagonal processors must on the other hand perform the more complex task of computing the encoding of the angle θ which satisfies $\tan 2\theta = 2b/(a-d)$, given the entries (a, b, d) of their current block. The first phase of the steps is critical, since, during that phase, rotations must be applied simultaneously with their evaluation. Such concurrency may be achieved if the encoding is generated recursively, that is, one sign γ_i every clock cycle. Indeed, although several cycles may be needed to propagate a bit along a row of the array, q cycles are available for that propagation if the off-diagonal processors spend the first q cycles of the first phase performing the scaling part of the rotations. For practical array sizes, this q-cycle upper bound on the propagation delay can be met without much

difficulty.

The encoding will be generated on line if the diagonal processors can perform rotations by $2\theta_i$ for $1 \leq i \leq p$ and, moreover, if the decomposition of 2θ as $\sum_{i=1}^{p} \gamma_i \cdot 2\theta_i$ is generated recursively. The rotation by $\gamma_i \cdot 2\theta_i$ is

$$R(\gamma_i \cdot 2\theta_i) = R(\gamma_i \theta_i) \cdot R(\gamma_i \theta_i) = \begin{bmatrix} 1-t_i^2 & \gamma_i \cdot 2t_i \\ -\gamma_i \cdot 2t_i & 1-t_i^2 \end{bmatrix} / (1 + t_i^2) \ ;$$

therefore, up to a scaling factor independent of the sign γ_i, it is a multiplication by

$$\begin{bmatrix} 1-t_i^2 & \gamma_i \cdot 2t_i \\ -\gamma_i \cdot 2t_i & 1-t_i^2 \end{bmatrix} .$$

Since the tangents t_i are integer powers of two, this multiplication may easily be implemented with shifts and addition/subtractions. Specifically, the operations are

$$x_{i+1} = (1 - t_i^2)x_i + \gamma_i \cdot 2t_i\, y_i \quad , \quad y_{i+1} = (1 - t_i^2)y_i - \gamma_i \cdot 2t_i\, x_i \quad .$$

These operations may be performed in the same amount of time as an off-diagonal processor cycle by a processor with four barrel shifters, two linear arrays of carry save adders (plus some inverters) and two adders. The barrel shifters are of two different types, in order to shift x_i and y_i in parallel by the two amounts $2t_i$ and t_i^2. The arrays of carry-save adders funnel data into the two adders. Specifically, they compress the triples $(x_i, -t_i^2 x_i, \gamma_i \cdot 2t_i\, y_i)$ and $(y_i, -t_i^2 y_i, -\gamma_i \cdot 2t_i\, x_i)$ into pairs of inputs for the two adders which compute, respectively, x_{i+1} and y_{i+1}. The carry-save adder delay is much smaller than the adder delay: hence the claim that a diagonal processor can perform the whole multiplication in one cycle. The signs γ_i can be generated recursively by using the simple control law [5]

$$\gamma_i = sign(x_i \cdot y_i) \ .$$

This law ensures that the rotation is always performed in the proper direction, so that, starting with $x_1 = (a - d)/2$ and $y_1 = b$, the absolute value of the angle between (x_i, y_i) and $(sign(a-d), 0)$ is always less than $\sum_{j=i}^{p} 2\theta_j + 2\epsilon$.

This novel processor architecture, introduced to implement the diagonal of the symmetric eigenvalue array, will also be used for the processors in the SVD array. The processor has the flexibility to perform either the coupled pair

$$x_+ = (1 - \alpha t^2)x + \gamma \cdot 2ty \ , \quad y_+ = (1 - \alpha t^2)y - \gamma \cdot 2tx \ ,$$

or the decoupled pair

$$x_+ = (1 + t^2)x + \beta\delta \cdot 2tx \ , \quad y_+ = (1 + t^2)y + \beta\delta \cdot 2ty \ ,$$

where α and β equal either 0 or 1, γ and δ equal either 1 or -1, and t can be any negative integer power of two between 2^{-2} and ϵ (note that $t = 2^{-1}$ is not included). The x and y registers may be loaded independently at the start of each cycle. The computation of $(a - d)/2$, which can be seen to require two cycles, and the computation of $a - d$ and $2b$, which requires four cycles because each computation needs two cycles and they cannot be performed concurrently, illustrate the limitations of the processor.

Assume for a moment that at the start of the first phase of a step the diagonal processors have already set their respective $x_1 = (a - d)/2$ and $y_1 = b$, and that the associated sign, γ_1, has just been sent to the off-diagonal processors in the corresponding row. We shall assume furthermore that γ_1 reaches all these processors before the end of cycle q. On the first cycle, the diagonal processors perform the (unscaled) rotation by $\gamma_1 \cdot 2\theta_1$ and evaluate γ_2, which is sent to the off-diagonal processors in the corresponding row. The last sign, γ_p, is evaluated on cycle $p-1$. The off-diagonal processors spend the first q cycles scaling the

vectors which they are rotating; by doing the scaling iterations at the beginning of the phase rather than at its end, q cycles are provided for the horizontal propagation of the signs γ_i. On cycle $q+1$ the off-diagonal processors apply from the left the (unscaled) rotation, by $\gamma_1\theta_1$, to their respective block. They apply the last rotation, by $\gamma_p\theta_p$, on cycle $p+q$. The diagonal processors initiate on cycle p the computation of the updates of the diagonal elements a and d, which are easily seen to satisfy

$$\bar{a} = \frac{a+d}{2} + [\frac{a-d}{2}\cos 2\theta + b\sin 2\theta]$$

$$\bar{d} = \frac{a+d}{2} - [\frac{a-d}{2}\cos 2\theta + b\sin 2\theta] .$$

To obtain the term between brackets, which is the first component of the vector $((a-d)/2, b)$ after its rotation by 2θ, the rotation of $((a-d)/2, b)$ is completed as follows. On cycle p apply the unscaled rotation by $\gamma_p \cdot 2\theta_p$. Then, for the next q cycles, perform the scaling part of the rotation, according to the decoupled equations

$$x_{i+1} = (1+s_i^2)x_i + \delta_i \cdot 2s_i x_i \ , \ \ y_{i+1} = (1+s_i^2)y_i + \delta_i \cdot 2s_i y_i \ .$$

The evaluation of the term between brackets thus terminates on cycle $p+q$.

In the second phase, every off-diagonal processor rotates its 2×2 block from the right by the angle θ whose encoding, issued on the first phase by the diagonal processor in the same column, has already been received. This requires $p+q$ cycles, like the first phase. The diagonal processors spend the first two cycles computing $(a+d)/2$, perform the addition that gives $\bar{a}$ on the next two cycles, and effect on the following two cycles the subtraction that yields $\bar{d}$. The remaining $p+q-6$ cycles provide, as we shall see presently, more than enough time to let the diagonal processors take the head start on the first phase of the step, as we earlier assumed.

In between two steps, the processors must exchange data. In contrast with the propagation of the single bits γ_i, which takes place along horizontal and vertical directions, this exchange is performed along diagonal connections. Each processor has the ability to exchange data with its four neighboring processors along the diagonal directions [1],[2]. Both diagonal elements within a block are exchanged with elements on the diagonals of those blocks which reside in the two neighboring processors along the same diagonal direction; the same holds for the other two elements in the block with respect to the other diagonal direction. In particular, the diagonal entries of the matrix are always exchanged between diagonal processors. The interchange itself may take a few cycles; it will require four cycles assuming 32-bit precision and one processor per VLSI chip, with 64 bidirectional I/O pins dedicated to this exchange. The diagonal processors, which finish updating the entries in their block about p cycles ahead of the off-diagonal processors, can exchange their diagonal elements with a p-cycle lead time. Thus are they afforded the time to evaluate the new $(a - d)/2$ before the start of the next step.

III. SINGULAR VALUE DECOMPOSITION

The array architecture is a square of $\frac{n}{2} \times \frac{n}{2}$ processors with horizontal and vertical connections, which carry to the off-diagonal processors the angle encodings computed by the diagonal processors, and with diagonal interconnections between neighboring processors, which convey the entries of the 2×2 blocks interchanged before each step. Both diagonal and off-diagonal processors consist of two processing units. The processing units are all identical, and can perform the same functions as a diagonal processor of the symmetric eigenvalue array.

During the first phase of a step, the diagonal processors rotate in parallel the vectors $(a - d, b + c)$ and $(a + d, b - c)$, in order to evaluate recursively the

encodings of $\theta^+ \triangleq \theta + \theta'$ and $\theta^- \triangleq \theta - \theta'$ as well as to update the diagonal elements within their respective blocks. The elementary angles for these rotations are $2\theta_i$, identical to the ones used by the diagonal processors in the eigenvalue array. In order to generate the encoding as soon as possible, the unscaled part of the rotations must be performed first. On the first two cycles the two processing units of a diagonal processor compute $b + c$ and $b - c$; the components $a - d$ and $a + d$ have already been computed during the previous step. At the end of the second cycle, the two processing units contain $(x_1^+, y_1^+) = (a - d, b + c)$ and $(x_1^-, y_1^-) = (a + d, b - c)$, and the signs $\gamma_1^+ = sign(x_1^+ \cdot y_1^+)$ and $\gamma_1^- = sign(x_1^- \cdot y_1^-)$ are generated. The operations effected in parallel in the two processing units on cycle $i+2$, $1 \leq i \leq p$, are

$$x_{i+1}^+ = (1 - t_i^2)x_i^+ + \gamma_i^+ \cdot 2t_i\, y_i^+ \quad , \quad y_{i+1}^+ = (1 - t_i^2)y_i^+ - \gamma_i^+ \cdot 2t_i\, x_i^+ \quad ,$$

$$\gamma_{i+1}^+ = sign(x_{i+1}^+ \cdot y_{i+1}^+) \quad ,$$

and

$$x_{i+1}^- = (1 - t_i^2)x_i^- + \gamma_i^- \cdot 2t_i\, y_i^- \quad , \quad y_{i+1}^- = (1 - t_i^2)y_i^- - \gamma_i^- \cdot 2t_i\, x_i^- \quad ,$$

$$\gamma_{i+1}^- = sign(x_{i+1}^- \cdot y_{i+1}^-) \quad .$$

The pairs of bits (γ_i^+, γ_i^-), $1 \leq i \leq p$, generated by a diagonal processor, make up the encoding of (θ^+, θ^-). They are sent via 2-bit wide horizontal and vertical connections to all the processors which are located either on the row or on the column of the diagonal processor. The last unscaled rotations, performed on cycle $p+2$, align the vectors $(a - d, b + c)$ and $(a + d, b - c)$ along the $(1, 0)$ direction to the best angular accuracy that a diagonal processor can achieve. While the pair of bits $(\gamma_{p+1}^+, \gamma_{p+1}^-)$ generated on that cycle is of no utility, the first components of the rotated vectors are useful for computing the updates $\bar{a}$ and $\bar{d}$ of the diagonal elements. Indeed, these components are the computed values, before scaling, of

$x^+ \triangleq (a - d)\cos\theta^+ + (b + c)\sin\theta^+$ and $x^- \triangleq (a + d)\cos\theta^- + (b - c)\sin\theta^-$,

and the updates of a and d are easily shown to satisfy

$$\bar{a} = \frac{x^- + x^+}{2} \ , \quad \bar{d} = \frac{x^- - x^+}{2} \ .$$

The scaling part of the rotations that yield x^+ and x^- begins on cycle $p+3$ and employs the decoupled equations

$$x_{i+1}^+ = (1 + s_i^2)x_i^+ + \delta_i \cdot 2s_i x_i^+ \ , \quad y_{i+1}^+ = (1 + s_i^2)y_i^+ + \delta_i \cdot 2s_i y_i^+ \ ,$$

$$x_{i+1}^- = (1 + s_i^2)x_i^- + \delta_i \cdot 2s_i x_i^- \ , \quad y_{i+1}^- = (1 + s_i^2)y_i^- + \delta_i \cdot 2s_i y_i^- \ .$$

Since the scaling requires q steps its execution will spill over onto the first two cycles of the second phase.

During the first phase the off-diagonal processors apply rotations by θ to the columns of their respective block. During the first q cycles of the phase each off-diagonal processor normalizes the lengths of the vectors it operates upon, thus giving the pairs (γ_i^+, γ_i^-), generated by the diagonal processor belonging to the same row, $q-2$ cycles to reach an off-diagonal processor. The significant, unscaled, part of the rotations by θ is applied on the next p cycles. The angle of the ith rotation increment is $(\gamma_i^+ + \gamma_i^-)\theta_i$, hence the total amount of rotation after p iterations is, as it should be, the half-sum of the decompositions of θ^+ and θ^- as $\sum_{i=1}^{p} \gamma_i^+ \cdot 2\theta_i$ and $\sum_{i=1}^{p} \gamma_i^- \cdot 2\theta_i$.

In order to be able to perform the scaling in q iterations, some care is needed when implementing the unscaled rotations. Observe that the amount of the ith unscaled rotation can be $+2\theta_i$, $-2\theta_i$ or simply 0. Since in the first two cases the rotation increases the length of the vector to which it is applied by $1 + t_i^2$, the nil rotation should increase the length by the same amount. A processing unit can effect this stretching in one cycle by multiplying both vector

components by $1 + t_i^2$. It is only because of this operation that the elementary angles for the rotations have been selected equal to $2\theta_i$ instead of θ_i, where $\tan\theta_i = t_i$ is an integer power of two. Indeed if the angles θ_i were selected as rotation increments, the vector components would have to be multiplied by $\sqrt{1 + t_i^2}$, which the CORDIC hardware cannot do in one cycle. (A related property is that $2\theta_p$ is the smallest rotation increment that may be applied in the proposed implementation, corresponding to an accuracy of 2ϵ. Thus, assuming an identical number of cycles per step, the accuracy of the Jacobi rotations is one bit less for the SVD array than for the eigenvalue array.)

We may now recapitulate more formally the operations performed by the off-diagonal processors during the first phase. The processors apply rotations by θ to the columns of their respective block. Each processing unit within a processor deals independently with one of the two columns. The q scaling iterations performed on a column are of the form

$$x_{i+1} = (1 + s_i^2)x_i + \delta_i \cdot 2s_i x_i \ , \quad y_{i+1} = (1 + s_i^2)y_i + \delta_i \cdot 2s_i y_i \ .$$

The p unscaled rotation iterations that follow satisfy

$$x_{i+1} = (1 - \sigma_i t_i^2)x_i + \gamma_i^+ \cdot (1 + \sigma_i)t_i y_i \ , \ y_{i+1} = (1 - \sigma_i t_i^2)y_i - \gamma_i^+ \cdot (1 + \sigma_i)t_i x_i \ ,$$

where $\sigma_i \triangleq sign(\gamma_i^+ \cdot \gamma_i^-)$ with γ_i^+ and γ_i^- received from the diagonal processor in the same row. Thus either a nil rotation is performed, if γ_i^+ and γ_i^- differ, or else a rotation by $\gamma_i^+ \cdot 2\theta_i$ is applied.

At the beginning of the second phase of the step, the diagonal processors terminate the scaling of the components x^+ and x^- and compute $\overline{a}$ and $\overline{d}$ from these components. This requires a total of only four cycles. Then the diagonal processors exchange their diagonal elements between neighbors and compute the new $a - d$ and $a + d$. Afterwards, they merely wait till the end of the phase for the off-diagonal processors to finish their rotations.

During the second phase, the off-diagonal processors apply rotations by θ' to the rows of their respective blocks. Each processing unit within a processor deals independently with one of the two rows. The q scaling iterations performed on a row are of the form

$$x_{i+1} = (1 + s_i^2)x_i + \delta_i \cdot 2s_i\, x_i \ , \quad y_{i+1} = (1 + s_i^2)y_i + \delta_i \cdot 2s_i\, y_i \ .$$

The p unscaled rotation iterations that follow satisfy

$$x_{i+1} = (1 + \sigma_i\, t_i^2)x_i + \gamma_i^+ \cdot (1 - \sigma_i)t_i\, y_i \ , \ y_{i+1} = (1 + \sigma_i\, t_i^2)y_i - \gamma_i^+ \cdot (1 - \sigma_i)t_i\, x_i \ ,$$

where $\sigma_i = sign\,(\gamma_i^+ \cdot \gamma_i^-)$ with γ_i^+ and γ_i^- received from the diagonal processor in the same column. Thus either a nil rotation is performed, if γ_i^+ and γ_i^- are equal, or else a rotation by $\gamma_i^+ \cdot 2\theta_i$ is applied. The second phase lasts $p+q$ cycles, like the first one.

The next few cycles are used to exchange entries between neighboring processors in exactly the same way as for the symmetric eigenvalue array. Note that the updated off-diagonal elements for the diagonal processors will be available only at the end of that exchange; they cannot be exchanged in advance like the diagonal elements.

IV. DISCUSSION

A key idea in our approach is to simplify the diagonal processors by having them generate simple rotation encodings. The diagonal processors of the symmetric eigenvalue array consist of one new processing element and those of the SVD array of two such processing elements. This new processing element is a fairly simple machine, only slightly more complex than a CORDIC rotor and requiring less than twice the CORDIC rotor area for the same cycle time. The simplification of the diagonal processors entails some extra cost for the off-diagonal processors, which must be able to construct rotations from the

encodings. If that extra cost were an increased processor complexity, the approach would be unpractical; there are many more off-diagonal processors than diagonal ones. Fortunately the CORDIC approach lets the tradeoff take place at another level: processor speed.

For 32-bit precision the diagonal processors proposed can be as fast if not faster (and certainly much more compact) than processors based on conventional approaches of evaluating the entries of the rotation matrix: using fast multiplication, division and square-root algorithms, or table look-ups and linear interpolations. The off-diagonal processors, however, are significantly slower than a fast multiplier based implementation of the rotation for about the same area. This relatively poor performance would doom the whole approach if it did not possess a key redeeming feature: the rotations may be applied simultaneously with respect to their evaluation. Indeed, while with a traditional processor architecture the application of the rotations cannot start before the complete evaluation of the entries of the rotation matrices, with the proposed architecture the application of the rotations by the off-diagonal processors starts at the same instant as their evaluation. The relative slowness of the off-diagonal processors contributes little to the total computation time: at the end of a step, the off-diagonal processors need about p cycles to terminate their computations after the diagonal processors have finished theirs. This time is to be compared to the time required, with a traditional processor architecture, for propagating the cosines and sines defining the rotations to the off-diagonal processors and then performing the rotations. These times are similar for practical pin counts, which impose a fair amount of sequentiality in propagating the sines and cosines. Thus we claim that our processor architecture leads to a performance similar to what could be obtained using a more conventional architecture based on fast arithmetic, while offering the advantages of simpler control and much simpler diagonal processors.

Two important architectural variants that use the CORDIC approach should be mentioned. They both apply to the SVD array. In the first variant the off-diagonal processors are replaced by simpler pairs of CORDIC rotors, as in the symmetric eigenvalue array. The diagonal processors, which must then generate the encodings for θ and θ' directly, employ the relations

$$\tan 2\theta = 2\frac{ab + cd}{a^2 + c^2 - b^2 - d^2}\ , \quad \tan 2\theta' = 2\frac{ac + bd}{a^2 + b^2 - c^2 - d^2}\ .$$

Thus, in addition to the two processing elements that generate the encodings for θ and θ' given the vectors $(a^2+c^2-b^2-d^2, 2(ab+cd))$ and $(a^2+b^2-c^2-d^2, 2(ac+bd))$, the diagonal processors possess fast multipliers for the speedy computation of the components of the vectors. The updates of the diagonal blocks are performed by pairs of CORDIC rotors, as for the off-diagonal blocks. The array is therefore a square of identical processors - pairs of CORDIC rotors - plus a diagonal of rather complex processors; it can be as fast as the original one if the time for the fast multiplications and the propagation of the first bit of the encoding of θ does not exceed q cycles. A second variant aims at removing the idle time that the diagonal processors experience during the second phase of each step. The diagonal processors are the same as in the first variant and generate the encodings for θ and θ' directly. A square array of identical processors applies the rotations to the diagonal and off-diagonal blocks simultaneously from the left and from the right. More precisely these processors apply the ith iteration of the rotation by θ and the rotation by θ' simultaneously, in effect removing the second phase of each step. The processors contain two processing units which, although they have more complex adders, are still very similar to the ones in the original SVD array. One processing unit operates on the diagonal entries of the current block and the other one on its off-diagonal entries. The speed up with respect to the original array can reach two, if the time spent for

the fast multiplications and the propagation of the first bit of the encodings of θ and θ' does not exceed q cycles.

One should be able to use the arrays, which are adapted to a specific problem size n, for the solution of oversized problems with only a marginal loss in efficiency. Instead of a block Jacobi scheme [2], a more appropriate procedure is to map the scalar Jacobi scheme on a virtual array whose dimensions match the problem size (more precisely match the first multiple of n greater or equal to the problem size) and then fold that array onto the real array. The folding is just like a paper folding into squares of the same size as the real array and defines the allocation of the 2×2 blocks of the matrix onto the real array. (Each processor could communicate with its own memory chip holding the blocks allocated to it.) The sequencing in time is such that, if we unfold our folded paper, the squares along the diagonal are processed first in order to determine the rotation angles. The exchanges between steps still involve only neighboring processors. The control remains simple and the efficiency very high.

ACKNOWLEDGEMENT

Jimmy Lam, from the Computer Science Department of Yale University, wrote the simulated annealing program mentioned in the paper.

REFERENCES

1 R.P. Brent and F.T. Luk, "The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays," *SIAM J. Sci. Statist. Comput.*, Vol. 6, pp. 69-84, Jan. 1985.

2 R.P. Brent, F.T. Luk and C. Van Loan, "Computation of the singular value decomposition using mesh connected processors," *J. VLSI Computer Systems*, Vol. 1, No. 3, pp. 242-270, 1985.

3 J.R. Cavallaro and F.T. Luk, "Architectures for a CORDIC SVD Processor," *Real Time Signal Processing IX, Proc. SPIE,* Vol. 698, Aug. 1986.

4 J.-M. Delosme, "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces," *Proc. 1983 IEEE Int. Conf. on ASSP,* Boston, MA, pp. 927-930, Apr. 1983.

5 J.-M. Delosme, "The matrix exponential approach to elementary operations," *Advanced Algorithms and Architectures for Signal Processing, Proc. SPIE,* Vol. 696, pp. 188-195, Aug. 1986.

6 P.J. Eberlein, "On the Schur Decomposition of a Matrix for Parallel Computation," *IEEE Trans. on Computers,* Vol. C-36, No. 2, pp. 167-174, Feb. 1987.

7 G.H. Golub and C.F. Van Loan, *Matrix Computations,* The Johns Hopkins University Press, 1983.

8 S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science,* Vol. 220, pp. 671-680, 1983.

9 J.M. Speiser, "Signal processing computational needs," *Advanced Algorithms and Architectures for Signal Processing, Proc. SPIE,* Vol. 696, pp. 2-6, Aug. 1986.

10 J.E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. on Electronic Computers*, Vol. EC-8, No. 3, pp. 330-334, Sept. 1959.

11 J.S. Walther, "A Unified Algorithm for Elementary Functions," *AFIPS Conf. Proc., 1971 Spring Joint Computer Conference*, Vol. 38, pp. 379-385, 1971.

END

7-87

DTIC