Proceedings of the
International Conference on

APPLICATION SPECIFIC ARRAY PROCESSORS

September 5-7, 1990 *Princeton, New Jersey*

Sponsored by
Princeton University

Co-Sponsored by
Industrial Development Board for Northern Ireland
NEC Research Institute

In Cooperation with
IEEE Computer Society

Edited by

Sun-Yuan Kung
Princeton University

Earl E. Swartzlander, Jr.
University of Texas at Austin

Jose A.B. Fortes
Purdue University

K. Wojtek Przytula
Hughes Research Laboratories

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and are published as presented and without change, in the interests of timely dissemination. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society Press, or The Institute of Electrical and Electronics Engineers, Inc.

Published by

IEEE Computer Society Press
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

Cover designed by Jack I. Ballestero

Printed in United States of America

IEEE Computer Society Press Order Number 2089
Library of Congress Number 90-82602
IEEE Catalog Number 90CH2920-7
ISBN 0-8186-9089-5 (case)
ISBN 0-8186-6089-9 (microfiche)
SAN 264-620X

Additional copies can be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama,
Minato-Ku
Tokyo 107, JAPAN

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331

THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

Table of Contents

Mapping Applications onto Architectures

Potpourri II

System Building Blocks

Special-Purpose Systems 2

Potpourri III

Systolic Arrays

BIT-LEVEL SYSTOLIC ALGORITHM FOR THE SYMMETRIC EIGENVALUE PROBLEM

JEAN-MARC DELOSME
Department of Electrical Engineering
Yale University

An arithmetic algorithm is presented which speeds up the parallel Jacobi method for the eigen-decomposition of real symmetric matrices. The matrices to which the plane Jacobi rotations are applied are decomposed into even and odd part, enabling the application of the rotations from a single side and thus removing some sequentiality from the original method. The rotations are evaluated and applied in a fully concurrent fashion with the help of an implicit CORDIC algorithm. In addition, the CORDIC algorithm can perform rotations with variable resolution, which lead to a significant reduction in the total computation time.

I. INTRODUCTION

The eigenvalue decomposition of a real symmetric matrix or the singular value decomposition (SVD) of an arbitrary real matrix may be obtained by the Jacobi method (Jacobi/Kogbetliantz). This method can be parallelized to a high degree [1], [2], resulting in a computation time that is approximately linear in the smallest dimension of the matrix. Furthermore the ratio of parallel to sequential hardware cost is of the same order as the gain in computation time. Thus, the parallel hardware is exercised with a fairly high efficiency, essentially independent of the matrix (smallest) dimension.

Although the Jacobi method requires more operations than the justly popular QR method (Francis/Golub and Kahan), a significantly higher degree of parallelism may be extracted from it, making it the method of choice for the fast diagonalization, via orthogonal transformations, of unstructured dense matrices. Our objective is to determine extremely fast ways of performing this diagonalization in the context of signal and image processing. This entails, starting from the Jacobi method and the parallelization scheme of Brent and Luk, the design of algorithms at a detailed level and the development of the associated, application-specific, array architectures. In this paper, after analyzing the elementary mathematical operations in the Jacobi method (*i.e.* the evaluation and application of Jacobi rotations), we devise arithmetic algorithms that effect these mathematical operations with few primitive operations (*i.e.* few shifts and adds) and enable the most efficient use of the parallel hardware. Moreover we modify the Jacobi algorithm in order to reduce the total number of primitive operations for achieving matrix diagonalization.

By targeting an implementation that is as fast as can be found, we are led to exploring and exploiting as much as possible the mathematical structure of the problem, hence to finding arithmetic algorithms better adapted to the problem at hand. Implementations which match lower data rates can then be generated by a fairly standard process of sequentialization. By considering the SVD problem, which may be viewed as a generalization of the symmetric eigenvalue problem, we first direct our search for structure toward fundamental objects and properties. The special features due to symmetry are exploited in a second phase. Our approach to algorithm design is thus hierarchical: first the main structure, then the refinements. This way we avoid focusing early on non-fundamental features and, as a result, being trapped in a local optimum. In fact, in order to uncover a global solution, we have embedded the problem into a further

CH2920-7/90/0000/0770$01.00 © 1990 IEEE

generalization: the SVD problem for *complex* matrices. Although this generalization is not discussed in this paper (it is presented in [6]), it did guide us in our search.

The parallel Jacobi algorithm of Brent and Luk is briefly described in Section II. This algorithm exhibits close to maximal parallelism and does it at a close to minimal communication cost, where 'close to' means 'up to a small constant multiplicative factor' [7]. It provides the starting point for the process of refinement, taking the above multiplicative factors closer to unity, that brings forth our array for the eigen-decomposition of real symmetric matrices. The Jacobi method is a succession of parallel steps, starting from an initial square matrix and converging to a diagonal matrix, in which plane rotations are applied on both sides of the 2×2 submatrices of the current iterate. In Section III a mathematical property, the existence of the decomposition of Clifford numbers into even and odd parts, is shown to enable the application of the rotations from the same side, thus leading to a parallel procedure for the evaluation and the application of the Jacobi rotations. While this procedure would cost many operations if the rotations were evaluated using the standard arithmetic operations, $\pm$, $\times$, $/$, $\surd$, it becomes cheap if CORDIC arithmetic, based on shifts and adds and reviewed in Section IV, is employed. Our 'implicit' CORDIC algorithm for the symmetric eigenvalue problem, which does not compute rotation angles explicitly, is presented in Section V. Evaluation and application of the rotations may be fully overlapped with this algorithm, a feat which cannot be achieved with an explicit CORDIC algorithm.

II. ARRAY ARCHITECTURE

The method of Jacobi, first applied to the eigen-decomposition of real symmetric matrices $B = U\Lambda U^T$, with U orthogonal and Λ real diagonal, was generalized by Kogbetliantz (1955) to the computation of the SVD of a real rectangular matrix $A = V\Sigma U^T$, where U and V have orthonormal columns and Σ is real diagonal with positive entries. We shall first expose the general case and then turn to the symmetric case, which can be viewed as a special case.

Without loss of generality, A may be assumed to have more rows than columns. By applying plane, Givens, rotations from the left, A may be decomposed into QB where Q has orthonormal columns and B is square. Thus the computation of the SVD of a rectangular matrix A reduces to the SVD computation of an associated square matrix B, and we can from now on only consider the decomposition of square matrices B.

Starting from a general real $n\times n$ matrix B, the Jacobi method performs a short sequence of sweeps to bring the matrix to diagonal form. In each sweep $n(n-1)/2$ pairs of plane rotations are applied to both sides of the matrix to annihilate each of the $n(n-1)$ off-diagonal elements once. Each pair of rotations may be represented by two $n\times n$ matrices, J_{ij}, applied to the matrix from the right, and J'_{ij}, applied to the matrix from the left, where the couple ij, with $1 \le i < j \le n$, is distinct for each pair in the sweep. Both rotation matrices differ from the identity matrix of order n by the principal submatrix formed at the intersection of the row and column pairs corresponding to i and j. These principal submatrices have the form

$$\begin{bmatrix} \cos\theta_{ij} & \sin\theta_{ij} \\ -\sin\theta_{ij} & \cos\theta_{ij} \end{bmatrix} \text{ for } J_{ij} \text{ and } \begin{bmatrix} \cos\theta'_{ij} & -\sin\theta'_{ij} \\ \sin\theta'_{ij} & \cos\theta'_{ij} \end{bmatrix} \text{ for } J'_{ij},$$

and the angles θ_{ij} and θ'_{ij} are selected to zero out simultaneously the ij-th and ji-th entry of the matrix to which J_{ij} and J'_{ij} are applied.

The simultaneous application of p non-conflicting pairs of rotations, zeroing out $2p$ entries of the matrix to which they are applied, is called a (parallel) step. For ease of presentation we shall assume that n is even and refer to [1] and [7] for n odd. Since any partition of the set $\{1, ..., n\}$ into pairs $\{i, j\}$ has $n/2$ parts, a maximally parallel step would apply $n/2$ pairs of rotations simultaneously. If a sweep is decomposed into a sequence of such steps, each forcing n matrix entries to 0, and if for all the steps in the same sweep the indices ij of the pairs of rotations are distinct, the number of steps in a sweep is minimal, equal to $n-1$. Such a scheme may be constructed by selecting a cyclic permutation, P, of the indices $\{2, 3, ..., n\}$. By partitioning into contiguous pairs the ordered concatenation $\{1\} \cup S$, where S is the *ordered* set $\{2, 3, ..., n\}$, a set of pairs, $\{12; 34; ...; n-1,n\}$, is obtained whose order is induced from the order $\{1\} \cup S$. These are the pairs of indices for the pairs of rotations applied in the first step of a sweep. Next the ordered concatenation $\{1\} \cup PS$ is partitioned into contiguous pairs, with order induced by the order $\{1\} \cup PS$, defining the pairs of indices for the second step. The order $\{1\} \cup P^2S$ defines the pairs of indices for the third step, and so on until $\{1\} \cup P^{n-2}S$ for the $(n-1)$-th step. The following order is $\{1\} \cup P^{n-1}S = \{1\} \cup S$; indeed this is the beginning of the next sweep. We shall index the pairs at a given step k by a single number I, $1 \leq I \leq n/2$; thus J_{ij} will alternately be written J_I and, in particular, J_{34} and J_2 represent the same matrix at step 1. Brent and Luk have selected the cyclic permutation

$$2 \rightarrow 3 \rightarrow 5 \cdots \rightarrow n-3 \rightarrow n-1 \rightarrow n \rightarrow n-2 \rightarrow \cdots 6 \rightarrow 4 \rightarrow 2,$$

which has the desirable property that the indices ij in the I-th pair at step k come from the *neighboring* pairs at step $k-1$, with indices $I-1$, I, or $I+1$.

The matrix B is transformed into a diagonal matrix through a sequence of steps, starting with $B_0 = B$ and computing at step k

$$B_k = \prod_{I=1}^{n/2} J_I' \, B_{k-1} \prod_{I=1}^{n/2} J_I \, .$$

Although this is not written explicitly, the two sets of rotations $\{J_I, 1 \leq I \leq n/2\}$ and $\{J_I', 1 \leq I \leq n/2\}$ depend on the step k. Moreover, since the rotations within each set are disjoint, each set of rotations is applied in parallel. At a high level, the scheme of Brent and Luk leads directly to a parallel architecture for the SVD, taking the form of a square array with $n/2$ processors on a side. Processor IJ holds at the beginning of step k the 2×2 submatrix of B_{k-1} sitting at the intersection of rows i and j and of columns r and s, where ij and rs are respectively the I-th and J-th pairs of indices at step k. Each diagonal processor, such as processor II or processor JJ, evaluates the two plane rotations, J_I and J_I' or J_J and J_J', that zero out the two off-diagonal entries of the submatrix it holds, and updates accordingly the two diagonal entries. From each diagonal processor a representation of each of the two rotations is sent either along the column to which the processor belongs, for the rotations applied from the right such as J_I and J_J, or along the corresponding row, for the rotations applied from the left such as J_I' and J_J'. (The choice of representation *per se* will be discussed in Section V.) Each off-diagonal processor, IJ with $I \neq J$, applies J_J from the right and J_I' from the left to the submatrix it holds. Then the entries are exchanged between *neighboring* processors in the array in such a way that processor IJ holds at the beginning of step $k+1$ the submatrix whose row indices and column indices are respectively the I-th and J-th pair of indices at step $k+1$.

For the symmetric eigenvalue problem, at every step J_I' is imposed to be equal to J_I^T, $1 \leq I \leq n/2$. This reduces the amount of computation to be performed by the diagonal processors. Moreover, since all the iterates B_k are symmetric, the array is truncated to a triangular array, *i.e.* all the processors below the diagonal are removed. Such an array is displayed in Figure 1 for $n = 10$; the arrows indicate the communications taking place during the exchanges while the horizontal and vertical links that carry

the representations of the rotations are not shown. (Note that the amount of data communicated during the exchanges could be reduced to about half, which can be argued to be minimal [7], if the order used so far, $\{1\} \cup P^{k-1}S$ with P the cyclic permutation of Brent and Luk and k the step number, is kept when k is odd and is replaced by $\overline{P}(\{1\} \cup P^{k-1}S)$, where $\overline{P}\{1, 2, ..., n\} = \{3, 4, 1, 2, 7, 8, 5, 6, \cdots\}$, when k is even. However this scheme is more complicated to implement.)

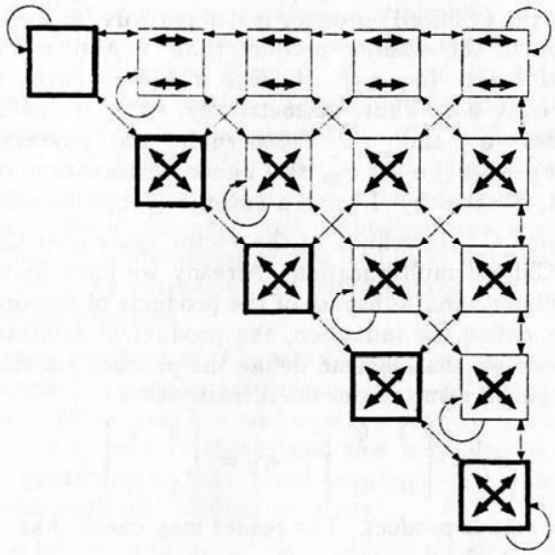

Figure 1. Array for the eigen-decomposition of a symmetric matrix of order 10.

III. CLIFFORD ALGEBRA

We shall consider throughout this section the SVD problem; specialization of the results to the symmetric eigenvalue problem will come in Section V. A diagonal processor, *II*, *evaluates* the left and right rotations, J_I' and J_I, which diagonalize the 2×2 matrix it contains, and computes the new diagonal entries:

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

An off-diagonal processor, *IJ*, *applies* from the left the rotation J_I' received from processor *II* to the matrix it holds, and *applies* from the right the rotation J_J received from processor *JJ*:

$$\begin{bmatrix} \cos\theta_I' & -\sin\theta_I' \\ \sin\theta_I' & \cos\theta_I' \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \cos\theta_J & \sin\theta_J \\ -\sin\theta_J & \cos\theta_J \end{bmatrix}.$$

In a search for the most parallel way of evaluating the rotations and updating the diagonal entries, and also of applying the rotations, we shall study the structure of the space of real 2×2 matrices. The underlying structure to be exploited is that of a Clifford algebra: the Clifford algebra of order 2, $\mathbf{C}_2$. To introduce this structure, we start from a vector space over the reals, $\mathbf{E}_2$, of dimension 2; this vector space is a subspace of $\mathbf{C}_2$. The reason for the notation $\mathbf{E}_2$ is that an Euclidean norm is defined on the vector space, given by the quadratic form $\mathbf{u}^2 \triangleq u_1^2 + u_2^2$, where $\mathbf{u} = (u_1\, u_2)$

belongs to $\mathbf{E}_2$. (Note that a Clifford algebra could also be defined starting with a pseudo-Euclidean form, $u_1^2 - u_2^2$ in two dimensions.) The *scalar product* of two vectors, $\mathbf{u}$ and $\mathbf{v}$, is also an element of $\mathbf{C}_2$, defined as $\mathbf{u} \cdot \mathbf{v} \triangleq (\mathbf{uv} + \mathbf{vu})/2$ where, clearly, $\mathbf{uv} + \mathbf{vu} = (\mathbf{u} + \mathbf{v})^2 - \mathbf{u}^2 - \mathbf{v}^2$ is a scalar.

We have not yet defined $\mathbf{uv}$, the *Clifford product* of the vectors $\mathbf{u}$ and $\mathbf{v}$. Since $\mathbf{uv} = (\mathbf{uv} + \mathbf{vu})/2 + (\mathbf{uv} - \mathbf{vu})/2$, if the *exterior product* $(\mathbf{uv} - \mathbf{vu})/2 \triangleq \mathbf{u} \wedge \mathbf{v}$ is defined, then the (Clifford) product is defined: $\mathbf{uv} = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \wedge \mathbf{v}$. It follows from the definition of the exterior product that $\mathbf{v} \wedge \mathbf{u} = -\mathbf{u} \wedge \mathbf{v}$. Therefore, selecting an orthogonal basis $\{\mathbf{e}_1, \mathbf{e}_2\}$ of $\mathbf{E}_2$, $\mathbf{u} \wedge \mathbf{v} = (u_1\mathbf{e}_1 + u_2\mathbf{e}_2) \wedge (v_1\mathbf{e}_1 + v_2\mathbf{e}_2) = (u_1v_2 - u_2v_1)\,\mathbf{e}_1 \wedge \mathbf{e}_2$. Thus, geometrically, $\mathbf{u} \wedge \mathbf{v}$ defines the area of the parallelogram with sides $\mathbf{u}$ and $\mathbf{v}$. Furthermore the product $\mathbf{uv}$ is equal to $(u_1v_1 + u_2v_2)\mathbf{I} + (u_1v_2 - u_2v_1)\,\mathbf{e}_1 \wedge \mathbf{e}_2$, the linear combination of a scalar (proportional to the scalar unit, denoted by $\mathbf{I}$) and a *bivector* (proportional to $\mathbf{e}_1 \wedge \mathbf{e}_2$).

The Clifford algebra $\mathbf{C}_2$ is defined as the vector space over the reals which is the closure of $\mathbf{E}_2$ under Clifford multiplication. Already we have found two subspaces of $\mathbf{C}_2$, $\mathbf{E}_2$ and the two-dimensional subspace of the products of vectors. To go further we would have to formally define, by induction, the product of arbitrary elements of $\mathbf{C}_2$. Because of a lack of space we shall instead define the product via the shortcut of an isomorphism. The isomorphism results from the identification:

$$\mathbf{e}_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

and *Clifford product = matrix product.* The reader may check that

$$\mathbf{e}_1^2 = \mathbf{e}_1 \cdot \mathbf{e}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}, \quad \mathbf{e}_2 \cdot \mathbf{e}_2 = \mathbf{I}, \quad \mathbf{e}_1 \cdot \mathbf{e}_2 = (\mathbf{e}_1\mathbf{e}_2 + \mathbf{e}_2\mathbf{e}_1)/2 = 0\mathbf{I},$$

hence $\mathbf{e}_1$ and $\mathbf{e}_2$ form an orthonormal basis of $\mathbf{E}_2$. Moreover

$$\mathbf{e}_1 \wedge \mathbf{e}_2 = (\mathbf{e}_1\mathbf{e}_2 - \mathbf{e}_2\mathbf{e}_1)/2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

Hence linear combinations of scalars and bivectors are of the form

$$\mathbf{p} = p_1\mathbf{I} + p_2\mathbf{e}_1 \wedge \mathbf{e}_2 = \begin{bmatrix} p_1 & p_2 \\ -p_2 & p_1 \end{bmatrix},$$

and vectors are of the form

$$\mathbf{q} = q_1\mathbf{e}_1 + q_2\mathbf{e}_2 = \begin{bmatrix} q_1 & q_2 \\ q_2 & -q_1 \end{bmatrix}.$$

Now we observe that any 2×2 real matrix $\mathbf{m}$ may be decomposed as $\mathbf{p} + \mathbf{q}$:

$$\mathbf{m} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \mathbf{p} + \mathbf{q} = \begin{bmatrix} p_1 + q_1 & p_2 + q_2 \\ -p_2 + q_2 & p_1 - q_1 \end{bmatrix},$$

with $p_1 = \frac{a+d}{2}$, $p_2 = \frac{b-c}{2}$, $q_1 = \frac{a-d}{2}$, and $q_2 = \frac{b+c}{2}$. Thus the space of linear combinations of scalars, vectors and bivectors is isomorphic to the linear space of real 2×2 matrices. Since the set of real 2×2 matrices is closed under matrix multiplication, the space of linear combinations of scalars, vectors and bivectors is also closed under Clifford multiplication and is therefore the whole of the Clifford algebra $\mathbf{C}_2$.

Upon identifying the real 2×2 matrices with the Clifford algebra $\mathbf{C}_2$, a decomposition of the real 2×2 matrices into two parts, $\mathbf{p}$ and $\mathbf{q}$, has been brought to the fore. This is an instance of the so-called decomposition of Clifford numbers into *even* and *odd* parts. Indeed a Clifford algebra $\mathbf{C}_m$, built from a vector space $\mathbf{E}_m$, has for elements real linear combinations of scalars or 0-vectors, vectors or 1-vectors, bivectors or 2-vectors, and so on up to m-vectors. Thus any element may be decomposed in a unique way as the sum of an even part (a linear combination of even vectors) and an odd part (a linear combination of odd vectors). In other words $\mathbf{C}_m$ is the direct sum of two subspaces, an 'even' subspace denoted $\mathbf{C}_m^+$, and an 'odd' subspace denoted $\mathbf{C}_m^-$. The even subspace is closed under Clifford multiplication, written symbolically $\mathbf{C}_m^+ \mathbf{C}_m^+ = \mathbf{C}_m^+$, consequently it is a subalgebra of $\mathbf{C}_m$. The odd subspace satisfies $\mathbf{C}_m^- \mathbf{C}_m^- = \mathbf{C}_m^+$ and $\mathbf{C}_m^- \mathbf{C}_m^+ = \mathbf{C}_m^+ \mathbf{C}_m^- = \mathbf{C}_m^-$. (Of interest for the SVD computation of complex matrices is the Clifford algebra $\mathbf{C}_3$, isomorphic to the $2^m = 8$ dimensional space—over the reals—of complex 2×2 matrices, and with *even* subspace the *quaternions* and *odd* subspace the *antiquaternions* [6].) The *even* subspace of $\mathbf{C}_2$ is the algebra of *complex* numbers; by extension, the *odd* subspace of $\mathbf{C}_2$, $\mathbf{E}_2$, may be called the subspace of *anticomplex* numbers.

The units of the even subspace of $\mathbf{C}_2$ are elements $\mathbf{p} = p_1 \mathbf{I} + p_2 \mathbf{e}_1 \wedge \mathbf{e}_2$ with unit norm, where the norm is naturally defined as $(p_1^2 + p_2^2)^{\frac{1}{2}}$. Therefore they can be written under the form $\mathbf{u}(\theta) = \cos\theta \mathbf{I} + \sin\theta \mathbf{e}_1 \wedge \mathbf{e}_2$, with $0 \leq \theta < 2\pi$, and hence they are the plane rotations. It is easy to check (and this may also be derived as a special case of a property of quaternions) that plane rotations *commute* with even Clifford numbers and *anticommute* with odd Clifford numbers:

$$\mathbf{p}\,\mathbf{u}(\theta) = \mathbf{u}(\theta)\,\mathbf{p} \quad , \quad \mathbf{q}\,\mathbf{u}(\theta) = \mathbf{u}(-\theta)\,\mathbf{q} \quad .$$

This enables us to pull the Jacobi rotations from the right to the left, both for the evaluation of the rotations in the diagonal processors and for their application in the off-diagonal processors:

- *evaluation* in processor II,

$$\begin{aligned}\mathbf{u}(-\theta_I')\,\mathbf{m}\,\mathbf{u}(\theta_I) &= \mathbf{u}(-\theta_I')\,\mathbf{p}\,\mathbf{u}(\theta_I) + \mathbf{u}(-\theta_I')\,\mathbf{q}\,\mathbf{u}(\theta_I) \\ &= \mathbf{u}(-\theta_I')\,\mathbf{u}(\theta_I)\,\mathbf{p} + \mathbf{u}(-\theta_I')\,\mathbf{u}(-\theta_I)\,\mathbf{q} \quad ,\end{aligned}$$

hence, using the property that $\mathbf{u}(\theta)$ is isomorphic to the complex number $\exp(i\theta)$,

$$\mathbf{u}(-\theta_I')\,\mathbf{m}\,\mathbf{u}(\theta_I) = \mathbf{u}(-\theta_I' + \theta_I)\,\mathbf{p} + \mathbf{u}(-\theta_I' - \theta_I)\,\mathbf{q} = \overline{\mathbf{m}} = \begin{bmatrix} \overline{a} & 0 \\ 0 & \overline{d} \end{bmatrix} .$$

- *application* in processor IJ,

$$\begin{aligned}\mathbf{u}(-\theta_I')\,\mathbf{m}\,\mathbf{u}(\theta_J) &= \mathbf{u}(-\theta_I')\,\mathbf{p}\,\mathbf{u}(\theta_J) + \mathbf{u}(-\theta_I')\,\mathbf{q}\,\mathbf{u}(\theta_J) \\ &= \mathbf{u}(-\theta_I' + \theta_J)\,\mathbf{p} + \mathbf{u}(-\theta_I' - \theta_J)\,\mathbf{q} \quad .\end{aligned}$$

hence, by pulling the rotations from the right to the left and exploiting the fact that $\mathbf{p}$ and $\mathbf{q}$ are fully defined by their first column, the Jacobi rotations may be applied with 2 two-dimensional vector rotations instead of 4.

Representations of $\mathbf{u}(\theta_I)$ and $\mathbf{u}(-\theta_I')$ must be computed as intermediate forms in order to apply the Jacobi rotations and build up the matrices of singular vectors (or eigenvectors if B is symmetric) U and V^T as the products, accumulated over the steps, of the matrices $\prod_{I=1}^{n/2} J_I$ and $\prod_{I=1}^{n/2} J_I'$, respectively. The *evaluation* of these representations may be performed by finding the rotations $\mathbf{u}(\theta_I^-)$ and $\mathbf{u}(-\theta_I^+)$, where $\theta_I^- \triangleq \theta_I - \theta_I'$ and $\theta_I^+ \triangleq \theta_I + \theta_I'$, that force the second component of the vectors

$(p_1 \ -p_2)^T$ and $(q_1 \ q_2)^T$, respectively, to 0. This approach, exploiting the decomposition of Clifford numbers into even and odd part, has been used on general purpose computers, using standard arithmetic, from very early on (*e.g.* Forsythe and Henrici, 1960). However the use of that decomposition for the *application* of the rotations did not follow. To understand why we have to place ourselves in the context of machines using standard arithmetic. We first note that in this context the passage from $S^{\pm}_{diag} \triangleq \{\mathbf{u}(\theta_I^-), \mathbf{u}(-\theta_I^+), 1 \le I \le n/2\}$ to $S_{diag} \triangleq \{\mathbf{u}(\theta_I), \mathbf{u}(-\theta_I'), 1 \le I \le n/2\}$, and the passage from S_{diag} to $S_{off} \triangleq \{\mathbf{u}(\theta_{IJ}^-), \mathbf{u}(-\theta_{IJ}^+), 1 \le I \ne J \le n/2\}$, where $\theta_{IJ}^- \triangleq \theta_J - \theta_I'$ and $\theta_{IJ}^+ \triangleq \theta_J + \theta_I'$, are done via the trigonometric formulas for the tangents of the rotation angles, using the $\pm$, $\times$ and / operations and also, for the first passage, $\surd$ operations since tangents of half-angles must then be computed. The next observation is that a rotation is ultimately represented by its cosine and sine in this context and, given an intermediate tangent representation, generating the cosine/sine representation requires $+$, $\times$, /, and $\surd$ operations. The reason for not exploiting the decomposition in a *sequential* setting is now clear: the computation of the cosine/sine representation of S_{off} given the tangent representation of S_{diag} requires $O(n^2)$ $\pm$, $\times$, /, and $\surd$ operations while the computation of the cosine/sine representation of S_{diag} given its tangent representation costs only $O(n)$ $+$, $\times$, /, and $\surd$ operations; the halving of multiplications obtained when performing the two-dimensional vector rotations using the decomposition does not offset the large increase in $\pm$, $\times$, /, and $\surd$ operations needed to find the representation of the rotations. The bottom line in a *parallel* setting is that the use of the decomposition is not advantageous either, because it saves the time of the rotation of a two-dimensional vector, *i.e.* a multiply and add, at the expense of the time of the computation of $\tan\theta_{IJ}^-$ or $\tan\theta_{IJ}^+$ given $\tan\theta_J$ and $\tan\theta_I'$, *i.e.* a multiply and add *and a divide.* Yet the existence of the decomposition signals something significant. It removes some sequentiality at the level of the rotation operations and, if an arithmetic implementation of the rotations is employed that is better adapted to these operations than the traditional decomposition into $\pm$ and $\times$ (and the derived / and $\surd$) the advantage offered by the decomposition should clearly come out. CORDIC arithmetic provides the kind of 'adapted' implementation we are looking for.

IV. EXPLICIT AND IMPLICIT CORDIC ALGORITHMS

The CORDIC algorithm of Volder (1959) implements a plane rotation as a sequence of elementary plane rotations. The elementary rotations are rotations with tangents equal to $\sigma_i t_i$, where $\sigma_i = \pm 1$, $t_i = 2^{-i}$, $1 \le i \le l$ and l defines the angular resolution, 2^{-l}. Multiplying a two-dimensional vector by an elementary rotation,

$$\frac{1}{\sqrt{1+t_i^2}} \begin{bmatrix} 1 & \sigma_i t_i \\ -\sigma_i t_i & 1 \end{bmatrix},$$

would be easy to do with two shift-and-adds but for the scaling factor in front of the matrix. By pulling all the scaling factors together into a single multiplicative *constant* (for a given resolution), the basic form of the CORDIC algorithm is obtained:

Explicit CORDIC algorithm for plane rotation

- *evaluation* of a rotation that forces a vector $(x \ y)^T$ into the form $(x' \ 0)^T$

initialization: $x_1 = x$, $y_1 = y$, $z_1 = 0$, $\sigma_1 = \text{sign}(x_1 y_1)$

for $1 \le i \le l$

$$\begin{aligned} x_{i+1} &= x_i + \sigma_i t_i y_i \\ y_{i+1} &= -\sigma_i t_i x_i + y_i \\ z_{i+1} &= z_i - \sigma_i \tan^{-1} t_i \quad \text{(angles } \tan^{-1} t_i \text{ stored in a table)} \end{aligned}$$

$$\sigma_{i+1} = \text{sign}\,(x_i\, y_i)$$

- ***application*** of a rotation by an angle z to a vector $(x \quad y)^T$

initialization: $x_1 = x,\ y_1 = y,\ z_1 = z,\ \sigma_1 = -\text{sign}\,(z_1)$

for $1 \le i \le l$

$$\begin{aligned} x_{i+1} &= x_i + \sigma_i t_i y_i \\ y_{i+1} &= -\sigma_i t_i x_i + y_i \\ z_{i+1} &= z_i - \sigma_i \tan^{-1} t_i \\ \sigma_{i+1} &= -\text{sign}\, z_i \end{aligned}$$

These two sequences of iterations are both followed by the multiplication by the global multiplicative constant, decomposed into a minimal-length sequence of shift-and-adds.

The ***evaluation*** procedure employs essentially a bisection technique to force y_i toward 0, and concurrently updates the angle 'counter' z_i. The *application* procedure employs the same bisection technique to force the angle to zero, hence decomposing it into signed increments, and meanwhile rotates the vector by this sequence of increments.

Quite often, and this is true for the Jacobi method, a rotation that is to be applied is evaluated first, by forcing a vector along the first axis. In such instances it is not necessary to compute the rotation angle explicitly; the sequence of bits $\{\sigma_i, 1 \le i \le l\}$ also defines the angle, albeit in an implicit fashion. Given such a sequence, determined by forcing a vector along the first axis, a vector can be rotated by the corresponding angle with no need for an angle counter. The *implicit* CORDIC algorithm for plane rotation follows: just remove any reference to the variable z in both evaluation and application procedures; the sequence $\{\sigma_i, 1 \le i \le l\}$ will be given, instead of the angle z, for the application procedure. (The *implicit* algorithm is more fundamental than the *explicit* one; it can be generalized to the parallel implementation of higher dimensional rotations while the *explicit* algorithm cannot [5], [6].)

V. CORDIC JACOBI ROTATIONS

The first publication proposing that the CORDIC algorithm for plane rotation be used to implement the parallel Jacobi algorithm of Brent and Luk followed very closely the traditional approach, using standard arithmetic [3]. The explicit algorithm is used. The representation of $S_{diag}^{\pm}$ in terms of the angles, θ_I^- and θ_I^+, is computed by rotating in parallel onto the first axis the first columns of the even and odd parts of the 2×2 submatrices held in the diagonal processors. Recalling a result from Section III, these columns are formed, quite easily, as $(p_1 \quad -p_2)^T = 2^{-1}(a + d \quad c - b)^T$ and $(q_1 \quad q_2)^T = 2^{-1}(a - d \quad c + b)^T$. The representation of S_{diag} in terms of the angles, θ_I and θ_I', is then obtained by means of additions and single bit shifts: $\theta_I = (\theta_I^+ + \theta_I^-)/2$, $\theta_I' = (\theta_I^+ - \theta_I^-)/2$. The application of the rotations in the off-diagonal processors is done without the help of the decomposition into even and odd part, by applying in parallel to the two rows of the 2×2 submatrix held in processor IJ the rotation of angle θ_J and then applying in parallel to the two columns of the result the rotation of angle $-\theta_I'$. The computation of the updated, diagonal, matrices in the diagonal processors is done similarly, hence diagonal and off-diagonal processors finish a step at the same time. If we take as time unit the time to effect a CORDIC rotation, this implementation calls for 1 unit to evaluate S_{diag} and 2 units to apply the rotations, totaling 3 units per step.

Yang and Böhme recently observed that the explicit CORDIC algorithm enables a faster implementation of the Jacobi rotations, fully based on the decomposition of 2×2 real matrices into even and odd part. The representation of $S_{diag}^{\pm}$ is computed as in [3]. Following [4], the computation of the diagonal entries also exploits the decomposition:

once the first columns of $\mathbf{p}$ and $\mathbf{q}$ are rotated into $(p_1' \quad 0)^T$ and $(q_1' \quad 0)^T$, the entries are obtained readily as $\bar{a} = p_1' + q_1'$ and $\bar{d} = p_1' - q_1'$. The representation of S_{off} in terms of the angles θ_{IJ}^- and θ_{IJ}^+ is obtained by first evaluating the angles θ_I and θ_I' as in [3] and then merely adding and subtracting: $\theta_{IJ}^- = \theta_J - \theta_I'$ and $\theta_{IJ}^+ \triangleq \theta_J + \theta_I'$. The computation of S_{off} is definitely much easier than with standard arithmetic! The application of the rotations in the off-diagonal processors is done by first decomposing—as done in the diagonal processors—the 2×2 matrix held into a processor into even and odd part, $\mathbf{m} = \mathbf{p} + \mathbf{q}$, then applying in parallel the rotation by θ_{IJ}^- to the first column of $\mathbf{p}$ and the rotation by $-\theta_{IJ}^+$ to the first column of $\mathbf{q}$, obtaining vectors $(p_1' \quad -p_2')^T$ and $(q_1' \quad q_2')^T$, and finally reconstructing the rotated matrix as $a' = p_1' + q_1'$, $b' = p_2' + q_2'$, $c' = -p_2' + q_2'$, and $d' = p_1' - q_2'$. This implementation calls for 1 unit to diagonalize the matrices held in the diagonal processors and evaluate S_{off}, and 1 unit to apply the rotations, totaling 2 units per step. By better exploiting the mathematical structure than in [3], computation time is reduced by 1/3 with the same hardware.

The use of an *explicit* CORDIC algorithm imposes a degree of sequentiality which can be avoided in an implementation based on an *implicit* CORDIC algorithm. With the *explicit* algorithm the off-diagonal processors can start applying the rotations of a given step only after the rotation angles have been evaluated in the diagonal processors. Moreover, because of the exchange between processors, and more specifically between diagonal and off-diagonal processors, concluding each step, the 'evaluation' in the diagonal processors and the 'application' in the off-diagonal processors cannot be pipelined. This leads to the $1+2$ time units per step of [3] and the $1+1$ time units per step of [9], with the off-diagonal processors idle during the first time unit. However, as was first proposed in [4], the evaluation and application may be overlapped if the rotation angles are computed implicitly, bit by bit, and these bits are sent as soon as computed to the off-diagonal processors.

Assume an implicit CORDIC algorithm is employed in order to overlap the evaluation of the rotations in the diagonal processors and their application in the off-diagonal processors. In order to evaluate the rotations, and also to apply them as fast as possible, the decomposition into even and odd part is used in both diagonal and off-diagonal processors. Consider an off-diagonal processor, IJ. To generate an implicit, bit-level, representation of the angles $\theta_{IJ}^- = \theta_J - \theta_I'$ and $\theta_{IJ}^+ \triangleq \theta_J + \theta_I'$, implicit representations of θ_J and θ_I' must first be generated, in processors JJ and II respectively. These representations are themselves obtained from the bit level representations of $\{\theta_J^-, \theta_J^+\}$ and $\{\theta_I^-, \theta_I^+\}$, using the relations $\theta_J = (\theta_J^+ + \theta_J^-)/2$ and $\theta_I' = (\theta_I^+ - \theta_I^-)/2$. Now the bits, or better 'digits', in the implicit representations are coefficients of angles onto the basis formed by the 'elementary' angles $\tan^{-1} t_i$. Since $t_i = 2^{-i}$ these angles are not commensurable, in the sense that one angle cannot be obtained as a linear combination of other angles with coefficients that are signed powers of two, *i.e.* by which a multiplication can easily be performed. This justifies the 'basis' denomination employed earlier. This also implies that if an angle is represented by a sequence $\sigma \triangleq \{\sigma_i, 1 \le i \le l\}$ with $\sigma_i = \pm 1$ and another angle is represented by a sequence $\sigma' \triangleq \{\sigma_i', 1 \le i \le l\}$ with $\sigma_i' = \pm 1$, finding a representation of the sum of the two angles by a sequence $\sigma^+ \triangleq \{\sigma_i^+, 1 \le i \le l\}$ *with* $\sigma_i' = \pm 1$ is difficult. In particular the computation of σ_1^+ depends on the whole sets σ and σ'. Sequentiality would thus come back to haunt us. To get around this problem one should pursue the 'basis' paradigm and think of the sequences σ and σ' as vectors, with ith component σ_i and σ_i' respectively. A representation, σ^+, of the sum is obtained by merely adding components and defining $\sigma_i^+ = \sigma_i + \sigma_i'$, equal to 0 or ± 2. This construction is denoted $\sigma^+ = \sigma + \sigma'$, like a vector addition. In general the components, or digits, in the representations are 0 or signed powers of two. Thus, if an angle must be divided by two, it should not have components equal to ± 1. Since $\theta_I' = (\theta_I^+ - \theta_I^-)/2$ the components of σ_I^- and σ_I^+

may, it seems, be taken equal to ± 1, then $\sigma_I' = 2^{-1}(\sigma_I^+ - \sigma_I^-)$ has components equal to 0 or ± 1. However this means that on the ith iteration, the elementary rotation would have tangent 0 or $\pm t_i$. The scaling factors differ in both cases and, to preserve a constant global scaling, when the tangent is 0 the components of the vector being rotated, x_i and y_i, should be multiplied by $\sqrt{1+t_i^2}$. Unfortunately this cannot be done in a single iteration with shift-and-add hardware. On the other hand, if the components of $\sigma_I^-/2$ and $\sigma_I^+/2$ are taken equal to ± 1, they can be evaluated by rotating along the first axis the first columns of $\mathbf{p}$ and $\mathbf{q}$ using an implicit CORDIC algorithm with 'double' elementary rotations

$$\left\{ \frac{1}{\sqrt{1+t_i^2}} \begin{bmatrix} 1 & \sigma_i t_i \\ -\sigma_i t_i & 1 \end{bmatrix} \right\}^2 = \frac{1}{1+t_i^2} \begin{bmatrix} 1-t_i^2 & \sigma_i \cdot 2t_i \\ -\sigma_i \cdot 2t_i & 1-t_i^2 \end{bmatrix} .$$

Indeed, the ith elementary rotation rotates by $\pm 2 \tan^{-1} t_i$ and hence the process of rotating with these elementary rotations a vector along the first axis generates the decomposition $\{\sigma_i = \pm 1, 1 \le i \le l\}$ of half the angle between the vector and the axis. Of course, as in the standard algorithm of Section IV, the rotations are applied unscaled and the multiplication by the constant equal to the the product of the scaling factors, decomposed into a short sequence of shifts and adds, is applied afterwards. From the implicit representations $\sigma_I^-/2$ and $\sigma_I^+/2$, with components ± 1, the implicit representation of θ_I' is obtained readily as: $\sigma_I' = \sigma_I^+/2 - \sigma_I^-/2$, with components 0 or ± 1. Similarly, the representation of θ_J is obtained from $\sigma_J^-/2$ and $\sigma_J^+/2$ according to: $\sigma_J = \sigma_J^+/2 + \sigma_J^-/2$. However, because we use the decomposition into even and odd part, we are really interested in θ_{IJ}^- and θ_{IJ}^+. Their implicit representations are $\sigma_{IJ}^- = \sigma_J - \sigma_I'$ and $\sigma_{IJ}^+ = \sigma_J + \sigma_I'$ or, in terms of the sequences evaluated by the diagonal processors II and JJ,

$$\sigma_{IJ}^- = \sigma_J^+/2 + \sigma_J^-/2 - \sigma_I^+/2 + \sigma_I^-/2 \ , \quad \sigma_{IJ}^+ = \sigma_J^+/2 + \sigma_J^-/2 + \sigma_I^+/2 - \sigma_I^-/2 \ .$$

Therefore the representations have components 0, ± 2 or ± 4.

A highly parallel implementation of the parallel Jacobi algorithm of Brent and Luk for the SVD, based both on the decomposition of 2×2 matrices into even and odd part and on the use of implicit CORDIC algorithms, may now be described. The algorithm exploits the decomposition exactly like the algorithm in [9]; the differences are only in the CORDIC algorithms employed:

- In *diagonal* processor II, the vectors $(p_1 \ -p_2)^T = 2^{-1}(\ a+d \quad c-b\ )^T$ and $(q_1 \ \ q_2)^T = 2^{-1}(\ a-d \quad c+b\ )^T$ are formed first. Then, in parallel, the two vectors are rotated along the first axis with double elementary rotations, thus generating at each iteration one of the signs in each sequence $\sigma_I^-/2$ and $\sigma_I^+/2$. As soon as evaluated, both signs are sent (more precisely, propagated in a systolic fashion) to the off-diagonal processors along both row I and column I. The scaling iterations are then applied in parallel to the two vectors, yielding $(p_1' \ \ 0)^T$ and $(q_1' \ \ 0)^T$. Finally, the diagonal entries of the rotated matrix are obtained as $\bar{a} = p_1' + q_1'$ and $\bar{d} = p_1' - q_1'$.

- In *off-diagonal* processor IJ, the vectors $(p_1 \ -p_2)^T = 2^{-1}(\ a+d \quad c-b\ )^T$ and $(q_1 \ \ q_2)^T = 2^{-1}(\ a-d \quad c+b\ )^T$ are formed first. Then, in parallel, the scaling iterations are applied to the two vectors. (Note that the global scaling factor is the square of the scaling factor for double elementary rotations, which is itself the square of the scaling factor of the standard CORDIC algorithms of Section IV.) Before the last scaling iteration the first sign in each sequence, $\sigma_I^-/2$, $\sigma_I^+/2$, $\sigma_J^-/2$ and $\sigma_J^+/2$, reaches the processor. The sequence of unscaled elementary rotations is then applied simultaneously on both vectors; to the p-vector are applied the rotations defined by the components of σ_{IJ}^- and to the q-vector are applied the rotations defined by the components of $-\sigma_{IJ}^+$. Let us denote, locally, by σ_i the ith component of σ_{IJ}^- or $-\sigma_{IJ}^+$; the

associated angle is $\sigma_i \tan^{-1} t_i$. If $\sigma_i = \pm 4$ the unscaled elementary rotation matrix is

$$\begin{bmatrix} 1 & t_i \\ -t_i & 1 \end{bmatrix}^{\pm 4} = \begin{bmatrix} 1 - 6t_i^2 + t_i^4 & \pm (4t_i - 4t_i^3) \\ \pm (-4t_i + 4t_i^3) & 1 - 6t_i^2 + t_i^4 \end{bmatrix}.$$

If $\sigma_i = \pm 2$ the unscaled elementary rotation matrix is

$$(1 + t_i^2) \begin{bmatrix} 1 & t_i \\ -t_i & 1 \end{bmatrix}^{\pm 2} = \begin{bmatrix} 1 - t_i^4 & \pm (2t_i + 2t_i^3) \\ \pm (-2t_i - 2t_i^3) & 1 - t_i^4 \end{bmatrix}.$$

If $\sigma_i = 0$, a nil rotation, the unscaled elementary rotation matrix is

$$(1 + t_i^2)^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 + 2t_i^2 + t_i^4 & 0 \\ 0 & 1 + 2t_i^2 + t_i^4 \end{bmatrix}.$$

Denoting the resulting rotated vectors by $(p_1' \quad -p_2')^T$ and $(q_1' \quad q_2')^T$, the rotated matrix is eventually constructed as $a' = p_1' + q_1'$, $b' = p_2' + q_2'$, $c' = -p_2' + q_2'$, and $d' = p_1' - q_2'$.

The diagonal processors consist of two 'double' rotation implicit CORDIC modules. A custom chip implementing such modules has been designed; it operates on 32-bit fixed point words, with 5 extra guard bits. It has been fabricated with a 2μ CMOS process and performs the CORDIC iterations at 11 MHz. Its area is slightly smaller than the area of a chip implementing the method of Yang and Böhme, using the explicit CORDIC algorithm of Section IV. It requires both $2t_i$ and t_i^2 shifters instead of just t_i shifters; the shifter area is 1.4 times the area of the shifter for the 'explicit' method. However the shifters require less area than the adders, and the adder area for the explicit method is about 1.4 times the area for our method. (To add 3 numbers an array of 3-to-2 carry save adders, whose area is about one tenth of the area of a fast adder, is used to reduce the numbers to be added to 2). Finally, a ROM is needed to store the angles $\tan^{-1} t_i$ in the explicit method.

The off-diagonal processors consist of two 'quadruple' rotation implicit CORDIC modules. The shifter area is about twice that of the diagonal processor modules. The adder area is slightly larger than that of a diagonal module: 5-to-2 carry save adders replace 3-to-2 carry save adders and extra wiring is needed to bring in the inputs. The total area should be about 1.4 times the area of a diagonal module. The cycle time for an iteration must be about 15 per cent longer than for the standard CORDIC algorithm used by Yang and Böhme. However, because of the complete overlap of the application of the rotations in the off-diagonal processors with their evaluation in the diagonal processors, a step requires half as many cycles as with Yang and Böhme's method.

In the special case of the *symmetric* eigenvalue problem a simpler, very elegant, parallel architecture is obtained. Indeed, the symmetry implies that $\theta_I' = \theta_I$, hence $\theta_I^- = 0$ and only the sums θ_I^+ need to be evaluated by the diagonal processors. More precisely, only the sequence of signs $\sigma_I^+/2$ must be generated and sent. The off-diagonal processors apply rotations defined by the components of

$$\sigma_{IJ}^- = \sigma_J - \sigma_I' = \sigma_J^+/2 - \sigma_I^+/2 \quad \text{and} \quad \sigma_{IJ}^+ = \sigma_J + \sigma_I' = \sigma_J^+/2 + \sigma_I^+/2 \ .$$

- In *diagonal* processor II, consisting of a single double rotation module, the vector $(q_1 \quad q_2)^T = 2^{-1}(a - d \quad 2b)^T$ is formed first. Then the vector is rotated along the first axis with double elementary rotations, thus generating at each iteration one of the signs in the sequence $\sigma_I^+/2$. As soon as evaluated, each sign is propagated to the off-diagonal processors along both row I and column I. The scaling iterations are then applied to the vector, yielding $(q_1' \quad 0)^T$. Finally, the diagonal entries of the rotated

matrix are obtained as $\bar{a} = 2^{-1}(a + d) + q_1'$ and $\bar{d} = 2^{-1}(a + d) - q_1'$.

- In *off-diagonal* processor IJ, consisting of two double rotation modules, the vectors $(p_1 \ -p_2)^T = 2^{-1}(\ a + d \quad c - b\ )^T$ and $(q_1 \ \ q_2)^T = 2^{-1}(\ a - d \quad c + b\ )^T$ are formed first. Then, in parallel, the scaling iterations are applied to the two vectors. (The global scaling factor is the square of the scaling factor for the standard CORDIC algorithm.) Before the last scaling iteration the first sign in each sequence, $\sigma_I^+/2$ and $\sigma_J^+/2$, reaches the processor. The sequence of unscaled elementary rotations is then applied simultaneously on both vectors; to the p-vector are applied the rotations defined by the components of σ_{IJ}^- and to the q-vector are applied the rotations defined by the components of $-\sigma_{IJ}^+$. These components, denoted locally by σ_i, can only take three values: 0 and ± 2. If $\sigma_i = \pm 2$ the unscaled elementary rotation matrix is

$$\begin{bmatrix} 1 & t_i \\ -t_i & 1 \end{bmatrix}^{\pm 2} = \begin{bmatrix} 1 - t_i^2 & \pm 2t_i \\ \pm(-2t_i) & 1 - t_i^2 \end{bmatrix}.$$

If $\sigma_i = 0$, a nil rotation, the unscaled elementary rotation matrix is

$$(1 + t_i^2)\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 + t_i^2 & 0 \\ 0 & 1 + t_i^2 \end{bmatrix}.$$

Only one module type, the implicit double rotation module already designed, is needed. Moreover, thanks to the *implicit* nature of the algorithm, rotations with *variable angular resolution* [5], [8] can easily be evaluated and applied, still in a fully parallel way. By starting from a low resolution and increasing the resolution in later steps, the number of CORDIC iterations per step may be decreased significantly with almost no increase in the total number of steps.

ACKNOWLEDGEMENTS

The work presented in this paper has been supported by the Defense Advanced Research Projects Agency under contract N00014-88-K-0573.

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 & Comp. Syst.*, 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, "A Processor for Two-dimensional Symmetric Eigenvalue and Singular Value Arrays," *Proc. 21st Asilomar Conf. on Circuits, Systems and Computers*, Pacific Grove, CA, pp. 217-221, Nov. 1987.

[5] J.-M. Delosme, "CORDIC Algorithms: Theory and Extensions," *Advanced Algorithms and Architectures for Signal Processing IV, Proc. SPIE 1152*, pp. 131-145, Aug. 1989.

[6] J.-M. Delosme, "Parallel Computation of Real and Complex SVD Using Implicit CORDIC Arithmetic," *Proc. 2nd Workshop on SVD and Signal Processing, June 1990* (forthcoming book, Elsevier Science Publishers).

[7] G.R. Gao and S.J. Thomas, "An Optimal Parallel Jacobi-Like Solution Method for the Singular Value Decomposition," *IEEE Int. Conf. on Parallel Processing*, pp. 47-53, 1988.

[8] F.T. Luk and D.E. Schimmel, "A Novel Bit-Level Algorithm for the Symmetric Eigenvalue Problem," contributed presentation, *1989 SIAM Annual Meeting*, San Diego, July 1989.

[9] B. Yang and J.F. Böhme, "Reducing the Computations of the SVD Array Given by Brent and Luk," *Adv. Algor. & Arch. for Sig. Proc. IV, Proc. SPIE 1152*, pp. 92-102, Aug. 1989.