Proceedings of SPIE—The International Society for Optical Engineering

Volume 698

Real Time Signal Processing IX

William J. Miceli
Chair/Editor

Sponsored by
SPIE—The International Society for Optical Engineering

Cooperating Organizations
Center for Applied Optics/University of Alabama in Huntsville
Center for Electro-Optics/University of Dayton
Institute of Optics/University of Rochester
Jet Propulsion Laboratory/California Institute of Technology
NASA/Ames Research Center
Optical Sciences Center/University of Arizona

21-22 August 1986
San Diego, California

Published by
SPIE—The International Society for Optical Engineering
P.O. Box 10, Bellingham, Washington 98227-0010 USA
Telephone 206/676-3290 (Pacific Time) • Telex 46-7053

Architectures for a CORDIC SVD Processor

Joseph R. Cavallaro and Franklin T. Luk

School of Electrical Engineering, Cornell University
Ithaca, New York 14853

Abstract

Architectures for systolic array processor elements for calculating the singular value decomposition (SVD) are proposed. These special purpose VLSI structures incorporate the coordinate rotation (CORDIC) algorithms to diagonalize 2×2 submatrices of a large array. The area-time complexity of the proposed architectures is analyzed along with topics related to a prototype implementation.

Introduction

Real-Time Signal Processing is undergoing rapid development due to recent advances in parallel architectures and VLSI. Many important algorithms that were once considered too computationally complex are now being reinvestigated. One such algorithm is the Singular Value Decomposition. The SVD provides a reliable way to detect and correct the ill-conditioning that can occur in data matrices received from sensor arrays.[1] Digital image processing also uses this algorithm for image enhancement.[2]

Parallel architectures, in particular systolic arrays, show great potential for improving the performance of the SVD.[3] Recent research has shown that special purpose VLSI structures are possible for the SVD.[4,5] It has been suggested that novel architectures which "map" the algorithm more closely to hardware are desirable.[6] The coordinate rotation algorithms, CORDIC, have been shown to have enormous potential in this application[4,7] due to their ability to compute inverse tangents and vector rotations.

With the systolic array organization of Brent, Luk, and Van Loan,[8] a high degree of parallelism can be obtained. In this paper, several novel architectures for a CORDIC 2×2 SVD processor are proposed. These modular 2×2 submatrix units can be combined to build a larger Brent-Luk-Van Loan array for use in real-time signal processing applications. The array contains diagonal processors that compute the two-sided rotations, and off-diagonal processors that apply them.

In this paper, the area and time complexity of the various architectures will be analyzed. The goal is the production of a CORDIC architecture that will compute the SVD in the minimum amount of time and area. A computer simulation has been performed to compare these architectures and a prototype system is planned which is based upon the architecture with the smallest execution time. As part of this work, the domain of convergence of the CORDIC inverse tangent function has been extended due to an enhancement to the algorithm. Additionally, the results which are presented here for a fixed-point implementation can be extended to floating-point arithmetic through modifications to the data paths of the CORDIC processors.

SVD - Jacobi method

The singular value decomposition [9] of an $p\times p$ matrix M is

$$M = U\Sigma V^T , \tag{1}$$

where U and V are orthogonal matrices and Σ is a diagonal matrix of singular values.

The Jacobi method seeks to systematically reduce the off-diagonal elements to zero. This is done by applying a sequence of plane rotations to M which transforms M into Σ. Several sweeps over the entire matrix M may be necessary to complete the SVD. Within each sweep, the matrix elements need to be paired and appropriate rotations need to be calculated. The $p\times p$ matrix is distributed over an array of $\left\lfloor\frac{p}{2}\right\rfloor \times \left\lfloor\frac{p}{2}\right\rfloor$ simple 2×2 processors where the basic operation is the two-sided rotation of each 2×2 matrix.

Basic methods for a 2×2 matrix

A 2×2 SVD can be described as

$$R(\theta_l)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix}, \tag{2}$$

where θ_l and θ_r are the left and right rotation angles, respectively. The rotation matrix is

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}, \tag{3}$$

and the input matrix is

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}. \tag{4}$$

The goal is to compute the rotations efficiently. Several methods are possible to solve this problem. One method uses $\theta_{SYM} = (\theta_l - \theta_r)$ to symmetrize M and θ_r to diagonalize M, while a second method uses a parallel calculation of θ_l and θ_r to diagonalize M.

The rotation parameters can be calculated from the inverse tangents of the elements of M. Also, the diagonalization of M can be performed by treating M as a pair of vectors and using the rotation angles to transform M. The computation of these vector rotations and inverse tangents can be performed efficiently by the CORDIC algorithms.

CORDIC algorithms

The CORDIC algorithms were first presented in 1959 by J. Volder.[10] CORDIC is an acronym for Coordinate Rotation Digital Computer. Further theoretical work was done by J. Walther [11] in 1971 to show the applicability of CORDIC to various functions. During the last several years, there has been renewed interest in CORDIC algorithms, principally due to the possibility of VLSI implementation[12] and the application to real-time signal processing.[4,6]

In this section, the CORDIC algorithms will be described. Additionally, the applicability of CORDIC to the basic operations in the SVD will be presented along with the limitations of the algorithm. The CORDIC algorithms provide an iterative VLSI method to calculate the transcendental and hyperbolic functions. The goals have been fast hardware calculation of sin, cos, arctan, sinh, cosh, arctanh, product, quotient, and square root. The target application given here is a math processor within a special purpose systolic array.

In a typical serial computer, the calculation of the rotation angles for the SVD is expensive and can be avoided by finding the sines and cosines directly. Matrix-vector multiplication can then be used to apply the rotations to the 2×2 submatrix. With the CORDIC algorithms, the inverse tangent function is a primitive operation and the angles can be found explicitly, without penalty. Also, vector rotations are primitive CORDIC operations and can replace traditional matrix-vector multiplication.

CORDIC equation set

The CORDIC algorithms are based upon defining a vector (x_0, y_0) in the 2-plane, and then applying a rotational transformation. That is, the vector (x_0, y_0) is rotated through an angle θ, in the clockwise direction, to (x_0', y_0'). The CORDIC equations describe a rotation in one of three modes: circular, linear, or hyperbolic. For the SVD, the rotations are in the circular mode and the equations are:

$$\begin{bmatrix} x_0' \\ y_0' \end{bmatrix} = R(\theta) \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}. \quad (5)$$

The CORDIC algorithms decompose the rotation angle into a sequence of n known smaller angles, such that

$$\theta = \pm\theta_0 \pm \theta_1 \cdots \pm \theta_{n-1} = \sum_{i=0}^{n-1} \delta_i \theta_i , \quad (6)$$

where $\theta_i > 0$ and $\delta_i = \pm 1$. From the geometry of rotations, it is clear that the result of n rotations using the sequence of θ_i's is equivalent to that of one rotation using θ. The number of known angles in the sequence determines the accuracy of the CORDIC algorithms. In order to achieve n bits of accuracy, at least n rotations must be performed.

From the rotation equations, the recurrence equations describing these rotations can be found. If the recurrence equations are divided by $\cos\theta_i$, then

$$\frac{x_{i+1}}{\cos\theta_i} = x_i + \delta_i y_i \tan\theta_i , \quad (7)$$

$$\frac{y_{i+1}}{\cos\theta_i} = y_i - \delta_i x_i \tan\theta_i . \quad (8)$$

The key contribution of Volder[10] and Walther[11] was to set $\tan\theta_i = \beta^{-i}$ where β is the machine radix. In most applications, binary arithmetic is used, so $\beta = 2$, and therefore multiplication by $\tan\theta_i$ becomes a simple arithmetic shift operation. For example, when $i = 0$, then $2^{-i} = 1$ and $\theta_i = \tan^{-1}(2^{-i}) = 45°$. Again, for $i = 1$, $\theta_i = 26.7°$. Obviously, as i increases, θ_i decreases toward 0.

The CORDIC formulation is not yet complete since the vector is not only rotated but also scaled at each iteration. This scaling is only by a constant, and can be factored from the recurrence equations. If $k_i \equiv \cos\theta_i$, then the CORDIC equations are:

$$x_{i+1} = k_i (x_i + \delta_i y_i 2^{-i}), \quad (9)$$

$$y_{i+1} = k_i (y_i - \delta_i x_i 2^{-i}). \quad (10)$$

If the multiplication by k_i is postponed until after the completion of n iterations, then the scale factor, K_n, can be defined as:

$$K_n = \prod_{i=0}^{n-1} k_i = \prod_{i=0}^{n-1} \cos\theta_i . \quad (11)$$

The final CORDIC equations are:

$$x_{i+1} = x_i + \delta_i y_i 2^{-i} , \quad (12)$$

$$y_{i+1} = y_i - \delta_i x_i 2^{-i} . \quad (13)$$

These equations are executed for n iterations and then a final scale factor multiplication by K_n is performed.

Scale factor considerations

The major limitations of the CORDIC algorithms are the treatment of the scale factors and the narrow domain of convergence. Many proposals [6, 12, 13, 14] have been made to cope with the scale factor problems.

In most CORDIC designs, the number of iterations, n, is fixed. This permits K_n to be fixed and for a traditional sequence of angles without repetitions, $K_n \approx 0.607$. In many cases, fewer than n iterations are necessary. However, the storage of K_n for all possible values of n is necessary if early termination is permitted. This limitation of the CORDIC algorithms is presented in the discussion by Bridge et al.[15] of techniques for latency reduction.

Two classes of techniques are described in the literature to cope with the scale factor dilemma. They include either

special scale factor compensation iterations [12,14] or modified repetitive angle sequences.[6,13] These methods have been developed to eliminate the final costly multiplication by K_n that would otherwise be necessary with Walther's[11] original algorithm.

Despain described a "compensated" CORDIC algorithm.[14] This method applies a correction factor at selected iterations to try to force the scale factor, K_n, to unity. In this way, the magnitude correction iterations can be combined with the rotation iterations by using an additional parameter.

Haviland and Tuszynski [12] have implemented an approach similar to that of Despain. Their CORDIC processor is capable of performing either a CORDIC rotation step or a special scale factor reduction step. If these extra steps are performed for certain iterations, then the scale factor is reduced to unity.

The special scale factor compensation iteration technique has several drawbacks. First, the area complexity of the control structure is increased to allow for the special cycles and extra data handling. Second, although the extra iterations do not add to the time complexity as much as an explicit multiplication, these iterations do not extend the domain of convergence.

Ahmed[6] suggested a method that is based upon repeating certain iterations. Since K_n is based upon the product of the individual $\cos\theta_i$, it is possible to repeat certain iterations so that K_n will become a power of the machine radix. Ahmed proposed an example angle sequence that yields $K_n \approx 0.50$. Therefore, a simple arithmetic shift after the last iteration will correct for scaling. As a benefit, the extra iterations increase the domain of convergence at the expense of extra time required for a complete operation.

Delosme [13] extended Ahmed's work and combined it with that of Haviland and Tuszynski. He described an optimization procedure to eliminate the scale constant that uses the least number of scale factor compensation iterations and repeated iterations. However, this method produces an implementation that requires increased complexity in the control structure.

CORDIC operation modes

The CORDIC algorithms can be generalized to provide the calculation of several functions. In order to facilitate these operations, a third equation is added to the two rotation equations to accumulate the choice of angle used at each iteration:

$$z_{i+1} = z_i + \delta_i \theta_i . \tag{14}$$

The variable, z_i, contains the total rotation angle used, θ_i is the current rotation angle increment, and $\delta_i = \pm 1$ indicates whether to add or subtract this angle increment.

In a full CORDIC processor, either the initial z_0 value can be reduced to zero (z-reduction) or the initial y_0 value can be reduced to zero (y-reduction). Through the appropriate selection of operating mode, the CORDIC processor can yield various elementary functions.

Vector rotation

In the circular mode, the z-reduction will yield a vector rotation or the sine and cosine of the original angle. Again consider the CORDIC equations. If, after n iterations, $z_n = 0$, then the angle $\theta = z_0$ and

$$x_n = K_n(x_0 + y_0\tan(z_0)), \tag{15}$$

$$y_n = K_n(y_0 - x_0\tan(z_0)). \tag{16}$$

This represents rotating (x_0, y_0) by the angle z_0. The application of vector rotations is an important step in the SVD. Note, however, that the scale factor K_n does remain in this calculation. This requires the use of one of the scale factor correction techniques, such as the repetitive angle sequences proposed by Ahmed.

Inverse tangent

In the circular mode, the y-reduction will yield the quantity $\tan^{-1}(y_0/x_0)$. This can be shown as follows. Consider the CORDIC equations:

$$x_n = K_n(x_0 + y_0\tan\theta), \tag{17}$$

$$y_n = K_n(y_0 - x_0\tan\theta), \tag{18}$$

$$z_n = z_0 + \theta. \tag{19}$$

If, after n iterations, $y_n = 0$, then

$$\frac{y_0}{x_0} = \tan\theta. \tag{20}$$

Thus, $\theta = \tan^{-1}(y_0/x_0)$ and if $z_0 = 0$, then

$$z_n = \tan^{-1}(y_0/x_0). \tag{21}$$

Note that the scale factor K_n cancels from the calculation.

Convergence issues

Walther [11] has shown that the domain of convergence of the CORDIC algorithms is limited by the sum of the series of the n known rotation angles. Therefore, since $\theta_i > 0$, the maximum angular rotation, α_0, is given by

$$\alpha_0 = \sum_{i=0}^{n-1} \theta_i . \tag{22}$$

If a non-repetitive sequence of angles ($i = 0, \cdots, n-1$) is used for the circular mode, then $\alpha_0 \approx 99°$. Once the angle α satisfies $|\alpha| > \alpha_0$, the CORDIC algorithms no longer converge. The result remains the same as that for $\text{sign}(\alpha)\,\alpha_0$.

The CORDIC convergence properties are related to the behavior of the tangent function. For any $i = 0, \cdots, n-1$, it is required that

$$\theta_i - \sum_{j=i+1}^{n-1} \theta_j < \theta_{n-1} . \tag{23}$$

In the circular mode, the inverse tangent function is used and this relation holds since

$$\tan^{-1}(2^{-i}) < 2\tan^{-1}(2^{-(i+1)}). \quad (24)$$

Since the circular mode convergence is limited to ± 99°, which covers the first and fourth quadrants of the unit circle, a new extension to the algorithm is proposed here to allow for angles in the second and third quadrants. An initial test is performed to check the signs of x_0 and y_0. If both x_0 and y_0 are negative (third quadrant), then the signs of both x_0 and y_0 are changed in order to move the angle into the first quadrant. Similarly, if x_0 is negative and y_0 is positive (second quadrant). then the signs of both x_0 and y_0 are changed in order to move the angle into the fourth quadrant. These modifications to the CORDIC algorithm allow the computation of the $\tan^{-1}(y_0/x_0)$ for all x_0 and y_0 except $x_0 = y_0 = 0$. This property makes the CORDIC module an excellent choice for finding the rotation parameters for the SVD.

Area and time complexity

The VLSI model of computation concerns both the area and the time needed to perform an operation. The best VLSI architecture for the solution of a given problem has the least area and time.[16] The area, A_{CSVD}, and time, T_{CSVD}, complexity of the proposed CORDIC SVD architectures will be compared and presented in terms of the area, A_C, and time, T_C, complexity of a basic fully parallel CORDIC processor.

The area complexity of an n bit CORDIC processor which performs n iterations can be determined from the fully parallel CORDIC processor design[6] shown in Figure 1. The main substructures will be a programmable logic array (PLA) for finite state control, a ROM for storage of the angles used by the CORDIC algorithm, and hardware for the x, y, and z variables, such as barrel shifters (SH), adders (ADD), and registers (REG). Therefore, the total area of a CORDIC processor, A_C, is

$$A_C = A_{PLA} + A_{ROM} + 2A_{SH} + 3A_{ADD} + 3A_{REG}. \quad (25)$$

For a fixed-point implementation, the largest area in this design will be used by the barrel shifters which have been selected to multiply by 2^{-i} in the least amount of time. Since a constant time shift is desired, the area complexity of an n-bit barrel shifter will be $O(n^2)$. Therefore, the area complexity of an entire CORDIC module will be

$$A_C \approx 2A_{SH} = O(n^2). \quad (26)$$

The internal structure of a parallel fixed point CORDIC processor is based upon the form of a CORDIC rotation equation:

$$x_i \leftarrow x_i + \delta_i \text{SHIFT}(y_i). \quad (27)$$

Therefore, the time for one CORDIC iteration, T_{Ci}, is

$$T_{Ci} = T_{ADD} + T_{SH} + T_{ST}, \quad (28)$$

where T_{ADD}, T_{SH}, T_{ST} are, respectively, the time for addition, shifting, and the sign test that determines $\delta_i = \pm 1$.

The total time for a complete CORDIC operation, T_C, is

$$T_C = n(T_{ADD} + T_{SH} + T_{ST}), \quad (29)$$

where n is the number of bits in the operands. For example, the time to compute an inverse tangent, T_{ATAN}, is T_C.

The relative complexity of the primitive operations can be compared by making the following assumptions. First, if a barrel shifter design is used for the shifter implementation, then all distance shifts occur in equal time, and the approximation can be made that $T_{SH} << T_{ADD}$. In a two's complement fixed-point implementation, the sign test will determine whether addition or subtraction is to be performed, and $T_{ST} << T_{ADD}$. From these assumptions, the limiting factor in a CORDIC SVD processor is the time needed to perform an addition, T_{ADD}. The time for a CORDIC operation depends linearly on the number of bits in the operands and

$$T_C \approx T_{ADD}\, n = O(T_{ADD}\, n). \quad (30)$$

CORDIC SVD processor architectures

The Jacobi method for the SVD has been shown to rely heavily upon two basic functional blocks, the inverse tangent and the vector rotation. The CORDIC algorithms have been described and have been shown to be precisely capable of performing these functions. Thus. a general algorithm for a 2× 2

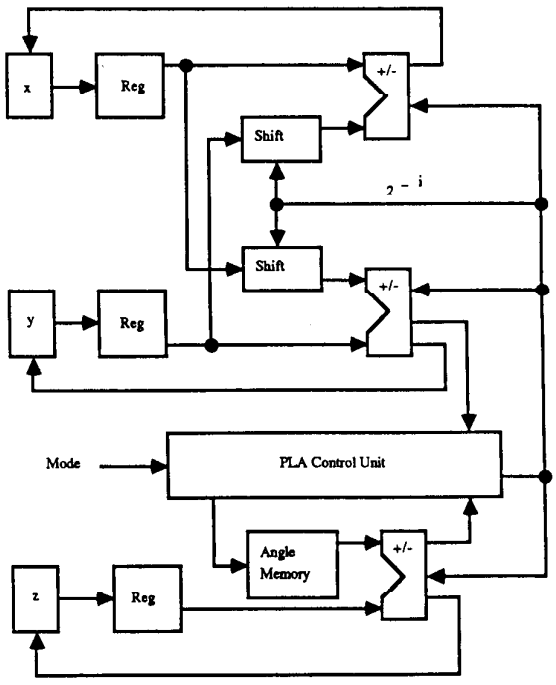

Figure 1. Parallel fixed-point CORDIC processor.

CORDIC SVD processor would be:

```
Algorithm CORDIC SVD ( ):
    begin
        Use CORDIC angle-solver module to find
            rotation angles;
        Use CORDIC rotation module to
            transform the 2× 2 matrix;
    end.
```

The four novel CORDIC architectures to be discussed perform variations on this algorithm with different time and area costs. A computer simulation has been developed to analyze the methods. Each method which is introduced possesses an advantage in area or time. The first method is the most basic while the fourth method exploits the maximum amount of parallelism to obtain the minimum area-time complexity.

CORDIC two step architectures

The two step approach [8] first uses a rotation to symmetrize M. The angle, θ_{SYM}, is defined as

$$\theta_{SYM} = (\theta_l - \theta_r) = \tan^{-1}\left[\frac{b-c}{a+d}\right], \qquad (31)$$

and is applied to M as follows:

$$R(\theta_{SYM})^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} e & f \\ f & g \end{bmatrix}. \qquad (32)$$

After the matrix is symmetrized, the diagonalization angle θ_{DIAG} can be found from:

$$\theta_{DIAG} = \theta_r = \frac{1}{2}\tan^{-1}\left[\frac{2f}{g-e}\right]. \qquad (33)$$

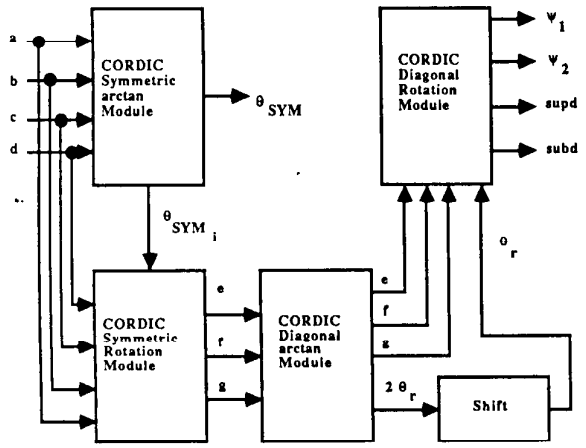

Figure 2. CORDIC SVD symmetrization, diagonalization method.

A two-sided rotation using θ_r will diagonalize the matrix:

$$R(\theta_r)^T \begin{bmatrix} e & f \\ f & g \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix}. \qquad (34)$$

Symmetrization, diagonalization method

This method is a direct mapping of the equations to CORDIC inverse tangent and vector rotation modules. The architecture is sequential and will require the largest CORDIC SVD execution time, T_{CSVD}, to diagonalize a 2× 2 matrix. Figure 2 illustrates the implementation of the following algorithm:

```
Algorithm CORDIC SVD Sym-Diag ( ):
    begin
        parallel do { b−c, a+d };
        Use CORDIC module to find θ_SYM;
        Use CORDIC rotation module to apply θ_SYM;
        parallel do { g−e, SHIFT( f ) };
        Use CORDIC module to find 2θ_r;
        θ_r ← SHIFT( 2θ_r );
        Use CORDIC rotation module to apply θ_r;
    end.
```

The total execution time is

$$T_{CSVD} = T_{SYM} + T_{DIAG}, \qquad (35)$$

where T_{SYM} and T_{DIAG} are, respectively, the time for symmetrizing and diagonalizing the matrix M. An initial addition, T_{ADD}, is needed to prepare the operands for the calculation of θ_{SYM}. The inverse tangent computation time, T_{ATAN}, and the symmetrization rotation time, T_{ROT}, both equal the time for a CORDIC vector rotation, T_C. The total symmetrization time is

$$T_{SYM} = T_{ADD} + T_{ATAN} + T_{ROT} \approx 2T_C. \qquad (36)$$

The calculation of the diagonalization angle, θ_r, requires a shift, T_{SH}, after the inverse tangent computation and the initial additions. The two-sided rotation time, T_{T-S}, is equivalent to the time for two CORDIC rotations, or $2T_C$. Therefore, the total diagonalization time is

$$T_{DIAG} = T_{ADD} + T_{ATAN} + T_{SH} + T_{T-S} \approx 3T_C. \qquad (37)$$

These results can be combined to yield

$$T_{CSVD} = 5T_C. \qquad (38)$$

A reduction in latency can be achieved by pipelining the symmetrization angle inverse tangent module with the symmetrization rotation module. Since both modules utilize the same CORDIC angles, the intermediate results, δ_{SYM_i} and θ_{SYM_i}, can be processed by the rotation module after an initial sign test time, T_{ST}. Pipelining reduces T_{SYM} by approximately T_C and the final execution time for this method is

$$T_{CSVD} = 4T_C. \qquad (39)$$

The area used by this architecture can be expressed in terms of the area of a basic CORDIC processor, A_C. Each inverse tangent module has area, A_{ATAN}, equal to a CORDIC processor. The rotation modules require twice the area since they operate on two vectors. A first estimate of the total area is

$$A_{CSVD} = A_{SYM} + A_{DIAG} , \quad (40)$$

where

$$A_{SYM} = A_{ATAN} + A_{ROT} = A_C + 2A_C , \quad (41)$$

and

$$A_{DIAG} = A_{SH} + A_{ATAN} + A_{SH} + A_{T-S} = A_{SH} + A_C + A_{SH} + 2A_C . \quad (42)$$

Therefore,

$$A_{CSVD} = 6A_C + 2A_{SH} . \quad (43)$$

A reduction in the total area can be achieved by increasing the utilization of the hardware. For example, the shift required to produce θ_r can be performed by the inverse tangent module. Also, two inverse tangent modules may not be needed since the module that finds θ_{SYM} will be available to calculate θ_r. Similarly, the two-sided rotation can be performed by the symmetrization rotation module. The minimum hardware configuration is limited by the pipelined symmetrization section. The final total area for this architecture is

$$A_{CSVD} = A_{SYM} = 3A_C . \quad (44)$$

The reduction in area due to increased utilization is offset, however, by the increased complexity of the internal interconnection and control structures.

Approximation method

In the previous architecture, the diagonalization proceeded sequentially since the angle, $2\theta_r$, was first calculated, then multiplied by ½, and finally applied to rotate the matrix. This new architecture, shown in Figure 3, seeks to reduce T_{CSVD} by pipelining the second or diagonalization rotation. A simple one-half approximation is performed between the arctan and rotation modules. Since the CORDIC algorithm angles decrease by approximately one-half, the rotation module can choose the next angle in the sequence in order to perform a rotation by θ_r. This technique will allow pipelining which will reduce T_{CSVD} by approximately T_C and the execution time will be

$$T_{CSVD} = 3T_C . \quad (45)$$

The area requirements remain the same for this architecture and

$$A_{CSVD} = 3A_C . \quad (46)$$

Although T_{CSVD} is reduced, convergence of the SVD - Jacobi method may have been slowed due to the inexact computation of θ_r.

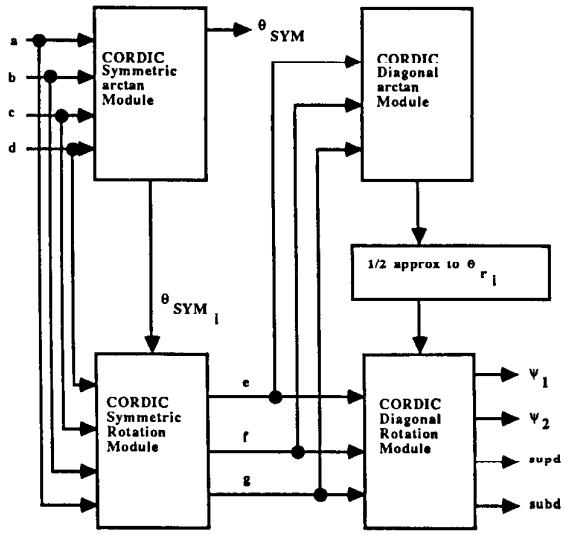

Figure 3. CORDIC SVD symmetrization, approximate diagonalization method.

Semi-parallel method

This architecture, shown in Figure 4, seeks to exploit parallelism in the computation of the rotation angles. In parallel with the computation of the symmetrization angle, θ_{SYM}, a second CORDIC inverse tangent module computes

$$\theta_{SUM} = (\theta_l + \theta_r) = \tan^{-1}\left(\frac{c+b}{d-a}\right) . \quad (47)$$

Once these two angles have been found, θ_r can be computed by a subtraction and a shift, since

$$\theta_{SUM} - \theta_{SYM} = (\theta_l + \theta_r) - (\theta_l - \theta_r) = 2\theta_r . \quad (48)$$

The parallel computation of the rotation angles overlaps the symmetrization and diagonalization processes and reduces the total time to

$$T_{CSVD} = 3T_C . \quad (49)$$

The area requirement increases for this architecture since an additional inverse tangent module is needed for the simultaneous calculation of θ_{SYM} and θ_{SUM}. The total area is

$$A_{CSVD} = 4A_C . \quad (50)$$

Although this architecture has reduced T_{CSVD} while preserving numerical accuracy, the area has enlarged. Since area is an important parameter for VLSI systems, a design which minimizes both area and time is desired.

CORDIC direct two angle architectures

The direct two angle method [8] calculates θ_l and θ_r by computing the inverse tangents of the data elements of M.

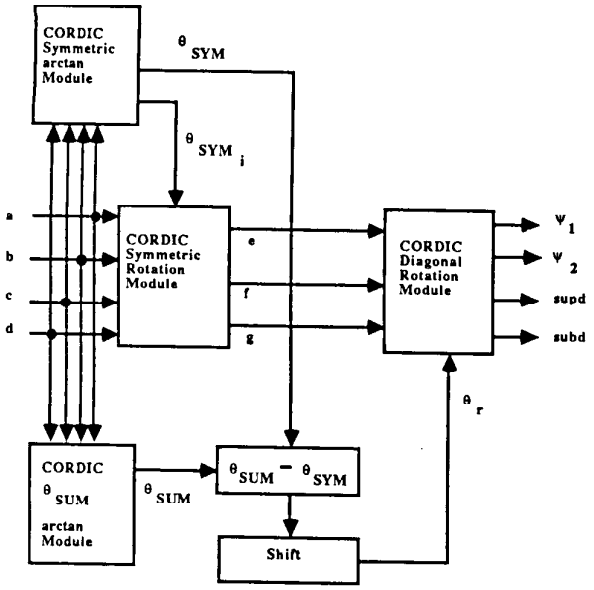

Figure 4. CORDIC SVD semi-parallel symmetrization, diagonalization method.

Given M and θ_{SUM} as defined in (4) and (47), θ_{DIFF} is then

$$\theta_{DIFF} = (\theta_r - \theta_l) = \tan^{-1}\left(\frac{c-b}{d+a}\right). \qquad (51)$$

The two angles, θ_l and θ_r, can then be separated from the sum and difference results and applied to the two-sided rotation module as in (2).

Parallel diagonalization method

In this architecture, shown in Figure 5, the calculation of θ_{SYM} is replaced by the calculation of θ_{DIFF}. Additionally, the entire symmetrization rotation is eliminated. These modifications allow the area of the processor to be reduced while preserving the time needed for computation. The algorithm can be summarized as follows:

Algorithm CORDIC SVD Parallel ( ):
begin
parallel do $\{b+c,\ c-b,\ d-a,\ d+a\}$;
parallel do begin
Find $\theta_{SUM} = (\theta_l + \theta_r)$;
Find $\theta_{DIFF} = (\theta_r - \theta_l)$;
end;
parallel do Separate θ_l, θ_r;
Use CORDIC rotation module to apply θ_l, θ_r;
end.

The time complexity of the complete CORDIC 2×2 SVD processor can be determined from the longest path. Initially, the sums and differences of the matrix elements of M need to be determined. These four additions can be done in parallel. Therefore, the preprocess time is $T_{PRE} = T_{ADD}$.

The angles θ_{SUM} and θ_{DIFF} are computed in parallel in $T_{ATAN} = T_C = n(T_{ADD} + T_{SH} + T_{ST})$ by two CORDIC modules. The separation of θ_l and θ_r can be computed in parallel using an adder followed by a shifter, $T_{SEP} = (T_{ADD} + T_{SH})$. Finally, the two-sided CORDIC rotation can be performed in $T_{T-S} = 2T_C$. The total time for a CORDIC 2×2 SVD, T_{CSVD}, is

$$T_{CSVD} = T_{PRE} + T_{ATAN} + T_{SEP} + T_{T-S}. \qquad (52)$$

This expression can be simplified to yield:

$$T_{CSVD} = 3n(T_{ADD} + T_{SH} + T_{ST}) + 2T_{ADD} + T_{SH} \approx 3T_C = O(T_{ADD}\, n). \qquad (53)$$

The area required by this architecture is approximately twice that of a single CORDIC processor. The calculation of θ_{SUM} and θ_{DIFF} uses two CORDIC modules. Also, these two modules can perform the additions and shifts that are required to prepare θ_l and θ_r. Finally, these modules will be available and can be reconfigured to compute the diagonalization of the 2×2 submatrix. Therefore, this architecture requires an area

$$A_{CSVD} = 2A_C. \qquad (54)$$

CORDIC SVD diagonalization module

In a prototype system, the CORDIC Parallel Diagonalization Method would be used since the least time and area are needed. The basic floor-plan of a VLSI implementation is

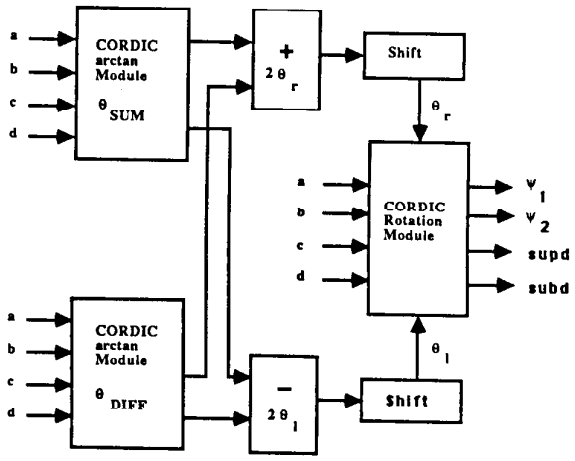

Figure 5. CORDIC SVD parallel diagonalization method.

shown in Figure 6. Three major sections are visible: two CORDIC processors, and an interconnection network. The CORDIC processors are based upon the design shown in Figure 1. The intra-module interconnection network will allow the same chip to function as both an angle solver and a rotation module, and will permit flexibility in designing, constructing, and reconfiguring a large array.

Finite state control for the interconnection network and for the SVD algorithm will be provided by a PLA. The array will be connected in a mesh configuration.[8] Each module will possess the necessary control for systolic I/O. A basic layout for an SVD array composed of CORDIC modules is given in Figure 7.

In order to reduce the execution time, attention must be paid to addition techniques, since each CORDIC processor will perform $O(n)$ additions for each 2×2 diagonalization. Several alternative addition algorithms can be utilized including ripple-carry, carry look-ahead, signed-digit, and on-line addition. Efficient methods for addition which ensure that the time for addition, T_{ADD}, can be minimized will be important for system implementation. Additionally, the CORDIC data paths could be modified to provide for a floating-point representation. Finally, methods for fault detection and reconfiguration will become important for large arrays of processors. All of these factors will have an effect upon the integration density achievable in VLSI.

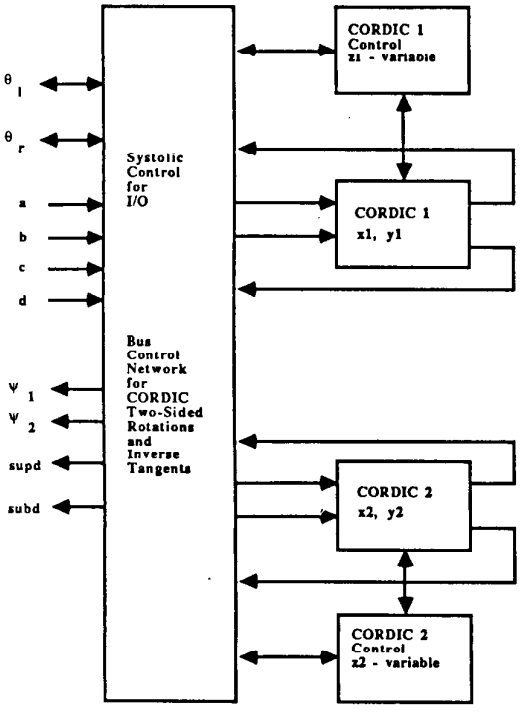

Figure 6. CORDIC SVD diagonalization module.

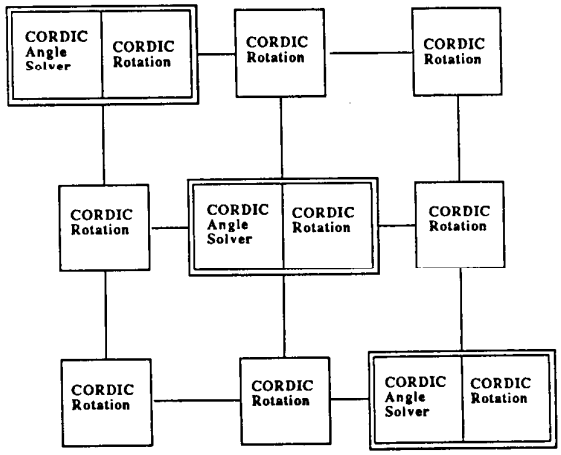

Figure 7. CORDIC SVD array architecture.

Summary

The CORDIC 2×2 SVD processor achieves a simple structure requiring a small number of functional units that are interconnected in a manner suitable for VLSI implementation. The Jacobi method for the SVD has been reviewed and the applicability of the CORDIC algorithms has been discussed. Novel architectures for a special purpose systolic processor have been presented and analyzed. The design objective has been the minimization of both area and time. The Parallel Diagonalization Method has area, $A_{CSVD} = 2A_C$, and time, $T_{CSVD} = 3T_C$, where A_C and T_C are the area and time for one CORDIC operation. respectively. A VLSI implementation of this architecture is planned as part of a prototype system.

Acknowledgements

This work was supported in part by the Army Research Office under contract DAAL 03–86–K–0109.

References

1. Sibul, L. H., "Application of Singular Value Decomposition to Adaptive Beamforming," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2, pp. 33.11.1-33.11.4 , San Diego, CA (March 1984).

2. Andrews, H. C. and Patterson, C. L., "Singular Value Decompositions and Digital Image Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-24,(1), pp. 26-53 (February 1976).

3. Speiser, J. M. and Whitehouse, H. J., "A Review of Signal Processing with Systolic Arrays," *Proc. SPIE Real-Time Signal Processing*, 431,(VI), pp. 2-6 , San Diego, CA (August 1983).

4. Finn, A., "Systolic Array Computation of the Singular Value Decomposition," Ph. D. Dissertation, School of Electrical Engineering. Cornell University, Ithaca, NY (May 1983).

5. Moreno, J., "Analysis of Alternatives for a Singular Value Decomposition Processor," Master's Dissertation, Computer Science Department, University of California at Los Angeles, Los Angeles, CA (October 1985).

6. Ahmed, H. M., "Signal Processing Algorithms and Architectures," Ph. D. Dissertation, Dept. of Electrical Engineering, Stanford University, Stanford, CA (June 1982).

7. Sibul, L. H. and Fogelsanger, A. L., "Application of Coordinate Rotation Algorithm to Singular Value Decomposition," *IEEE International Symposium on Circuits and Systems*, pp. 821-824 , Montreal, Canada (1984).

8. Brent, R. P., Luk, F. T., and Van Loan, C. F., "Computation of the Singular Value Decomposition Using Mesh-Connected Processors," *Journal of VLSI and Computer Systems*, 1,(3), pp. 242-270 (1985).

9. Golub, G. H. and Van Loan, C. F., *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD (1983).

10. Volder, J., "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, EC-8,(3), pp. 330-334 (Sept. 1959).

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

12. Haviland, G. L. and Tuszynski, A. A., "A CORDIC Arithmetic Processor Chip," *IEEE Transactions on Computers*, C-29,(2), pp. 68-79 (Feb. 1980).

13. Delosme, J. M., "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces," *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2, pp. 927-930 , Boston, MA (April 1983).

14. Despain, A. M., "Fourier Transform Computers Using CORDIC Iterations," *IEEE Transactions on Computers*, C-23,(10), pp. 993-1001 (October 1974).

15. Bridge, C., Fisher, P., and Reynolds, R., "Asynchronous Arithmetic Algorithms for Data-Driven Machines," *IEEE 5th Symposium on Computer Arithmetic*, pp. 56-62 , Ann Arbor, MI (May 1981).

16. Ullman, J. D., *Computational Aspects of VLSI*, Computer Science Press, Rockville, MD (1984).