JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 5, 271–290 (1988)

CORDIC Arithmetic for an SVD Processor

JOSEPH R. CAVALLARO AND FRANKLIN T. LUK

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

Received August 19, 1987

Arithmetic issues in the calculation of the Singular Value Decomposition (SVD) are discussed. Traditional algorithms using hardware division and square root are replaced with the special-purpose CORDIC algorithms for computing vector rotations and inverse tangents. The CORDIC 2 × 2 SVD processor can be twice as fast as one assembled from traditional hardware units. A CORDIC SVD processor array is suitable for VLSI implementation and is important for use in real-time signal processing applications.

1. INTRODUCTION

Recent advances in parallel architectures and VLSI have encouraged the use of special-purpose arithmetic techniques for the implementation of computationally complex scientific algorithms. Singular Value Decomposition (SVD) is an important algorithm for image processing [2] and is especially well suited to analyzing data matrices from sensor arrays [16].

Traditional SVD algorithms for use on uniprocessor systems rely heavily upon costly division and square root operations to compute the required rotation parameters. Special-purpose parallel architectures and numerical algorithms can increase the efficiency of the SVD by more effectively mapping the algorithm to hardware [18, 17].

The systolic array structure of Brent, Luk, and Van Loan [4] uses an expandable square array of simple 2 × 2 processors to compute the SVD of a large matrix. In this paper, a novel architecture for a CORDIC 2 × 2 processor is presented. In addition, a new scheme which simplifies CORDIC scale factor correction for the two-sided vector rotation is introduced. The reduction in the area and time complexity of the SVD processor through the use of the coordinate rotation algorithms (CORDIC) is also analyzed. The replacement of explicit multiplication, division, and square root units by CORDIC modules

produces a processor that is twice as fast as one constructed from modules based upon traditional radix-2 addition-oriented arithmetic algorithms. More importantly, the CORDIC modules produce a regular structure suitable for VLSI implementation.

2. SVD-Jacobi Method

The singular value decomposition of a $p \times p$ matrix M is given by

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

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 $\lceil p/2 \rceil \times \lceil p/2 \rceil$ simple 2×2 processors where the basic operation is the two-sided rotation of each 2×2 matrix.

2.1. *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 efficient computation of the rotation parameters is essential. Several methods are possible to solve this problem. The two-step method first applies $\theta_{SYM} = (\theta_l - \theta_r)$ to symmetrize M and then utilizes θ_r to diagonalize M. The direct two-angle method [4] calculates θ_l and θ_r by computing the inverse

tangents of the data elements of M. Given M as defined in (4), θ_{SUM} and θ_{DIFF} are

$$\theta_{SUM} = (\theta_l + \theta_r) = \tan^{-1}\left(\frac{c+b}{d-a}\right), \tag{5}$$

$$\theta_{DIFF} = (\theta_r - \theta_l) = \tan^{-1}\left(\frac{c-b}{d+a}\right). \tag{6}$$

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

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. However, these operations still involve costly multiplication, division, and square root.

With the CORDIC algorithms, the inverse tangent function is a primitive operation and the angles can be found explicitly, without penalty. If CORDIC processors are used, then the rotation parameters can be calculated from the inverse tangents of the elements of M. Also, vector rotations are primitive CORDIC operations and can replace traditional matrix–vector multiplication. 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. Thus, a general algorithm for a 2×2 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.
```

3. CORDIC Algorithms

The Coordinate Rotation Digital Computer algorithms (CORDIC) were first presented in 1959 by Volder [20]. Further theoretical work was done by Walther [21] in 1971 to show the applicability of CORDIC to various transcendental and hyperbolic functions. The CORDIC algorithms allow fast iterative hardware calculation of sin, cos, arctan, sinh, cosh, arctanh, product, quotient, and square root.

Recently, there has been renewed interest in the use of CORDIC algorithms for real-time signal processing [1], primarily due to the possibility of VLSI implementation [13]. The applicability of CORDIC to the basic operations in the SVD will be presented along with the limitations of the algorithm. The CORDIC SVD processor described here may be used as a math coprocessor or within a special-purpose systolic array.

3.1. *CORDIC Recurrence Equations*

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

$$\begin{bmatrix} x'_0 \\ y'_0 \end{bmatrix} = R(\theta) \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}. \tag{7}$$

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

$$\theta = \pm\theta_0 \pm \theta_1 \cdots \pm \theta_{n-1} = \sum_{i=0}^{n-1} \delta_i \theta_i, \tag{8}$$

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

$$\frac{x_{i+1}}{\cos\theta_i} = x_i + \delta_i y_i \tan\theta_i, \tag{9}$$

$$\frac{y_{i+1}}{\cos\theta_i} = y_i - \delta_i x_i \tan\theta_i. \tag{10}$$

The key contribution of Volder [20] and Walther [21] 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 θ_i

$= \tan^{-1}(2^{-i}) = 45°$. Again, for $i = 1$, $\theta_i = 26.7°$. Obviously, as i increases, θ_i decreases toward 0.

3.2. *CORDIC Scale Factor Correction*

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

$$x_{i+1} = k_i(x_i + \delta_i y_i 2^{-i}), \tag{11}$$

$$y_{i+1} = k_i(y_i - \delta_i x_i 2^{-i}). \tag{12}$$

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

$$K_n = \prod_{i=0}^{n-1} k_i = \prod_{i=0}^{n-1} \cos \theta_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}}. \tag{13}$$

The final CORDIC iteration equations which are to be implemented in hardware are

$$x_{i+1} = x_i + \delta_i y_i 2^{-i}, \tag{14}$$

$$y_{i+1} = y_i - \delta_i x_i 2^{-i}. \tag{15}$$

The desired final values of the CORDIC operation, x_{final} and y_{final}, are not readily obtained by these equations. Instead, after n iterations the values

$$x_n = \frac{x_{\text{final}}}{K_n}, \qquad y_n = \frac{y_{\text{final}}}{K_n}, \tag{16}$$

are determined. As a last step, a scale factor correction needs to be performed to yield

$$\begin{aligned} x_n K_{\text{SFC}} &= \left(\frac{x_{\text{final}}}{K_n}\right) K_{\text{SFC}} \approx C x_{\text{final}}, \\ y_n K_{\text{SFC}} &= \left(\frac{y_{\text{final}}}{K_n}\right) K_{\text{SFC}} \approx C y_{\text{final}}, \end{aligned} \tag{17}$$

where the scale factor correction constant is $K_{\text{SFC}} \approx CK_n$. The constant C is either 1 or a power of the machine radix, β, so that it can be easily cleared by a simple shift to yield x_{final} and y_{final}.

For the SVD processor, a CORDIC scale factor correction is required after the two-sided vector rotation. While a single CORDIC vector rotation module requires correction by K_n, a two-sided vector rotation module requires correction by K_n^2. The time complexity of the existing schemes will be described and a novel method which offers a factor of 2 speedup for the two-sided vector rotation will be presented.

The most direct scheme involves a multiplication by K_n using the CORDIC hardware in the linear mode, as mentioned by Walther [21]. This scheme has been dismissed as too costly since it may require n shifts and n additions. However, since K_n is a hardware constant, its binary representation can be used to determine which shifts and additions truly need to be performed. If the CORDIC processor is modified to perform special iterations for both the x and the y variables of the form

$$x \leftarrow x + x_n 2^{-j}, \tag{18}$$

then selected multiples of x_n and y_n are accumulated. The index j is used instead of i to distinguish the scale factor correction iterations from the normal CORDIC iterations. The total scale factor correction constant is

$$K_{\text{SFC}} = \sum_{j \in J} 2^{-j} \approx K_n. \tag{19}$$

If $n = 32$ bits, then set J has 18 elements and $J = \{1, 4, 5, 7, 8, 10, 11, 12, 14, 17, 18, 19, 21, 22, 24, 25, 27, 29\}$. After approximately $n/2$ special iterations, the scale factor would then be corrected and the final values determined.

Haviland and Tuszynski [13] proposed a method whereby the special scale factor correction iterations,

$$x \leftarrow x - x2^{-j}, \tag{20}$$

are performed for both the x and the y variables. This scheme also causes a multiplication of x_n and y_n by K_{SFC} and produces x_{final} and y_{final} as in (17). The scale factor correction constant will be

$$K_{\text{SFC}} = \prod_{j \in J} (1 - 2^{-j}) \approx K_n, \tag{21}$$

where $j \in J$ and $J = \{2, 3, 4, 7, 8, 10, 12, 14, 16, 19, 20, 22, 23, 24, 25, 31\}$. Note that there are 16 elements in this set or approximately $n/2$, and that the reduction in extra iterations is slight compared to direct multiplication.

Ahmed [1] seeks to make the constant, C, a power of the machine radix by repeating certain full CORDIC iterations. A final shift will then clear the remaining scale constant. This scheme operates differently from the previous

methods since it relies on using extra CORDIC rotation iterations instead of special multiplicative scaling iterations. Recall from (9) and (10) that at each CORDIC rotation iteration, x_j and y_j will increase by $(\cos\theta_j)^{-1}$ or k_j^{-1}. Therefore, these extra iterations will produce a correction constant

$$K_{\text{SFC}} = \prod_{j \in J} \frac{1}{\cos\theta_j} \approx 2K_n, \tag{22}$$

where $C = 2$ and the set J contains 28 elements or almost n. When K_{SFC} is applied to x_n, then

$$x_n K_{\text{SFC}} = \left(\frac{x_{\text{final}}}{K_n}\right) K_{\text{SFC}} \approx 2x_{\text{final}}, \tag{23}$$

and x_{final} can be found by a simple shift since the machine radix is chosen to be 2. As an extra benefit, this method extends the domain of convergence of the CORDIC algorithm. However, all CORDIC operations for the SVD can be made to fall within the basic CORDIC domain of convergence. Therefore, the time penalty caused by almost n extra iterations does not make Ahmed's method attractive for the SVD.

Delosme [6] combines the methods of Ahmed and Haviland and Tuszynski by repeating both CORDIC rotation iterations and special scale iterations and produces a low overhead scale factor correction scheme for a single CORDIC operation. When $n = 32$, this scheme requires seven extra CORDIC rotation iterations, and two special scale iterations or a total of about $n/4$. The variables x_n and y_n are again multiplied by $(\cos\theta_j)^{-1}$ when CORDIC iterations are applied and (1 ± 2^{-j}) when special scale factor iterations are applied. The scale factor constant,

$$\begin{aligned} K_{\text{SFC}} &= \left(\prod_{j \in J} \frac{1}{\cos\theta_j}\right)(1 - 2^{-2})(1 + 2^{-6}) \\ &\approx 2K_n, \end{aligned} \tag{24}$$

where $J = \{0, 1, 3, 5, 6, 8, 14\}$ and $C = 2$, also requires a final shift to yield x_{final} and y_{final} as in (23).

Each of the above schemes requires numerical methods to determine the appropriate sequence for a particular word length. The number of iterations is chosen to reduce the approximation error to less than 2^{-n}. Unfortunately, these schemes lack a systematic approach and are difficult to extend to the two-sided rotation required by the SVD. A new method is now presented for the two-sided rotation.

In the SVD processor, one CORDIC scale factor correction can be performed for the complete two-sided vector rotation instead of for each single vector rotation. Thus, the scale factor will be the square of the single rotation factor. From (13),

$$K_n^2 = \prod_{i=0}^{n-1} \frac{1}{(1 + 2^{-2i})}, \tag{25}$$

and the novel observation is made that each term resembles the special scale factor iterations shown in (20). If the CORDIC processor performs special iterations of the form

$$x \leftarrow x - x2^{-2j}, \tag{26}$$

similar to those of Haviland and Tuszynski, then the scale factor correction constant will be

$$K_{\text{SFC}}^2 = \prod_{j \in J} (1 - 2^{-2j}) \approx 2K_n^2, \tag{27}$$

where $C = 2$ and $J = \{1, 3, 5, \ldots, (2\lceil n/4 \rceil - 1)\}$ for $n > 0$. The scale factor is removed as in (17) and a final shift will cancel the above factor of 2. A total of only $\lceil n/4 \rceil$ extra iterations for the complete two-sided rotation is required since many terms cancel when the products are formed. The systematic calculation of the scale factor correction sequence for any value of n is also possible. This new method greatly improves the performance of the two-sided rotation module and results in a factor of 2 speedup over two applications of Delosme's single-rotation method.

3.3. *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:

$$z_{i+1} = z_i + \delta_i \theta_i. \tag{28}$$

The variable, z_i, contains the total rotation angle applied, θ_i is the current rotation angle increment, and $\delta_i = \pm 1$. Through the appropriate selection of each δ_i, 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).

3.3.1. *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

$$x_n = K_n(x_0 + y_0 \tan\theta), \tag{29}$$

$$y_n = K_n(y_0 - x_0 \tan\theta), \tag{30}$$

$$z_n = z_0 + \theta. \tag{31}$$

If, after n iterations, $y_n = 0$, and if $z_0 = 0$, then $\tan\theta = (y_0/x_0)$ and

$$z_n = \tan^{-1}\left(\frac{y_0}{x_0}\right). \tag{32}$$

Note that the scale factor K_n cancels from the calculation.

3.3.2. *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

$$x_n = K_n(x_0 + y_0 \tan(z_0)), \tag{33}$$

$$y_n = K_n(y_0 - x_0 \tan(z_0)). \tag{34}$$

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.

3.4. *Convergence Issues*

Walther [21] 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

$$\alpha_0 = \sum_{i=0}^{n-1} \theta_i. \tag{35}$$

If a nonrepetitive sequence of angles ($i = 0, \ldots, n-1$) 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 $i = 0, \ldots, n-1$, it is required that

$$\theta_i - \sum_{h=i+1}^{n-1} \theta_h < \theta_{n-1}. \tag{36}$$

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

$$\tan^{-1}(2^{-i}) < 2\tan^{-1}(2^{-(i+1)}). \tag{37}$$

A new extension to the inverse tangent algorithm is proposed here to enable the CORDIC processor to determine the principal value of the inverse tangent function. In the SVD algorithm, it is desirable to limit rotations to ±90° based upon (5) and (6). However, the CORDIC processor considers the entire unit circle in computing the inverse tangent, although circular mode convergence is limited to only ±99°. Therefore x_0 and y_0 values with vector representations in the second and third quadrants are transformed into the fourth and first quadrants, respectively. In this method, 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.

3.5. *Area and Time Complexity of a Basic CORDIC Processor*

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 [19]. 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 [1] shown in Fig. 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

$$A_C = A_{PLA} + A_{ROM} + 2A_{SH} + 3A_{ADD} + 3A_{REG}. \tag{38}$$

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 $O(n^2)$. Although other schemes which require

less area exist, a constant time shift would no longer be possible [19]. Therefore, the area complexity of an entire CORDIC module will be

$$A_C \approx 2A_{SH} = O(n^2). \tag{39}$$

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

$$x_i \leftarrow x_i + \delta_i \text{SHIFT}(y_i). \tag{40}$$

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

$$T_{Ci} = T_{ADD} + T_{SH} + T_{ST}, \tag{41}$$

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

$$T_C = n(T_{ADD} + T_{SH} + T_{ST}), \tag{42}$$

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 $T_{SH} \ll T_{ADD}$. In a two's complement fixed-point implementation, the sign test will determine whether addition or

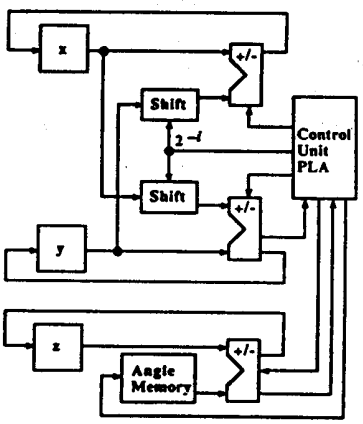

FIG. 1. Parallel fixed-point CORDIC processor.

subtraction is to be performed, and $T_{ST} \ll T_{ADD}$. From these assumptions, the limiting factor in a CORDIC SVD processor is the time needed to perform an n-bit addition, T_{ADD}. The time for a CORDIC operation depends linearly on the number of bits in the operands and

$$T_C \approx T_{ADD}n = O(T_{ADD}n). \tag{43}$$

If vector rotation is to be performed, then additional scale factor correction iterations need to be performed. For a one-sided vector rotation, an additional CORDIC scale factor correction time, $T_{SFC} \approx 1/4T_C$, is required using Delosme's method [6]. For the two-sided rotation, the method introduced here can be used and the scale factor correction time is $T_{SFC} = 1/8T_C$ per rotation. Therefore the total time for a CORDIC two-sided rotation, T_{T-S}, is

$$T_{T-S} = 2(T_C + T_{SFC}) = 2(T_C + \tfrac{1}{8}T_C) = 2.25T_C. \tag{44}$$

4. CORDIC SVD Processor Architecture

Four novel CORDIC architectures which perform variations on this algorithm with different time and area costs have been developed [5]. The first method, the Symmetrization Diagonalization Method, computes θ_{SYM} and θ_r and requires execution time $T_{CSVD} = 4T_C$ and area $A_{CSVD} = 3A_C$. This can be reduced in the Approximation Method to time $T_{CSVD} = 3T_C$ and area $A_{CSVD} = 3A_C$. The Semiparallel Method maintains the same time $T_{CSVD} = 3T_C$ but requires an increase in area to $A_{CSVD} = 4A_C$.

4.1. *Parallel Diagonalization Method*

The fourth method, the Parallel Diagonalization Method [5], which is based upon determining θ_{SUM} and θ_{DIFF} directly, results in a reduction in the time and area necessary for the 2×2 processor. In this architecture, shown in Fig. 2, 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

```
Algorithm CORDIC SVD Parallel ( ):
        begin
          parallel do {b + c, c − b, d − a, d + a};
          parallel do begin
                    Find θ_SUM = (θ_l + θ_r);
                    Find θ_DIFF = (θ_r − θ_l);
                    end;
```

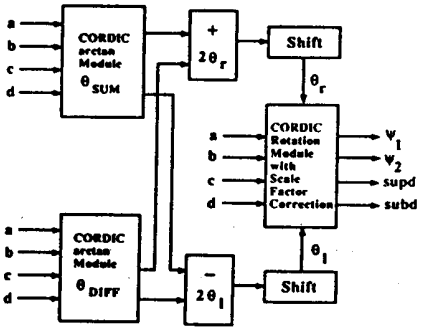

FIG. 2. CORDIC SVD parallel diagonalization method.

parallel do Separate θ_l, θ_r;
Apply θ_l, θ_r using CORDIC two-sided
rotation module;
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 with the new scale factor correction method can be performed in $T_{T-S} = 2.25T_C$. The total time for a CORDIC 2 × 2 SVD, T_{CSVD}, is

$$T_{CSVD} = T_{PRE} + T_{ATAN} + T_{SEP} + T_{T-S}. \tag{45}$$

This expression can be simplified to yield

$$\begin{aligned} T_{CSVD} &= 3.25n(T_{ADD} + T_{SH} + T_{ST}) + 2T_{ADD} + T_{SH} \\ &\approx 3.25T_C = O(T_{ADD}n). \end{aligned} \tag{46}$$

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 and scale factor correction of the 2×2 submatrix. Therefore, this architecture requires an area

$$A_{CSVD} = 2A_C. \tag{47}$$

5. Comparison with Traditional Arithmetic Techniques

The SVD can also be computed using traditional multiplication, division, and square root. An algorithm to calculate the rotation parameters is given in Brent, Luk, and Van Loan [4]. This algorithm produces the sine and cosine pairs for the diagonalization. An analysis of this algorithm shows that the execution time, T_{TCS}, is

$$T_{TCS} = 5T_{ADD} + 4T_{MULT} + 3T_{DIV} + 2T_{SQRT}, \tag{48}$$

and the area, A_{TCS}, is

$$A_{TCS} = 4A_{ADD} + 4A_{MULT} + A_{DIV} + A_{SQRT}, \tag{49}$$

if the maximum amount of parallelism is exploited.

A two-sided multiplication then follows to diagonalize the matrix. With parallel hardware, this can be done in time, T_{TT-S}, equal to

$$T_{TT-S} = 2T_{MULT} + 2T_{ADD}, \tag{50}$$

with area, A_{TT-S}, equal to

$$A_{TT-S} = 8A_{MULT} + 4A_{ADD}. \tag{51}$$

The total execution time for a 2×2 SVD on "traditional" hardware, T_{TSVD}, is

$$T_{TSVD} = 7T_{ADD} + 6T_{MULT} + 3T_{DIV} + 2T_{SQRT}. \tag{52}$$

The total area required to support the parallelism in the matrix multiplication, A_{TSVD}, is

$$A_{TSVD} = 4A_{ADD} + 8A_{MULT} + A_{DIV} + A_{SQRT}. \tag{53}$$

5.1. *Division, Square Root, and Multiplication Algorithms*

In order to compare the area and time complexity of the traditional hardware with the CORDIC hardware, the traditional radix-2 addition-based division, square root, and multiplication algorithms are discussed. The following

algorithms are chosen to represent current widely implemented algorithms found on general-purpose processor systems. The time complexity of a CORDIC operation has been shown to be related to the number of n-bit additions, T_{ADD}. The traditional division, square root, and multiplication algorithms can also be analyzed and expressed in terms of the number of n-bit additions. Ripple-carry addition will be assumed throughout the initial area and time complexity analysis for consistency and simplicity.

The nonrestoring division algorithm [15] is an iterative technique that is similar to the CORDIC algorithm. For an n-bit result, a total of n additions/subtractions and shifts must be performed. Thus, the time for division is $T_{DIV} \approx nT_{ADD}$.

A similar algorithm exists for the computation of square roots [14]. The basic algorithm requires two additions and a shift at each of n iterations. However, if a hardware enhancement is performed, this result can be reduced to a single addition and shift at each iteration [3]. Therefore the time for square root is $T_{SQRT} \approx nT_{ADD}$.

For multiplication, the most basic approach requires n shifts and n additions, such that $T_{MULT} \approx nT_{ADD}$. However, the total multiplication time can be reduced with special-purpose multiplication arrays, such as Wallace trees, which are composed of carry-save adder cells. In a basic radix-2 tree carry-save multiplier, fewer than n additions are performed. The total multiplication time can be reduced to the time for one full $2n$-bit carry-propagate addition plus $f \approx \log_{3/2} n$ full-adder time steps [9]. A ripple-carry adder can be used to perform the $2n$-bit addition in the time required for two n-bit additions, or $2T_{ADD}$. Also, the $f \approx \log_{3/2} n$ full-adder time steps are bounded by one n-bit addition time, T_{ADD}. Therefore, the time for multiplication is $T_{MULT} \approx 3T_{ADD}$. However, $O(n^2)$ full-adder cells are required for the tree multiplier [15].

5.2. *Area and Time Comparisons*

A comparison can be made between the CORDIC architectures and traditional arithmetic techniques. The total time for the traditional hardware can be related to the number of n-bit additions as in the CORDIC analysis. Thus, from the results of the previous section, the total traditional time for a 2×2 SVD is

$$\begin{aligned} T_{TSVD} &= 7T_{ADD} + 6T_{MULT} + 3T_{DIV} + 2T_{SQRT} \\ &\approx 7T_{ADD} + 6nT_{ADD} + 3nT_{ADD} + 2nT_{ADD} \approx 11.3T_{ADD} \end{aligned} \tag{54}$$

for $n \approx 24$ and for the basic traditional hardware. If basic tree carry-save multipliers are used, then the total time is reduced to

$$T_{TSVD} = 7T_{ADD} + 6(3T_{ADD}) + 3nT_{ADD} + 2nT_{ADD} \approx 6nT_{ADD} \tag{55}$$

for $n \approx 24$. The speedup factor, sp, can be defined as the ratio of the traditional 2×2 SVD time to the CORDIC 2×2 SVD time. If basic multipliers are used, then the speedup is sp ≈ 3.5. However, if basic tree carry-save multipliers are used, then the speedup is

$$\text{sp} = \frac{T_{\text{TSVD}}}{T_{\text{CSVD}}} \approx \frac{6nT_{\text{ADD}}}{3.25nT_{\text{ADD}}} \approx 2. \tag{56}$$

Thus, a CORDIC 2×2 SVD processor is approximately twice as fast as one built from traditional arithmetic units.

The total traditional SVD area, A_{TSVD}, is determined by the $O(n^2)$ full-adder cells needed for the Wallace multiplication tree. Therefore, the area of the CORDIC SVD processor and the area of a traditional arithmetic SVD processor are of the same order. The true area benefit of the CORDIC architecture is the regular design which eases VLSI implementation.

6. CORDIC SVD Diagonalization Module

In a prototype system, the CORDIC Parallel Diagonalization Method would be used since the least time and area are needed and the structure is regular. The basic floor plan of a VLSI implementation is shown in Fig. 3. Three major sections are visible: two CORDIC processors, and an interconnection network.

6.1. *Module Organization*

The CORDIC processors are based upon the design shown in Fig. 1. The intramodule interconnection network will allow the same chip to function as

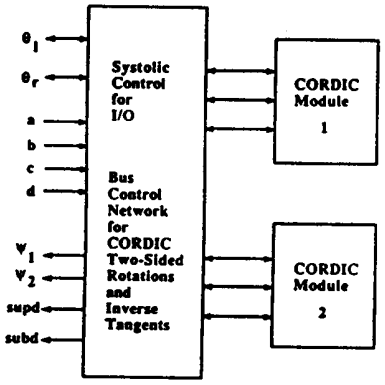

FIG. 3. CORDIC SVD diagonalization module.

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 [4]. 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 Fig. 4.

6.2. *Internal Data Representation*

In a fixed-point implementation, the number of bits in the internal data representation must be chosen to prevent loss of significance due to rounding and overflow. In a prototype implementation, all quantities will be considered to be fractions.

In order to determine the number of bits necessary to prevent overflow, the following analysis is presented. The internal registers in an SVD processor must be able to store the largest calculated singular value. Therefore, worst-case bounds on the largest singular value of the input matrix are necessary.

If the entire $p \times p$ matrix, M, is divided by the largest value, m_{ij}, then all of the elements of M will be fractions. In the worst case, all elements of M will remain equal to unity, and rank $(M) = 1$. From the Frobenius norm, the singular values can be determined since

$$\|M\|_F^2 = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_p^2 = p^2. \tag{57}$$

Since rank $(M) = 1$, the singular values are $\sigma_2 = \cdots = \sigma_p = 0$ and $\sigma_1 = p$. In order to prevent overflow, $\log_2 p$ extra bits will be needed to store the largest singular value in both CORDIC and traditional hardware implementations.

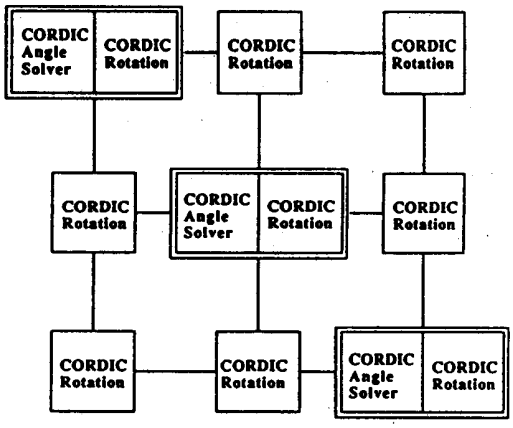

FIG. 4. CORDIC SVD array architecture.

The CORDIC algorithms internally consist of a sequence of shifts and additions. In order to prevent round-off errors from contaminating the final result, at least $\log_2 n$ additional low-order bits are necessary for intermediate values [21]. In traditional hardware implementations, the internal data paths of multiplication, division, and square root units use $2n$ bits for the storage of internal intermediate values. The internal intraprocessor data path should contain N_{int} bits, to preserve n bits of significance. For a fixed-point CORDIC implementation, N_{int} will be

$$N_{int} = n + \log_2 n + \log_2 p. \tag{58}$$

However, the external interprocessor data path will only need to contain $N_{ext} = n + \log_2 p$ bits to prevent overflow.

The accuracy of the fixed-point CORDIC processor can be confirmed through simulation of the algorithm. A computer simulation of a CORDIC processor with fixed-point data paths has been performed. The results verify the necessity of $\log_2 n$ additional low-order bits to control round-off errors. For example, if $n = 16$, then 20-bit arithmetic is necessary. Simulation and error bounds for SVD architectures have also been presented by Duryea [7].

6.3. *Implementation Issues*

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. Efficient methods for addition which minimize the time for addition, T_{ADD}, will be important for system implementation. In this paper, basic ripple-carry addition has been assumed. Therefore, the time for addition, T_{ADD}, equals $O(n)$, where n is the number of bits in the operands. The speed of the CORDIC processor can be further enhanced by using carry look-ahead addition, where $T_{ADD} = O(\log n)$. The replacement of ripple-carry adders with carry look-ahead adders will change the value of the constant, T_{ADD}. The basic time complexity analysis presented here will be preserved since both the CORDIC and the conventional units would be similarly modified.

Furthermore, 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.

7. SUMMARY

The SVD is a computationally complex algorithm that can benefit from special-purpose arithmetic algorithms. The CORDIC algorithms have been

shown to efficiently produce a 2×2 SVD processor architecture which can be implemented in VLSI. The CORDIC processor has been enhanced through improvements in the scale factor correction scheme for the two-sided vector rotation. The novel architecture of the CORDIC Parallel Diagonalization Method, which has been presented, has area, $A_{CSVD} = 2A_C$, and time, $T_{CSVD} = 3.25T_C$, where A_C and T_C are the area and time for one CORDIC operation, respectively. The structure is simple and more regular than a design using traditional multiplication, division, and square root cells and results in a factor of 2 speedup. A VLSI implementation of this architecture is planned as part of a prototype system.

ACKNOWLEDGMENTS

This work was supported in part by the Army Research Office under Contract DAAL 03-86-K-0109. The first author was also supported by an IBM Graduate Fellowship. The authors also thank Miloš Ercegovac for helpful discussions.

Note Added in Proof. The desire to improve the efficiency of the CORDIC algorithms has led to the application of on-line arithmetic [8]. The CORDIC SVD architecture can be modified to exploit internal pipelining through the use of on-line addition. The internal pipelining results in new, reduced time complexity bounds for the total CORDIC SVD execution time, T_{CSVD}. Recently, Ercegovac and Lang [10, 11] have developed redundant and on-line CORDIC schemes for the SVD. These schemes may lead to a speedup of about 4 over the conventional CORDIC implementation presented here. However, the simplicity of the CORDIC architecture, which is vital to VLSI implementation, may be lost due to the increased low-level complexity of the redundant CORDIC processor. Also, a new binary division algorithm [12] has recently been proposed. This division algorithm may improve the time performance of the traditional SVD algorithm and reduce the speedup afforded by the CORDIC SVD processor. These new results are encouraging since they highlight the potential for improvement to a CORDIC SVD processor implementation.

REFERENCES

1. Ahmed, H. M. Signal processing algorithms and architectures. Ph.D. dissertation, Department of Electrical Engineering, Stanford University, Stanford, CA, June 1982.
2. Andrews, H. C., and Patterson, C. L. Singular value decompositions and digital image processing. *IEEE Trans. Acoust. Speech Signal Process.* ASSP-24, 1 (Feb. 1976), 26–53.
3. Bannur, J., and Varma, A. The VLSI implementation of a square root algorithm. *IEEE 7th Symp. Computer Arithmetic*, Urbana, IL, June 1985, pp. 159–165.
4. Brent, R. P., Luk, F. T., and Van Loan, C. F., Computation of the singular value decomposition using mesh-connected processors. *J. VLSI Comput. Systems* 1, 3, (1985), 242–270.
5. Cavallaro, J. R., and Luk, F. T. Architectures for a CORDIC SVD processor. *Proc. SPIE Real-Time Signal Processing IX*, San Diego, CA, Aug. 1986, Vol. 698, pp. 45–53.
6. Delosme, J.-M. VLSI implementation of rotations in pseudo-Euclidean spaces. *IEEE Int. Conf. Acoustics, Speech and Signal Processing*, Boston, MA, Apr. 1983, Vol. 2, pp. 927–930.
7. Duryea, R. A. Finite precision arithmetic in singular value decomposition processors. Ph.D. dissertation, School of Electrical Engineering, Cornell University, Ithaca, NY, Aug. 1987.

8. Ercegovac, M. D., On-line arithmetic: An overview. *Proc. SPIE Real-Time Signal Processing VII*, San Diego, CA, Aug. 1984, Vol. 495, pp. 86–93.
9. Ercegovac, M. D., and Lang, T. *Digital Systems and Hardware/Firmware Algorithms*. Wiley, New York, 1985.
10. Ercegovac, M. D., and Lang, T. On-line schemes for computing rotation angles for SVDs. *Proc. SPIE Advanced Algorithms and Architectures for Signal Processing II*, San Diego, CA, Aug. 1987, Vol. 826, pp. 160–169.
11. Ercegovac, M. D., and Lang, T. Redundant and on-line CORDIC: Application to matrix triangularization and SVD. Report CSD-870046, Computer Science Department, UCLA, Los Angeles, CA, Sept. 1987.
12. Ercegovac, M. D., Lang, T., Nash, J. G., and Chow, L. P. An area–time efficient binary divider. *IEEE Int. Conf. on Computer Design*, Rye Brook, NY, Oct. 1987, pp. 645–648.
13. Haviland, G. L., and Tuszynski, A. A. A CORDIC arithmetic processor chip. *IEEE Trans. Comput.* C-29, 2 (Feb. 1980), 68–79.
14. Majerski, S. Square-rooting algorithms for high-speed digital circuits. *IEEE Trans. Comput.* C-34, 8 (Aug. 1985), 724–733.
15. Scott, N. R., *Computer Number Systems and Arithmetic*. Prentice-Hall, Englewood Cliffs, NJ, 1985.
16. Sibul, L. H. Application of singular value decomposition to adaptive beamforming. *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, San Diego, CA, March 1984, Vol. 2, pp. 33.11.1–33.11.4.
17. Sibul, L. H., and Fogelsanger, A. L. Application of coordinate rotation algorithm to singular value decomposition. *IEEE Int. Symp. Circuits and Systems*, Montreal, Canada, 1984, pp. 821–824.
18. Speiser, J. M., and Whitehouse, H. J. A review of signal processing with systolic arrays. *Proc. SPIE Real-Time Signal Processing VI*, San Diego, CA, Aug. 1983, Vol. 431, pp. 2–6.
19. Ullman, J. D. *Computational Aspects of VLSI*. Computer Science Press, Rockville, MD, 1984.
20. Volder, J. The CORDIC trigonometric computing technique. *IRE Trans. Electronic Comput.* EC-8, 3 (Sept. 1959), 330–334.
21. Walther, J. S. A unified algorithm for elementary functions. *AFIPS Spring Joint Computer Conf.*, 1971, pp. 379–385.