The CORDIC Householder Algorithm

Shen-Fu Hsiao Jean-Marc Delosme

Department of Electrical Engineering
Yale University
New Haven, CT 06520

Abstract

A novel n-dimensional (n-D) CORDIC algorithm, for Euclidean and pseudo-Euclidean rotations, is proposed. The new algorithm is closely related to Householder transformations. It is shown to converge faster than CORDIC algorithms developed earlier for $n = 3$ and 4. Processor architectures for the algorithm are presented. The area and time performance of n-D CORDIC processors are evaluated. For a comparable time performance, the processors require significantly less area than parallel Householder processors. Furthermore, arrays of n-D Euclidean CORDIC processors are shown to speed up the QR decomposition of rectangular matrices by a factor of $n-1$ with respect to a 2-D CORDIC processor array.

1 Introduction

1.1 2-D CORDIC Algorithms

The COordinate Rotation DIgital Computer (CORDIC), introduced by Volder [13] and extended later by Walther [15], is an algorithm operating on 2-dimensional (2-D) vectors to perform rotations using simple arithmetic primitives. The algorithm has several variants and can be used to calculate a variety of elementary functions. While the traditional approach to vector rotation calls for square-root extraction, division, and multiplication, the CORDIC algorithm requires only additions and shifts. The algorithm approximates a 2-D rotation R_2 (Euclidean or hyperbolic) by the product $\prod_{i=1}^{p} R_{2,i}$ of p elementary rotations

$$R_{2,i} = (1+\sigma t_i^2)^{-1/2} U_{2,i},\ U_{2,i} = \begin{pmatrix} 1 & \delta_i t_i \\ -\sigma\delta_i t_i & 1 \end{pmatrix}, \tag{1}$$

where $U_{2,i}$ is called the unscaled part of $R_{2,i}$,

$$\sigma = \begin{cases} 1 & \text{for a Euclidean rotation} \\ -1 & \text{for a hyperbolic (or pseudo-Euclidean) rotation,} \end{cases}$$

and t_i is a non-positive power of 2. When $t_{i+1} = t_i$, two consecutive rotations have the same magnitude; this is called a 'repetition'.

When evaluating the angle of a vector $[x_1\ x_2]^T$ with the first axis, a sequence of unscaled rotations $U_{2,i}$ is applied to bring the vector along the direction $[1\ 0]^T$ (evaluation process). The signs δ_i are selected according to $\delta_i = sign(x_{1,i-1} \cdot x_{2,i-1})$, where $x_{j,i-1}$ denotes the jth component of the vector at the beginning of iteration i and, initially, $x_{j,0} = x_j$. The sequence $\delta_i, 1 \le i \le p$, thus obtained may be used to rotate other vectors by the same amount (application process). Like the multiplication by the unscaled rotations $U_{2,i}$, multiplication by the scaling factor $\prod_{i=1}^{p}(1+\sigma t_i^2)^{-1/2}$ is implemented by additions and shifts, see *e.g.* [9]. While no repetition is needed for the 2-D Euclidean CORDIC algorithm, it is necessary in the hyperbolic case to have a repetition for $t_i = 2^{-4}, 2^{-13}, 2^{-40}, \ldots, 2^{-k}, 2^{-(3k+1)}, \ldots$ [15]. Fig. 1 illustrates the 2-D CORDIC processor architecture.

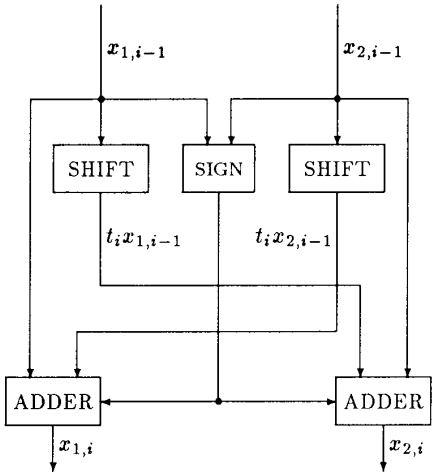

Figure 1: The architecture of a 2-D CORDIC processor.

1.2 3-D and 4-D CORDIC Algorithms

One trivial way of rotating an n-dimensional vector is to apply $(n-1)$ 2-D CORDIC rotations on $\lfloor n/2 \rfloor$ 2-D CORDIC processors in the time of $\lceil \log_2 n \rceil$ 2-D CORDIC rotations. For example, in order to align a 4-D vector with the $[1\ 0\ 0\ 0]^T$ direction, two 2-D

CORDIC rotations are applied in parallel to zero out the second and fourth components, followed by another 2-D CORDIC rotation to zero out the third component. Delosme *et al.* [6][7] have proposed faster 3-D and 4-D CORDIC algorithms which require a number of iterations only slightly larger than for a single 2-D CORDIC rotation. These algorithms are derived from the rational representation theorem of Cayley. According to that theorem, any n-D Euclidean rotation R_n with no eigenvalue equal to -1, can be expressed as

$$R_n = (I - T_n)(I + T_n)^{-1}$$

where T_n is an $n \times n$ skew-symmetric matrix.

The matrix $T_{3,i}$ associated to a 3-D elementary rotation matrix was selected in [6] as

$$T_{3,i} = \begin{pmatrix} 0 & -\gamma_i t_i & \delta_i t_i \\ \gamma_i t_i & 0 & -\gamma_i t_i \\ -\delta_i t_i & \gamma_i t_i & 0 \end{pmatrix}, \ \gamma_i, \delta_i \in \{1, -1\}, \tag{2}$$

where t_i is a negative power of 2. The corresponding elementary rotation matrix is

$$R_{3,i} = (I - T_{3,i})(I + T_{3,i})^{-1} = (1 + 3t_i^2)^{-1} \times \begin{pmatrix} 1 - t_i^2 & 2\gamma_i t_i + 2t_i^2 & -2\delta_i t_i + 2t_i^2 \\ -2\gamma_i t_i + 2t_i^2 & 1 - t_i^2 & 2\gamma_i t_i + 2t_i^2 \\ 2\delta_i t_i + 2t_i^2 & -2\gamma_i t_i + 2t_i^2 & 1 - t_i^2 \end{pmatrix}. \tag{3}$$

Its rotation axis is $[\gamma_i \ \delta_i \ \gamma_i]^T$, that is, either $[1\ 1\ 1]^T$ or $[1\ -1\ 1]^T$. When evaluating a rotation bringing a vector $[x_1 \ x_2 \ x_3]^T$ along the first axis $[1\ 0\ 0]^T$, the signs γ_i and δ_i are selected according to the control law

$$\begin{aligned} \gamma_i &= sign(x_{1,i-1} \cdot x_{2,i-1}), \\ \delta_i &= -sign(x_{1,i-1} \cdot x_{3,i-1}), \end{aligned} \tag{4}$$

where $x_{j,i-1}$ denotes the jth component of the vector at the beginning of iteration i.

In the 4-D case, the 4×4 skew-symmetric matrix $T_{4,i}$ defined as

$$T_{4,i} = \begin{pmatrix} 0 & \gamma_i t_i & \delta_i t_i & \epsilon_i t_i \\ -\gamma_i t_i & 0 & -\epsilon_i t_i & \delta_i t_i \\ -\delta_i t_i & \epsilon_i t_i & 0 & -\gamma_i t_i \\ -\epsilon_i t_i & -\delta_i t_i & \gamma_i t_i & 0 \end{pmatrix} \tag{5}$$

was selected in [6] [7]. Because the rotation matrix

$$\hat{R}_{4,i} = (I - T_{4,i})(I + T_{4,i})^{-1}$$

is the square of the matrix

$$R_{4,i} = \frac{1}{\sqrt{1 + 3t_i^2}} \begin{pmatrix} 1 & -t_i\gamma_i & -t_i\delta_i & -t_i\epsilon_i \\ t_i\gamma_i & 1 & t_i\epsilon_i & -t_i\delta_i \\ t_i\delta_i & -t_i\epsilon_i & 1 & t_i\gamma_i \\ t_i\epsilon_i & t_i\delta_i & -t_i\gamma_i & 1 \end{pmatrix}, \tag{6}$$

which has a simpler unnormalized part than $\hat{R}_{4,i}$, the matrices defined by equation (6) were selected as 4-D CORDIC elementary rotation matrices in [6] [7].

To prove the convergence of an n-D CORDIC algorithm over a given range, one must consider all the vectors in that range, which is a cone with vertex the origin in n-D space, and show that the sequence of cones which consist of all the vectors after each iteration converges toward the first axis. We wrote a program to compute these cones, determine—in case repetitions are needed—for which magnitudes t_i rotations should be repeated, and prove the convergence of the 3-D and 4-D CORDIC algorithms [7]. In the 3-D case, at the first iteration, the whole 3-D space C_0 is partitioned into the the 8 canonical orthants $C_0^1, C_0^2, \cdots, C_0^8$, and each of them is rotated by $R_{3,1}$ with the control signs selected according to equation (4). The union of these rotated orthants forms a cone C_1 with vertex the origin. At the second iteration, C_1 is partitioned into 8 sub-cones $C_1^1, C_1^2, \cdots, C_1^8$ corresponding to the 8 possible assignments for the control signs. Each sub-cone is rotated by $R_{3,2}$ with the appropriate control signs. The union of the rotated sub-cones forms a cone C_2. The procedure is repeated on C_2, and so forth for every iteration. The convergence behavior can be characterized by the rays of the cones C_i which are the farthest away from the first axis in some norm. The convergence of the 4-D CORDIC algorithm is analyzed similarly except that the cones C_i are partitioned into 16, instead of 8, sub-cones at each iteration. From these computations, we found out that it is indeed necessary to perform several repetitions to obtain convergent algorithms for R_3 and R_4. In [7], the repetitions were selected to achieve the fastest convergence rate.

These earlier studies did not provide a systematic method for constructing the n-D CORDIC elementary rotation matrices for arbitrary n. In Section 2, we propose a solution to that problem, applicable to Euclidean and pseudo-Euclidean CORDIC spaces of arbitrary dimension. The solution employs elementary rotations which are essentially Householder reflections. In Section 3, two CORDIC processor architectures are proposed. Their performance compares favorably with that of a parallel implementation of the Householder transformation. Section 4 develops an example indicative of the increase in throughput achievable in application-specific array processors when n-D CORDIC processors, with $n > 2$, are used in lieu of 2-D CORDIC processors.

2 CORDIC Householder Algorithm

2.1 The 3-D Euclidean Case

As mentioned above, the 3-D elementary rotation $R_{3,i}$ in equation (3) has either $[1\ 1\ 1]^T$ or $[1\ -1\ 1]^T$ as rotation axis. If instead the rotation axis is chosen to be either $[0\ 1\ 1]^T$ or $[0\ 1\ -1]^T$, the matrix $T_{3,i}$ is replaced by

$$\tilde{T}_{3,i} = \begin{pmatrix} 0 & -t_i s_{1,i} & -t_i s_{2,i} \\ t_i s_{1,i} & 0 & 0 \\ t_i s_{2,i} & 0 & 0 \end{pmatrix}, \tag{7}$$

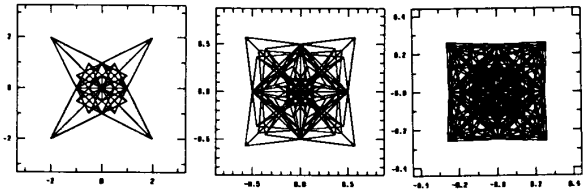

Figure 2: Domains after the first three iterations of $\bar{R}_3$.

and the elementary rotation matrices become

$$\bar{R}_{3,i} = (1+2t_i^2)^{-1}\times \begin{pmatrix} 1-2t_i^2 & 2t_i s_{1,i} & 2t_i s_{2,i} \\ -2t_i s_{1,i} & 1 & -2t_i^2 s_{1,i} s_{2,i} \\ -2t_i s_{2,i} & -2t_i^2 s_{2,i} s_{1,i} & 1 \end{pmatrix}, \quad (8)$$

with signs selected during the evaluation process according to

$$s_{1,i} = sign(x_{1,i-1} \cdot x_{2,i-1}), \quad s_{2,i} = sign(x_{1,i-1} \cdot x_{3,i-1}). \quad (9)$$

The matrix $\bar{R}_{3,i}$ is simpler than $R_{3,i}$. Moreover, as is discussed next, no repetition is necessary with this new family of elementary rotations.

Using the method outlined in Section 1, we can determine whether any repetition is needed and prove the convergence of the new 3-D algorithm by determining the cone C_i of all the rotated vectors after each iteration i. The cones C_i have the origin for vertex and surround the first axis. The intersections of the cones C_1, C_2 and C_3 with the plane $x_1 = 1$ are shown in Fig. 2. The cones are invariant under a permutation of x_2 and x_3. The new 3-D CORDIC algorithm, $\bar{R}_3$, converges slightly faster than R_3 since it does not require any repetition while R_3 calls for the repetition of the iterations with $t_i = 2^{-4}, 2^{-7}, 2^{-12}, 2^{-22}, \cdots$ [7]. Furthermore, the domain obtained by intersecting the cone C_i with the plane $x_1 = 1$ is bounded by a square box with half-side $2t_i/(1-2t_i^2)$.

2.2 The General n-D Case

The pseudo-norm $\|x\|_{\Sigma_n}$ of an n-D vector x in pseudo-Euclidean space is defined as $\|x\|_{\Sigma_n}^2 = x^T \Sigma_n x$, where Σ_n is a signature matrix of the form

$$\Sigma_n = \begin{pmatrix} 1 & 0 & 0 & \cdot & 0 \\ 0 & \sigma_1 & 0 & \cdot & 0 \\ 0 & 0 & \sigma_2 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \sigma_{n-1} \end{pmatrix}, \ \sigma_i \in \{-1, 1\}. \quad (10)$$

Without loss of generality, we will assume the values of σ_i ordered according to:

$$\sigma_i = \begin{cases} 1 & 1 \le i \le k-1 \\ -1 & k \le i \le n-1 \end{cases}. \quad (11)$$

If $k = n$, $\Sigma_n = I_n$ and $\|x\|_{\Sigma_n}$ is the Euclidean norm of the vector x. We shall view the Euclidean norm (and rotations) as a particular case of pseudo-Euclidean norm (and rotations).

For an $n \times n$ matrix H_n to represent a pseudo-Euclidean rotation, it should preserve the pseudo-norm of an arbitrary vector x, *i.e.*,

$$(H_n x)^T \Sigma_n (H_n x) = x^T \Sigma_n x.$$

Since the above equation is true for any vector, H_n must satisfy

$$H_n^T \Sigma_n H_n = \Sigma_n. \quad (12)$$

i.e., it must be orthogonal with respect to Σ_n.

If $\bar{T}_{3,i}$ in equation (7) is generalized to an n-dimensional space with signature matrix Σ_n, *i.e.*,

$$\bar{T}_{n,i} = S_{n,i}\Sigma_n \begin{pmatrix} 0 & -t_i & -t_i & \cdot\cdot & -t_i \\ t_i & 0 & 0 & \cdots & 0 \\ t_i & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ t_i & 0 & 0 & \cdots & 0 \end{pmatrix} S_{n,i}, \quad (13)$$

where the signs in the matrix $S_{n,i}$

$$S_{n,i} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & s_{1,i} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & s_{n-1,i} \end{pmatrix}, \quad (14)$$

are selected during the evaluation process according to

$$s_{j,i} = \sigma_i \cdot sign(x_{1,i-1} \cdot x_{j+1,i-1}), \quad (15)$$

the matrix $H_{n,i} = (I - \bar{T}_{n,i})(I + \bar{T}_{n,i})^{-1}$ is equal to

$$H_{n,i} = (1+ct_i^2)^{-1}\times \begin{pmatrix} 1-ct_i^2 & 2t_i & \cdots & 2t_i \\ -2t_i & 1+(c-2)t_i^2 & \cdots & -2t_i^2 \\ \vdots & \vdots & \cdots & \vdots \\ -2t_i & -2t_i^2 & \cdots & 1+(c-2)t_i^2 \\ 2t_i & 2t_i^2 & \cdots & 2t_i^2 \\ 2t_i & 2t_i^2 & \cdots & 2t_i^2 \\ \vdots & \vdots & \cdots & \vdots \\ 2t_i & 2t_i^2 & \cdots & 2t_i^2 \end{pmatrix} S_{n,i}$$

$$\begin{matrix} 1 & 2 & \cdots & k \end{matrix}$$

$$\begin{pmatrix} 2t_i & \cdots & 2t_i \\ -2t_i^2 & \cdots & -2t_i^2 \\ \vdots & \vdots & \vdots \\ -2t_i^2 & \cdots & -2t_i^2 \\ 1+(c+2)t_i^2 & \cdots & 2t_i^2 \\ 2t_i^2 & \cdots & 2t_i^2 \\ \vdots & \vdots & \vdots \\ 2t_i^2 & \cdots & 1+(c+2)t_i^2 \end{pmatrix} S_{n,i},$$

$$\begin{matrix} k+1 & \cdots & n \end{matrix} \quad (16)$$

where $c = 2k - n - 1$. $H_{n,i}$ does satisfy equation (12) and hence can be used as an elementary rotation matrix. Equation (16) defines the elementary rotation matrix for a general n-D CORDIC algorithm, applicable to both Euclidean and pseudo-Euclidean spaces. When $\Sigma_n = I_n$, $H_{n,i}$ is an elementary Euclidean rotation matrix and the more specific notation $\bar{R}_{n,i}$ is used instead of $H_{n,i}$. If the signature matrix for an *odd* dimensional pseudo-Euclidean space is such that $k = (n+1)/2$, *i.e.*, $c = 0$, *no scaling* is required to normalize the rotated vectors. Elementary CORDIC rotation matrices for $n = 2, 3$ and 4 are exhibited below:

- $n = 2$, $\sigma_1 = 1$:

$$\bar{R}_{2,i} = \frac{1}{1+t_i^2} S_{2,i} \begin{pmatrix} 1-t_i^2 & 2t_i \\ -2t_i & 1-t_i^2 \end{pmatrix} S_{2,i}, \qquad (17)$$

- $n = 2$, $\sigma_1 = -1$:

$$H_{2,i} = \frac{1}{1-t_i^2} S_{2,i} \begin{pmatrix} 1+t_i^2 & 2t_i \\ 2t_i & 1+t_i^2 \end{pmatrix} S_{2,i}, \qquad (18)$$

- $n = 3, \sigma_1 = \sigma_2 = 1$: $H_{3,i} = \bar{R}_{3,i}$ in equation (8),
- $n = 3$, $\sigma_1 = 1, \sigma_2 = -1$:

$$H_{3,i} = S_{3,i} \begin{pmatrix} 1 & 2t_i & 2t_i \\ -2t_i & 1-2t_i^2 & -2t_i^2 \\ 2t_i & 2t_i^2 & 1+2t_i^2 \end{pmatrix} S_{3,i}, \qquad (19)$$

- $n = 4$, $\sigma_1 = \sigma_2 = \sigma_3 = 1$:

$$\bar{R}_{4,i} = (1+3t_i^2)^{-1} \times S_{4,i} \begin{pmatrix} 1-3t_i^2 & 2t_i & 2t_i & 2t_i \\ -2t_i & 1+t_i^2 & -2t_i^2 & -2t_i^2 \\ -2t_i & -2t_i^2 & 1+t_i^2 & -2t_i^2 \\ -2t_i & -2t_i^2 & -2t_i^2 & 1+t_i^2 \end{pmatrix} S_{4,i}, \qquad (20)$$

- $n = 4$, $\sigma_1 = 1, \sigma_2 = \sigma_3 = -1$:

$$H_{4,i} = (1-t_i^2)^{-1} \times S_{4,i} \begin{pmatrix} 1+t_i^2 & 2t_i & 2t_i & 2t_i \\ -2t_i & 1-3t_i^2 & -2t_i^2 & -2t_i^2 \\ 2t_i & 2t_i^2 & 1+t_i^2 & 2t_i^2 \\ 2t_i & 2t_i^2 & 2t_i^2 & 1+t_i^2 \end{pmatrix} S_{4,i}. \qquad (21)$$

The elementary rotations $\bar{R}_{2,i}$ and $H_{2,i}$ are the square of the elementary rotations $R_{2,i}$ in equation (1) with σ set to 1 and -1, respectively. We call the algorithm of equation (1) a 'square-root' 2-D CORDIC algorithm and the algorithms of equations (17) and (18) 'rational' CORDIC algorithms. The rational 2-D Euclidean CORDIC algorithm has been used in [5] [8] to speed up the Singular Value Decomposition (SVD) computations and in [12] to obtain a redundant arithmetic algorithm with constant scaling. For $n = 4$, while the algorithm with elementary rotations $R_{4,i}$ given by equation (6) is a square-root CORDIC algorithm, the new algorithm with elementary rotations $\bar{R}_{4,i}$ is a rational CORDIC algorithm whose unnormalized elementary rotations are only slightly more complex.

The convergence of the new 4-D CORDIC algorithm can be analyzed and proved using the procedure outlined in Section 1. The cones C_i have for vertex the origin and surround the first axis, thus they are fully characterized by their intersection with the hyperplane $x_1 = 1$. Since they are symmetric with respect to x_2, x_3, x_4, these 3-D polytopes can further be projected orthogonally onto the hyperplane $x_2 = 0$ to study the convergence behavior. Fig. 3 represents this projection for the cones C_1, C_2 and C_3.

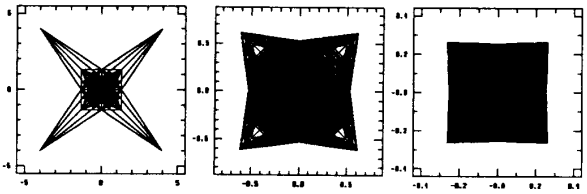

Figure 3: Domains after the first three iterations of $\bar{R}_4$.

Algorithm $\bar{R}_4$ does not require any repetition and converges faster than R_4, which requires repetition of the iterations with $t_i = 2^{-3}, 2^{-5}, 2^{-9}, 2^{-16}, \ldots$ [7]. The domain obtained by intersecting C_i with the hyperplane $x_1 = 1$ is bounded by a 3-D cube with half-side $2t_i/(1-3t_i^2)$.

2.3 Structure of the Elementary Rotations

The n-D elementary rotation matrix $H_{n,i}$ in equation (16) is the product of a reflection with respect to the hyperplane $x_1 = 0$ and a 'pseudo-Householder' reflection. Indeed, $H_{n,i}$ can be rewritten as

$$H_{n,i} = E_n \left(I_n - \frac{2\Sigma_n u u^T}{u^T \Sigma_n u} \right), \qquad (22)$$

where

$$E_n = \begin{pmatrix} -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \text{ and } u = \begin{pmatrix} 1 \\ t_i s_{1,i} \\ \cdot \\ \cdot \\ t_i s_{n-1,i} \end{pmatrix}.$$

The matrix $I_n - 2\Sigma_n u (u^T \Sigma_n u)^{-1} u^T$ is a reflection with respect to the hyperplane normal to the vector u. It is orthogonal with respect to Σ_n and has determinant -1. It is a generalization of the Householder reflection to pseudo-Euclidean spaces. The role of the multiplication by the reflection E_n is to transform the pseudo-Householder reflection into a proper rotation, with determinant 1. Thus, our new elementary rotation matrix $H_{n,i}$ is essentially a Householder reflection in pseudo-Euclidean space. This is the origin for the name given to the new CORDIC algorithm.

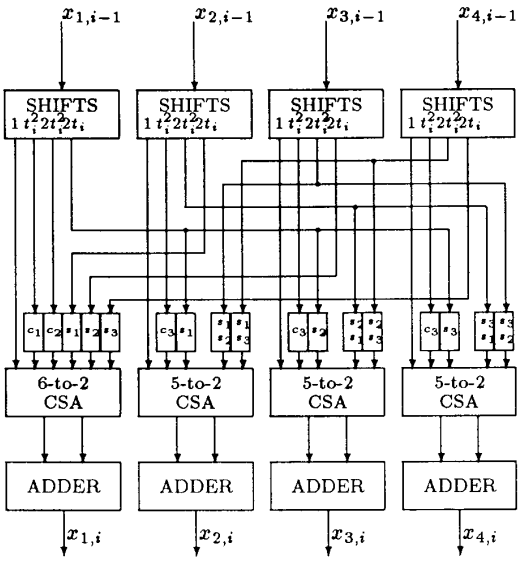

Figure 4: A possible architecture for a 4-D CORDIC processor.

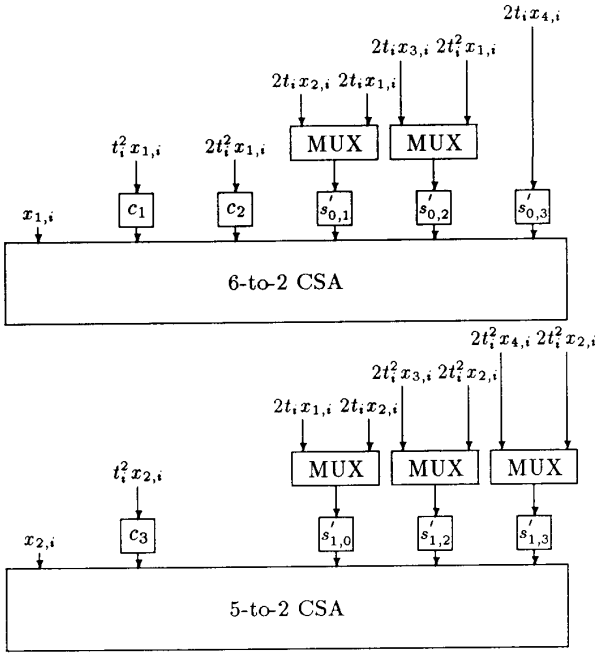

Figure 5: Modifications to the 4-D CORDIC processor architecture of Fig. 4 in order to perform scaling iterations in the processor

3 Processor Architectures

3.1 A 4-D Processor Architecture

A possible architecture to implement the 4-D CORDIC algorithm in equation (20) is illustrated in Fig. 4. By adding proper control inputs, c_1, c_2 and c_3, to the sign operators, this 4-D CORDIC processor can also execute the 2-D CORDIC algorithm in equation (17) and the 3-D CORDIC algorithm in equation (8). More precisely, in the (unscaled) rotation mode, the set of coefficients $(c_1, c_2, c_3) = (-1, 0, -1)$ selects the 2-D CORDIC function if $x_{3,i-1}$ and $x_{4,i-1}$ are preset to zero; $(c_1, c_2, c_3) = (0, -1, 0)$ selects the 3-D CORDIC function if $x_{4,i-1}$ is preset to zero; and $(c_1, c_2, c_3) = (-1, -1, 1)$ selects the 4-D CORDIC function.

Multiplication by a p-bit accurate approximation of the scaling factor $S_4 = \prod_{i=1}^{p}(1 + 3t_i^2)^{-1}$ can be performed by shifts and additions using q additional iterations, where q is small compared to p. However, the processor in Fig. 4 must be slightly modified to also perform the scaling operations. Several 2-to-1 multiplexers should be inserted at the input of the Carry-Save-Adders (CSA). Fig. 5 shows the modifications of the processor to enable scaling of the first and second components. The modifications for the third and fourth components are similar to those for the second component; the sign controls $(s'_{1,0}, s'_{1,2}, s'_{1,3})$ are replaced by $(s'_{2,0}, s'_{2,1}, s'_{2,3})$ and $(s'_{3,0}, s'_{3,1}, s'_{3,2})$, respectively.

The scaling factor S_4 is approximated by $\tilde{S}_4 = \prod_{i=1}^{q}(1 + a_it_i + b_it_i^2)$ where $a_i \in \{0, \pm 2\}$ and $b_i \in \{0, \pm 1, \pm 2, \pm 3, \pm 4, \pm 5\}$. In the rotation mode, the multiplexers select data from the left-hand side inputs. The sign controls $s'_{k,l,i}, 0 \le k \ne l \le 3$, at iteration i are selected such that $s'_{k,l,i} = s_{k,i}s_{l,i}$ where $s_{0,i} = 1$. The iteration index i is dropped for clarity in Fig. 5. In the scaling mode, the multiplexers select the data from their right-hand side input. The sign controls are such that

$$c_1 = c_3, \qquad c_2 = s'_{1,3} = s'_{2,3} = s'_{3,2},$$
$$s'_{0,1} = s'_{1,0} = s'_{2,0} = s'_{3,0}, \quad s'_{0,2} = s'_{1,2} = s'_{2,1} = s'_{3,1}.$$

We found that $q = 5$ extra iterations are required to perform the scaling operations in the 4-D CORDIC algorithm for 32 bits of accuracy.

A custom chip implementing the rational 2-D CORDIC rotation $\tilde{R}_{2,i}$ in equation (17) has been designed at Yale and fabricated through MOSIS with a 2μ CMOS process. It operates on 32-bit fixed-point words with 5 extra guard bits. The $2t_i$ and t_i^2 shifters are realized by trees of 2-to-1 multiplexers, providing higher performance than barrel shifters. The area of two shifters by t_i and t_i^2, A_{sh2}, is about 1.4 times the area of a single shifter, A_{sh}, due to the possibility of overlapping layouts. The adders in this chip are realized by a variant of the Abraham-Gajski tree structure [1]. The depth of the trees to realize both shifters and adders is $\lceil \log_2 b \rceil$ where b is the word length, including guard bits. The area of such an adder is about 1.6 times A_{sh}. However, it is possible to design a faster adder

with smaller area using multiple output domino logic and carry-look-ahead [10]. The area of this smaller adder, A_{add}, is about 0.6 times A_{sh}. The area of a 3-to-2 Carry-Save-Adder array, A_{csa}, is about 0.13 times A_{sh}. The area of a 2-to-1 word multiplexer, A_{mux}, is about 0.08 times A_{sh}. To sum up,

$$A_{sh2} = 1.4A_{sh} \quad A_{add} = 0.6A_{sh}$$
$$A_{csa} = 0.13A_{sh} \quad A_{mux} = 0.08A_{sh}.$$

Note that the sensitivity of these factors—1.4, 0.6, 0.13 and 0.08—to the bit accuracy is low since A_{sh}, A_{sh2} and A_{add} increase with bit accuracy slightly faster than linearly, while A_{csa} and A_{mux} are proportional to the bit accuracy.

The total area for the 4-D CORDIC processor, A_{4D}, is well approximated by the sum of the areas of the adders, shifters, CSA arrays, multiplexers and interconnection wires. The area of the wires, $A_{4D-wires}$, is about 2 times A_{sh} in this 4-D processor. Therefore

$$\begin{aligned} A_{4D} &\simeq 4A_{sh2} + 4A_{add} + A_{6-2CSA} \\ &\quad +3A_{5-2CSA} + 11A_{mux} + A_{4D-wire} \\ &\simeq 5.6A_{sh} + 2.4A_{sh} + 0.52A_{sh} \\ &\quad +1.17A_{sh} + 0.88A_{sh} + 2A_{sh} \\ &= 12.57A_{sh} \end{aligned}$$

Since the square-root 2-D CORDIC implementation in Fig. 1 requires

$$\begin{aligned} A_{2D} &\simeq A_{sh2} + 2A_{add} + 2A_{mux} + A_{2D-wire} \\ &\simeq 1.4A_{sh} + 1.2A_{sh} + 0.16A_{sh} + 0.58A_{sh} \\ &\simeq 3.34A_{sh}, \end{aligned}$$

the area of a 4-D CORDIC processor is about 3.8 times the area of a 2-D CORDIC processor.

The delay of the 6-to-2 and 5-to-2 CSA, T_{CSA}, is about three gate delays, $3T_{gate}$, where T_{gate} denotes the average delay per gate. The delay of a 2-to-1 multiplexer, T_{mux}, is T_{gate}. The delay of a shifter, $T_{shifter}$, is about $c_1\lceil \log_2 b \rceil$ gates delay for b-bit internal wordlength, where the factor c_1 accounts for the delay of the intermediate buffer stages driving the wires ($c_1 \simeq 1.3$). The delay of an adder is about $c_2\lceil \log_2 b \rceil$ gate delays, where c_2 accounts for the large carry-look-ahead gates ($c_2 \simeq 1.3$). The delay to drive the interconnection wires, T_{wires}, is about $2T_{gate}$. Therefore, for 32-bit accuracy, the delay of one 4-D CORDIC iteration is

$$\begin{aligned} T_{4D} &\simeq T_{shifter} + T_{mux} + T_{CSA} + T_{adder} + T_{wires} \\ &\simeq 6.5T_{gate} + T_{gate} + 3T_{gate} + 6.5T_{gate} + 2T_{gate} \\ &= 19T_{gate} \end{aligned}$$

while the delay for one 2-D CORDIC iteration is

$$\begin{aligned} T_{2D} &\simeq T_{shifter} + T_{mux} + T_{adder} + T_{wires} \\ &\simeq 6.5T_{gate} + T_{gate} + 6.5T_{gate} + 2T_{gate} \\ &= 16T_{gate}. \end{aligned}$$

To evaluate or apply a single 4-D rotation the proposed processor takes 60% of the time of two 2-D CORDIC processors at the expense of an area increase by a factor of 1.9. The gain in speed is thus obtained with a slight increase in the area-time product.

3.2 n-D Processor Architectures

Turning now to general n-D CORDIC algorithms, we found computationally that the first iteration (with $t_1 = 2^{-2}$) should be repeated twice for $n = 5, 6, 7$ and 8 to ensure convergence. The architecture in Fig. 4 can be easily generalized to implement an n-D CORDIC processor. For instance, a 5-D CORDIC processor requires an area $A_{5D} \simeq 1.3A_{4D}$ and a computation time $T_{5D} \simeq 1.1T_{4D}$. Therefore, a 5-D CORDIC processor takes about 40 % of the time ($= 3T_{2D}$) of two 2-D CORDIC processors to perform a 5-D rotation, with essentially no increase in the area-time product.

An alternative architecture for an n-D CORDIC processor, based on the the representation in terms of pseudo-Householder reflection given in equation (22), is shown in Fig. 6. Because of an additional shifter delay, this 'CORDIC Householder architecture' requires a longer computation time for each CORDIC iteration than the plain 'CORDIC architecture' exemplified in Fig. 4. On the other hand, it requires less area for high dimensional rotations ($n > 8$).

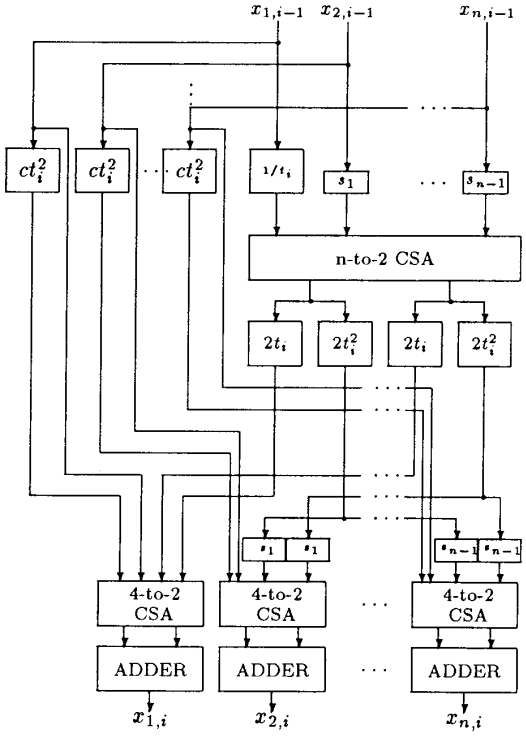

Figure 6: Architecture of an n-D pseudo-Euclidean CORDIC processor ($c = 2k - n - 1$).

3.3 Area and Time Comparisons

Two architectures have been presented: 'CORDIC architecture', which is the generalization of Fig. 4, and 'CORDIC Householder architecture' of Fig. 6. Since

the pseudo-Householder transformation is considered to be the most efficient way to implement n-D vector rotations, we shall compare the two CORDIC architectures to a highly parallel implementation of the pseudo-Householder transformation. To any n-D vector $\boldsymbol{x} = [x_1\ x_2\ \cdots\ x_n]^T$ in a space with signature matrix Σ_n may be associated a pseudo-Householder reflection bringing $\boldsymbol{x}$ along the axis $e_1 = [1\ 0\ \cdots 0]^T$:

$$H = I_n - 2\Sigma_n u(u^T\Sigma_n u)^{-1}u^T,$$

where $u = x + sign(x_1)\|x\|_{\Sigma_n}e_1$. The pseudo-Householder transformation calls for division, square-root and multiplications. Division and square-root extraction can be implemented by the Newton-Raphson method using multiplier and ROMs [14]. The multiplications are performed using n multipliers in parallel. For comparison purposes, the area and delay of a multiplier will be expressed in terms of the units A_{sh} and T_{gate}, respectively. The 32 × 32 multiplier presented in [11], which uses the Booth algorithm and a Wallace tree, will be used for our comparison. Compared to the 32 × 32 adder in [10], its area and delay are

$$A_{mpy} \simeq 35A_{add} \simeq 21A_{sh},$$
$$T_{mpy} \simeq 5A_{add} \simeq 5c_2\lceil\log_2 b\rceil T_{gate}.$$

In Fig. 7, $A_C, A_{CH}, A_H, T_C, T_{CH}$ and T_H denote the respective areas and delays of 'CORDIC', 'CORDIC Householder' and 'Householder' implementations for the Euclidean rotation of an n-D vector. Fig. 7(a) compares the areas of these implementations, Fig. 7(b) their delays, and Fig. 7(c) their area-time products. The two implementations of the new CORDIC algorithm are clearly preferable in terms of area-time product to the parallel implementation of the pseudo-Householder transformation.

4 Application to the QR Decomposition on a Processor Array

Givens rotations are widely used in modern signal processing algorithms and the square-root 2-D CORDIC algorithm is often employed to implement these rotations. Two well-known examples are the QR Decomposition (QRD) and Singular Value Decomposition (SVD)[2][3]. We shall examine here the QRD of a rectangular matrix to show how higher dimensional CORDIC algorithms can speed up array implementations of matrix computations.

The n-D CORDIC algorithm discussed in Section 2 enables $n-1$ elements to be zeroed out at each rotation instead of a single element for a Givens rotation. To triangularize an $l \times m$ matrix A with elements $a_{i,j}$, an n-D rotation is applied at time step 1 to zero out $a_{2,1}, \cdots, a_{n,1}$ in the first column of A. Simultaneously, this rotation is applied to the other elements of rows 1 to n. At time step 2, another n-D rotation is applied to rows 1 and $n+1$ to $2n-1$ to zero out $a_{n+1,1}, \cdots, a_{2n-1,1}$, while an $(n-1)$-D rotation is applied to rows 2 to n to zero out $a_{3,2}, a_{4,2}, \cdots, a_{n,2}$. Proceeding with this greedy approach, an $l \times m$ matrix is triangularized in

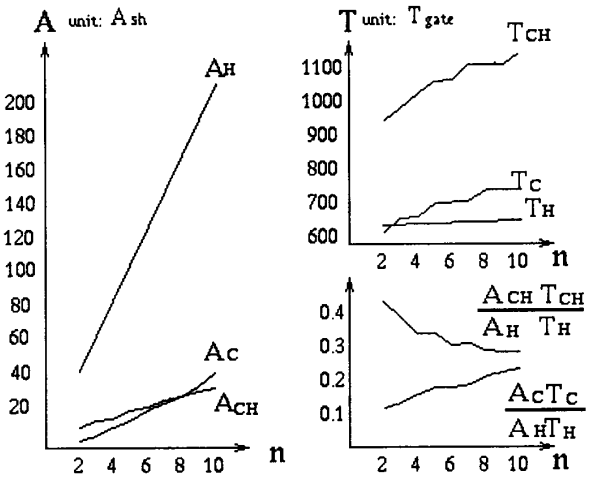

Figure 7: Area, time and area-time product as function of dimension n (b=32).

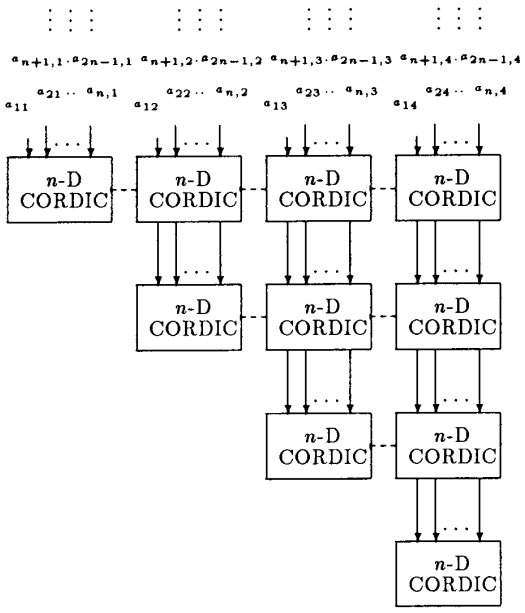

Figure 8: Array of n-D CORDIC processors for decomposing an $l \times 4$ matrix.

QRD	2-D CORDIC	4-D CORDIC
time	$16(l+m-2)\times$ $(\bar{p}+q_1)$	$19(\lceil\frac{l-1}{3}\rceil+m-1)\times$ $(\bar{p}+1+q_2)$
period	$16(l-1)(\bar{p}+q_1)$	$19(\lceil\frac{l-1}{3}\rceil)(\bar{p}+1+q_2)$
area	$1.7m(m+1)$	$6.4m(m+1)$

Table 1: Comparison, for the QRD of $l \times m$ matrices, of the computation times for one problem, and of the periods and processor areas. The time unit is T_{gate} and the area unit is A_{sh}.

$\lceil(l-1)/(n-1)\rceil + (m-1)$ time steps on a triangular array of $m(m+1)/2$ n-D CORDIC processors as shown in Fig. 8. Each time step allows for a whole n-D CORDIC rotation. If several $l \times m$ matrices are pipelined into the array, the pipeline period to compute one QRD is $\lceil(l-1)/(n-1)\rceil$ time steps. Note that the ability of an n-D CORDIC processor to also perform lower dimensional rotations is crucial in this context.

Table 1 compares the computation time, the pipeline period and the processor area required for the QRD of $l \times m$ matrices using the square-root 2-D CORDIC algorithm and the new 4-D CORDIC algorithm; $\bar{p}$ is the bit accuracy, and q_1 and q_2 are the number of additional iterations required for scaling operations in the 2-D and 4-D algorithms, respectively. The 4-D CORDIC algorithm provides a speed up of 2.5 at the expense of an area increase by a factor of 3.8. The time performance is thus significantly improved with only a 50% increase in the area-time product.

5 Conclusion

A general n-D CORDIC algorithm, applicable to rotations in Euclidean and pseudo-Euclidean spaces, has been introduced. In the 3-D and 4-D Euclidean cases, the algorithm does not require any repetition and thus improves on previously developed CORDIC algorithms. The elementary rotations in the new algorithm are essentially pseudo-Householder reflections. Parallel implementations of the algorithm require, for $n \leq 10$, much less area than a parallel implementation of the Householder transformation to achieve roughly the same computation time and, in addition, they are simpler to design. Finally, the example of the QR decomposition illustrates the use of the new n-D algorithms in application-specific processor arrays in order to speed up matrix computations.

6 Acknowledgement

This work was supported by the Defense Advanced Project Research Agency (DARPA) under contract N00014-88-K-0573.

References

[1] J. Abraham and D. Gajski, *Easily Testable, High-speed Realization of Register-Transfer-Level Operations*, Proc. 10th Fault-Tolrant Computing Symp., pp. 339-344, Oct. 1980.

[2] H.M. Ahmed, J.-M. Delosme and M. Morf, *Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing*, IEEE Computer, Vol. 15, No. 1, pp. 65-82, Jan. 1982.

[3] J.R. Cavallaro and F.T. Luk, *CORDIC Arithmetic for an SVD processor*, Journal of Parallel and Distributed Computing, pp. 271-290, June 1988.

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

[5] J.-M. Delosme, *A Processor for Two-dimensional Symmetric Eigenvalue and Singular Value Arrays*, Proc. 21st Asilomar Conf. on Circ., Syst., and Comp., pp. 217-221, Nov. 1987.

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

[7] J.-M. Delosme and S.-F. Hsiao, *CORDIC Algorithms in Four Dimensions*, Advanced Signal Processing Algorithms, Architectures and Implementations, Proc. SPIE 1348, pp. 349-360, July 1990.

[8] J.-M. Delosme, *Bit-level Systolic Algorithm for the Symmetric Eigenvalue Problem*, Proc. Int. Conf. on Application Specific Array Processors, pp. 770-781, Princeton, Sept. 1990.

[9] G. Haviland and A. Tuszynski, *A CORDIC Arithmetic Processor Chip*, IEEE Trans. on Computers, Vol. C-29, No. 2, pp. 68-79, Feb. 1980.

[10] I.S. Hwang and A.L. Fisher, *Ultrafast Compact 32-bit CMOS Adders in Multiple-Output Domino Logic*, IEEE J. Solid State Circuits, Vol. SC-24, pp. 358-369, Apr. 1989.

[11] M. Nagamatsu et al., *A 15-ns 32 x 32-b CMOS Multiplier with an Improved Parallel Structure*, IEEE J. Solid State Circuits, Vol. SC-25, pp. 494-496, Apr. 1990.

[12] N. Takagi, T. Asada and S. Yajima, *A Hardware Algorithm for Computing Sine and Cosine Using Redundant Binary Representation*, Trans. IECE Japan, Vol. J69-D, No. 6, pp. 841-847, June 86 (in Japanese). English translation available in *Systems and Computers in Japan*, Vol. 18, No. 8, pp. 1-9, Aug. 1987.

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

[14] S. Waser and M.J. Flynn, *Introduction to Arithmetic for Digital Systems Designs*, CBS College Publishing, 1982.

[15] J.S. Walther, *A Unified Algorithm for Elementary Functions*, AFIPS Conf. Proc., Vol. 38, pp. 379-385, 1971.