Proceedings

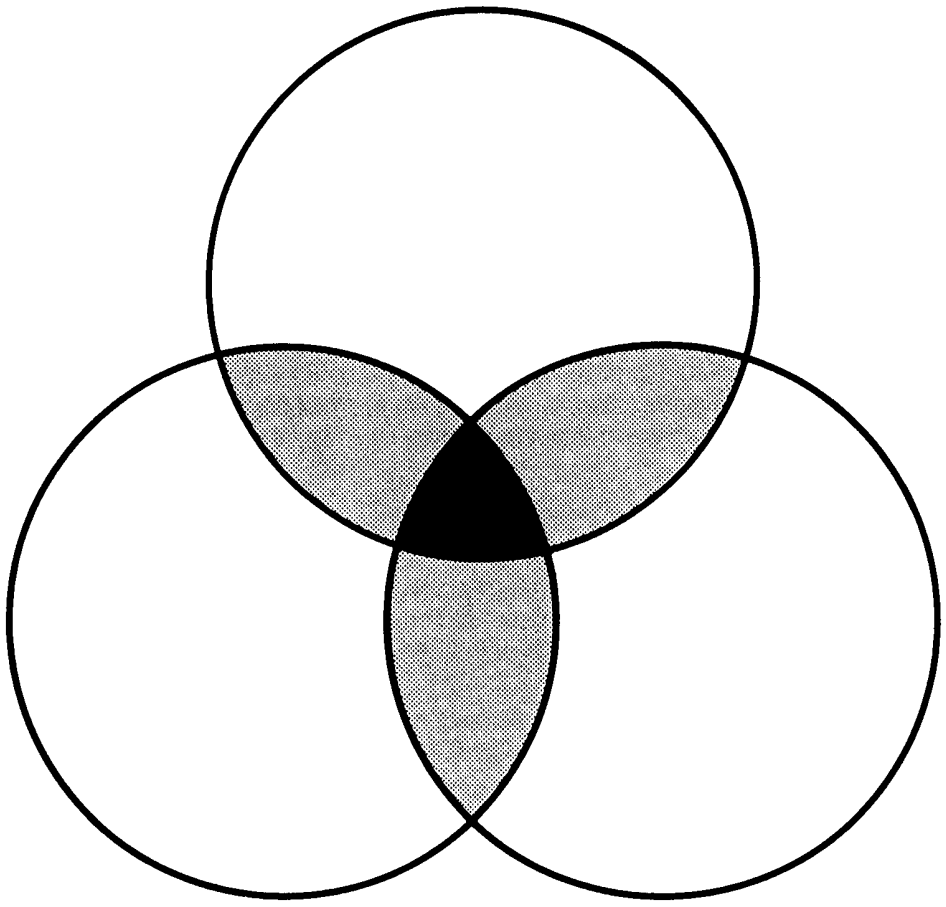
1988 IEEE INTERNATIONAL CONFERENCE ON Computer Design: VLSI in Computers & Processors

Sponsored by The Computer Society of the IEEE and IEEE Circuits and Systems Society
In cooperation with IEEE Electron Devices Society

Computer Society Order Number 872
Library of Congress Number 88-82095
IEEE Catalog Number 88CH2643-5
ISBN 0-8186-0872-2
SAN 264-620X

ICCD '88

Rye Town Hilton, Rye Brook, New York
October 3-5, 1988



THE COMPUTER SOCIETY
IEEE THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
IEEE COMPUTER SOCIETY PRESS

1988 IEEE International Conference on Computer Design: VLSI in Computers & Processors
Rye Brook, New York, October 1988

Floating-Point CORDIC for Matrix Computations

Joseph R. Cavallaro

Department of Electrical and Computer Engineering
Rice University
Houston, Texas 77251

Franklin T. Luk

School of Electrical Engineering
Cornell University
Ithaca, New York 14853

Abstract

The CORDIC algorithms provide a VLSI hardware technique for computing the inverse tangents and vector rotations needed by many matrix decomposition algorithms. Although normally stated for fixed-point arithmetic, the CORDIC algorithms may be performed in floating-point arithmetic. A novel simplified CORDIC processor composed of floating-point data paths with a fixed-point angle calculation is proposed. This hybrid processor allows sufficient accuracy for matrix computations.

1. Introduction

The Coordinate Rotation Digital Computer (CORDIC) [8] algorithms provide an efficient VLSI scheme for vector rotation. The simplicity of the algorithm is due to the use of basic structures, (adders, shifters, registers), simple control, and a small ROM for the storage of rotation angle information. The algorithms were originally described in a fixed-point format and have been applied to special purpose processors for real-time signal processing applications [8, 5, 1]. CORDIC has also been applied to matrix factorizations such as the QR decomposition [2] and the Singular Value Decomposition (SVD) [3]. However, for practical systems, a floating-point implementation is preferred. Although there has been some work in this area [9, 1, 6, 4], several issues still remain open. In particular, the accuracy of floating-point CORDIC is of concern, along with the number of bits required in the mantissa, the number of iterations for a given format, and the scale factor correction scheme. In this paper, these issues will be explored along with novel enhancements to the CORDIC algorithm.

2. Review of Previous Work

The CORDIC algorithms are based upon plane rotations. The vector (x_0, y_0) is rotated through an angle θ to (x_0', y_0') and the rotation equations are:

$$\begin{bmatrix} x_0' \\ y_0' \end{bmatrix} = \boldsymbol{R}(\theta) \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} . \qquad (1)$$

In the CORDIC algorithms, the total rotation angle θ is decomposed into the sum of a sequence of n known angles which are stored in a ROM. The angle θ is then expressed as:

$$\theta = \pm\theta_0 \pm \theta_1 \cdots \pm \theta_{n-1} = \sum_{i=0}^{n-1} \delta_i \theta_i , \qquad (2)$$

where $\theta_i > 0$ and $\delta_i = \pm 1$. After the rotation equations are divided by $\cos\theta_i$ and the resulting $\tan\theta_i$ term is set equal to 2^{-i}, for a radix 2 machine, the CORDIC iteration equations are:

$$\begin{aligned} x_{i+1} &= x_i + \delta_i y_i 2^{-i} , \\ y_{i+1} &= y_i - \delta_i x_i 2^{-i} , \\ z_{i+1} &= z_i + \delta_i \theta_i . \end{aligned} \qquad (3)$$

The additional z equation holds the accumulated rotation angle. A scale factor which is equal to the product of the $\cos\theta_i$ terms also occurs. In fixed-point CORDIC, the x and y equations are implemented in hardware by a register, barrel shifter, and adder. The z equation requires a register, adder, and an entry from the ROM table of angles.

Walther [9] devotes a section to the conversion from fixed-point to floating-point formats. He mentions that the basic iteration equations are easily expressed using floating-point arithmetic. In addition, he makes several brief comments about modifying the iteration sequence for floating-point. Ahmed [1] points out that the shifting required in the fixed-point algorithm gets converted into exponent subtraction and mantissa alignment in the floating-point case.

Johnsson [6] has more recently begun to address the problems of accuracy in a floating-point format. Drawing upon the work of Muller [7], whose *"explosion"* of double precision IEEE floating-point into 1075-bit fixed-point is mentioned, Johnsson seeks to reduce the number of bits and iterations through various approximations. However, firm bounds on the error and a unified floating-point algorithm remain to be addressed.

Both schemes presented by Walther and Johnsson perform at least n iterations. However, the first iteration is not rooted to $i = 0$ or 45°. The index, i, may float to correspond to the most significant bit in the argument. By continuing for n iterations, the mantissa is accurate to n bits. However, a large table of angles is necessary. For the IEEE single precision format [6], a table with 151 words is needed. Unfortunately, a different scale factor is required for each possible family of iterations. Walther suggests to form the scale factor as a product based upon the starting iteration. This scheme is only acceptable for low-speed desk calculators which were one of his original applications.

Johnsson suggests the creation of a second ROM to store the indices for scale factor correction using the Haviland and Tuszynski scheme [5]. Each ROM entry corresponds to the bit vector for a different starting iteration. He tries to achieve some savings by partitioning this table.

3. Improvements to Floating-Point CORDIC

In this paper, several important observations are made concerning practical error considerations for floating-point CORDIC and the special needs of rotation-based matrix algorithms. These observations lead to simplification of the floating-point CORDIC SVD architecture through the design of a hybrid processor which contains both fixed-point and floating-point data paths. The resulting design is suitable for VLSI implementation.

3.1. Practical Error Considerations for Floating-Point CORDIC

In the previous schemes, a full n bits are produced, even if the first iteration is for $i > 0$. Therefore, rotations by extremely small angles are performed. Johnsson tries to control this by making the approximations, $\sin z \approx z$ for $|z| < 2^{-n/2+1}$, and $\cos z \approx 1$ for $|z| < 2^{-n/2}$. Similarly, $\arctan(y/x) \approx y/x$ for $|y/x| < 2^{-n/2}$.

In many numerical applications, rotations by such small angles do not further contribute to the overall accuracy of the algorithm. A scheme where the smallest rotation angle is 2^{-n} may provide sufficient accuracy even if all n bits of the normalized floating-point mantissa have not been explicitly determined. By limiting the table of angles to n words, the floating-point CORDIC algorithm reduces to an algorithm with complexity similar to the fixed-point case.

3.2. Special Needs of Matrix Algorithms

A general floating-point CORDIC processor must produce accurate results to full machine precision for arbitrary arguments. This requires the use of a larger angle memory and non-standard scale factor correction constants. However, for matrix computations, such as the QR decomposition, and for iterative algorithms, such as the eigenvalue decomposition and the SVD, such accuracy in the rotation angle calculation is not required. The parallel Jacobi algorithm for the SVD of a matrix A is required to produce an error less than $\varepsilon \|A\|_F$ where ε is the machine precision. Therefore, it is not necessary to produce angles which are more accurate than ε. In our new floating-point CORDIC processor, the z equation and the associated angle memory is reduced to a fixed-point format containing the same number of bits as the mantissas of the x and y floating-point data paths.

3.3. Floating-Point CORDIC with a Fixed-Point Angle Path

In Figure 1, the novel design of a floating-point CORDIC processor for use in matrix computations is presented. The x and y data paths use a standard floating-point format. The shifter which is required in the fixed-point CORDIC processor is logically replaced by an exponent subtraction unit. The actual exponent subtraction can occur within the floating-point adder. The important reduction in algorithm complexity occurs in the z data path where the angle information is accumulated. The angle memory, z register, and adder remain in a fixed-point format. In this hybrid processor structure, the z data path only requires the same number of bits as the mantissas of the x and y data paths. For example, if IEEE single precision is used, then the z data path would be composed of 24 bits and the angle memory would contain 24 words. In all designs, an additional $\log n$ bits may be needed to control round-off error within the processor. Since a fixed number of iterations starting with the first angle, $i = 0, 1, \cdots, n-1$, is performed, the standard scale factor correction algorithm can be used. Therefore, the variable scale factor correction schemes suggested by Walther and Johnsson will not be necessary.

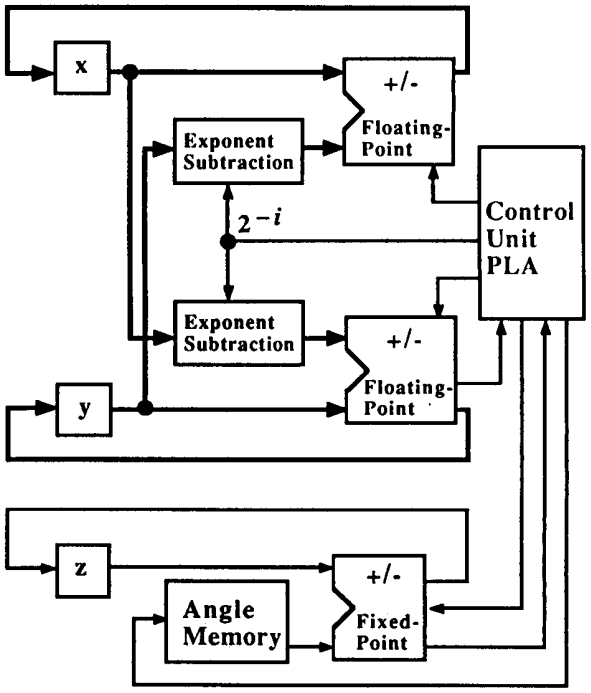
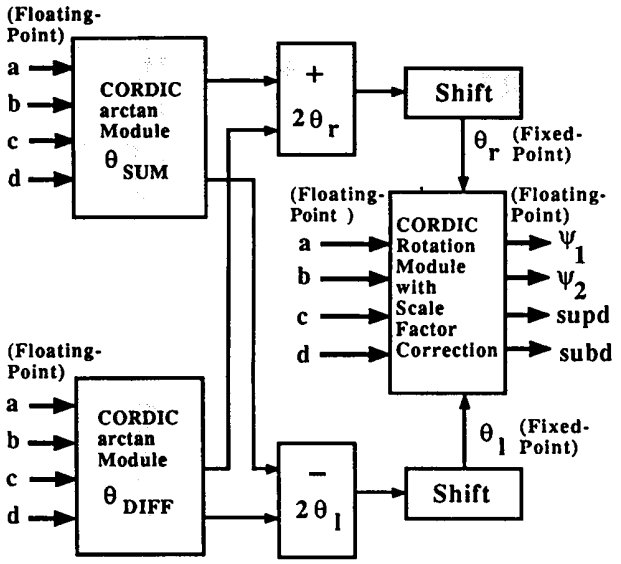


Figure 1. Floating-Point CORDIC Processor.

3.4. Floating-Point CORDIC SVD Processor Architecture

In Figure 2, the floating-point CORDIC processor is applied to the SVD parallel diagonalization method [3]. The data, a,b,c,d, are presented in floating-point to the CORDIC inverse tangent modules. The rotation angles, θ_{SUM} and θ_{DIFF} are output in a fixed-point format. These angles are modified to yield the fixed-point angles, θ_l and



Data in Floating-Point; Rotation angles in Fixed-Point.

Figure 2. Floating-Point CORDIC SVD Method.

θ_r. The CORDIC rotation module then applies these fixed-point angles to the floating-point data, a,b,c,d, to produce the floating-point outputs, ψ_1, ψ_2, *supd*, and *subd*. In this processor architecture, the data elements of the matrix A are represented in a floating-point format. The rotation angles, θ_l and θ_r, are derived from the data in fixed-point form and then applied through the CORDIC rotation modules to operate on floating-point data. The fixed-point angle format provides sufficient accuracy for the algorithm and eliminates the overhead of unnecessary iterations and the complexity of variable scale factor correction. The resulting simplification of the control structure will benefit the VLSI implementation.

4. Summary

In this paper, the floating-point CORDIC processor has been considered. Various schemes for a general-purpose floating-point processor have been discussed. A simplified hybrid processor is presented with floating-point data paths and a fixed-point angle calculation. This processor possesses sufficient accuracy for matrix computations such as the QRD, eigenvalue decomposition, and the SVD. The simplified structure allows for efficient VLSI implementation.

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 while at Cornell University.

References

1. Ahmed, H. M., "Signal Processing Algorithms and Architectures," Ph. D. Dissertation, Dept. of Electrical Engineering, Stanford Univ., Stanford, CA (June 1982).
2. Ahmed, H. M., Delosme, J. -M., and Morf, M., "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *IEEE Computer*, 15,(1), pp. 65-82 (January 1982).
3. Cavallaro, J. R. and Luk, F. T., "CORDIC Arithmetic for an SVD Processor," *Journal of Parallel and Distributed Computing*, pp. 271-290 (June 1988).
4. 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 (September 1987).
5. Haviland, G. L. and Tuszynski, A. A., "A CORDIC Arithmetic Processor Chip," *IEEE Trans. Computers*, **C-29**,(2), pp. 68-79 (Feb. 1980).
6. Johnsson, S. L. and Krishnaswamy, V., "Floating-Point CORDIC," Research Report YALEU/DCS/RR-473, Dept. of Computer Science, Yale Univ., New Haven, CT (April 1986).
7. Muller, J. -M., "Methodologies de Calcul des Functions Elementaires," Technical Report, Institute National Polytechnique de Grenoble (December 1985).
8. Volder, J., "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electronic Computers*, **EC-8**,(3), pp. 330-334 (Sept. 1959).
9. Walther, J. S., "A Unified Algorithm for Elementary Functions," *AFIPS Spring Joint Computer Conf.*, pp. 379-385 (1971).