

 Open access • Proceedings Article • DOI:10.1109/ISCAS.1992.230297

A systolic VLSI architecture for complex SVD — [Source link](#)

Nariankadu D. Hemkumar, Joseph R. Cavallaro

Institutions: Rice University

Published on: 10 May 1992 - International Symposium on Circuits and Systems

Topics: Singular value decomposition, Matrix (mathematics), CORDIC and Jacobi method

Related papers:

- [Computation of the Singular Value Decomposition Using Mesh-Connected Processors](#)
- [Matrix computations](#)
- [Improved SVD systolic array and implementation on FPGA](#)
- [The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays](#)
- [CORDIC arithmetic for an SVD processor](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/a-systolic-vlsi-architecture-for-complex-svd-2uxndzy2fj>

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 1345316

A systolic VLSI architecture for complex SVD

Hemkumar, Nariankadu Datatreya, M.S.

Rice University, 1991

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

RICE UNIVERSITY

A Systolic VLSI Architecture for Complex SVD

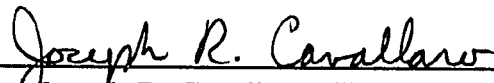
by

Nariankadu D. Hemkumar

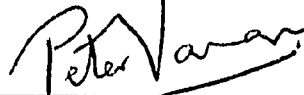
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:



Dr. Joseph R. Cavallaro, Chairman
Assistant Professor
Electrical and Computer Engineering



Dr. Peter J. Varman
Associate Professor
Electrical and Computer Engineering



Dr. Dan C. Sorensen
Professor
Mathematical Sciences

Houston, Texas

April, 1991

A Systolic VLSI Architecture for Complex SVD

Nariankadu D. Hemkumar

Abstract

This thesis presents a systolic algorithm for the SVD of arbitrary complex matrices, based on the cyclic Jacobi method with “parallel ordering”. As a basic step in the algorithm, a two-step, two-sided unitary transformation scheme is employed to diagonalize a complex 2×2 matrix. The transformations are tailored to the use of CORDIC (COordinate Rotation Digital Computer) algorithms for high speed arithmetic. The complex SVD array is modeled on the Brent-Luk-VanLoan array for real SVD. An array with $O(n^2)$ processors is required to compute the SVD of a $n \times n$ matrix in $O(n \log n)$ time. An architecture for the complex 2×2 processor with an area complexity twice that of a real 2×2 processor, is shown to have the best area/time tradeoff for VLSI implementation. Despite the involved nature of computations on complex data, the computation time for the complex SVD array is less than three times that for a real SVD array with a similar CORDIC based implementation.

Acknowledgments

I wish to express my heartfelt appreciation and gratitude to the many people without whose constant encouragement and support, this thesis would not have been possible.

First and foremost, I deeply acknowledge the tutelage and guidance of Dr. Joseph R. Cavallaro and thank him for giving me the opportunity to do research. He has truly been a “friend, philosopher and guide”, in every sense of the phrase.

I am sincerely grateful for the useful comments and suggestions afforded by Dr. Peter J. Varman and Dr. Dan C. Sorensen and their consenting to serve on the thesis committee.

Special thanks is due to my friend and fellow graduate student Kishore ‘kotax’ Kota, and my other office-mates Vinay Pai, Jim Carson and Jay Greenwood for making graduate academic life that much more a joy. I also thank ‘Boots’ and Raghu for their friendship and fraternity.

Finally, I wish to express my eternal indebtedness to my parents and my brother for the unflagging encouragement and support that I have received through the years and to whom I owe a lot.

To Amma and Appa

Contents

Abstract	ii
Acknowledgments	iii
List of Tables	viii
List of Illustrations	ix
1 Introduction	1
1.1 Systolic Array Processors	1
1.2 Parallel Architectures for the SVD	2
1.3 Contributions of the Thesis	4
1.4 Overview of the Thesis	5
2 Parallel Architectures for Real SVD	7
2.1 Singular Value Decomposition	7
2.2 SVD Algorithms	8
2.3 The SVD-Jacobi Method	9
2.4 SVD of a Real 2×2 Matrix	12
2.5 Systolic Architectures for Real SVD	13
3 CORDIC Architectures for Real SVD	16
3.1 CORDIC Algorithms	16
3.2 Architecture for a Basic CORDIC Processor	30

3.3	Cavallaro-Luk Architectures for the Real 2×2 Processor	31
4	The SVD of a Complex Matrix	37
4.1	Complex Number Representation	37
4.2	Complex Givens Rotation	38
4.3	QRD of a Complex 2×2 Matrix	39
4.4	SVD of a Complex 2×2 Matrix	41
4.5	Two-sided Unitary Transformations	44
4.6	Two-step SVD of a Complex 2×2 Matrix	47
5	CORDIC for Complex SVD	50
5.1	Modified CORDIC for Complex Arithmetic	50
5.2	CORDIC for the \mathcal{Q} Transformation	53
5.3	An Architecture for the Complex 2×2 Processor	61
5.4	Area/Time Analysis for the Complex 2×2 Processor	65
6	A Systolic Array for Complex SVD	71
6.1	The Brent-Luk-VanLoan Systolic Array	71
6.2	A Systolic Array for Complex SVD	80
6.3	\mathcal{Q} Transformations for the EVD of Hermitian Matrices	87
6.4	Performance of the Systolic SVD Arrays	88
6.5	Wavefront vs. Systolic Computation	90
7	Simulation of the Complex SVD Array	92
7.1	The Connection Machine	92
7.2	Simulation Model	93
7.3	Simulation Experiments and Results	96

7.4	Terminating Computations on the Complex SVD Array	97
8	Conclusions	100
8.1	Summary	100
8.2	Future Work	102
	Bibliography	103

Tables

3.1	Radius and Angle in Coordinate Systems	18
3.2	Transformations for CORDIC Operation Modes	23
3.3	Area/Time Complexity for Fixed-point SVD Architectures	34
5.1	Complex Multiplication with CORDIC	52
5.2	CORDIC Application of the Unitary Transformation	58
5.3	Pre-Processing Rules for Argument Addition	61
5.4	CORDIC SVD Processor Algorithm - Step I	67
5.5	CORDIC SVD Processor Algorithm - Step II	68
6.1	Algorithm Interchange	76
6.2	BLV Array : Processor Algorithm	79
6.3	Complex SVD : Processor Algorithm	86
6.4	Relative Performance of Matrix Decomposition Arrays	88
6.5	Step-wise Execution Times of the Matrix Decomposition Arrays	89
7.1	Time-Stamped Snapshots of a 2×2 Processor Array	95
7.2	Complex CORDIC SVD Convergence Behavior	97

Illustrations

3.1	Coordinate Systems	19
3.2	Results of CORDIC Iterations	22
3.3	Parallel Fixed-point CORDIC Module	32
3.4	The Parallel Fixed-point SVD Processor Architecture	33
3.5	Parallel Floating-point CORDIC Module	35
5.1	CORDIC Complex Givens Rotation Angle Calculation	53
5.2	CORDIC Complex Givens Rotation Application	54
5.3	Deprettere and van der Veen Complex Rotation in CORDIC	55
5.4	Rotation in the Complex Plane	60
5.5	Angle and Modulus in Modified Polar Coordinates	62
5.6	Diagonalization Step - I in CORDIC	63
5.7	Diagonalization Step - II in CORDIC	64
5.8	Complex 2×2 Processor Architecture	66
6.1	The Brent-Luk-VanLoan Systolic Array	73
6.2	Interprocessor Communication Links for Processors	74
6.3	Staggering of Computations in the BLV Array	78
6.4	Processors for the BLV Array	80

6.5	Data Exchange Timing for the BLV Array	81
6.6	Staggering of Computations on the Complex SVD Array	82
6.7	Data Exchange Timing for the Complex SVD Array	84
6.8	Processors for the Complex SVD Array	85
7.1	Mapping of the Systolic Array onto the CM2	94
7.2	Comparative Convergence of Real and Complex SVD Schemes	98

Chapter 1

Introduction

Real-time signal processing concerns combined with the advent of parallel algorithms and architectures, have pushed systolic arrays to the forefront of special-purpose computing. The Singular Value Decomposition (SVD) is a matrix factorization technique of considerable importance in engineering applications. Most systolic arrays proposed for the SVD in the literature, assume real input matrices. In this thesis, a systolic algorithm for the SVD of an arbitrary complex matrix is presented along with a CORDIC (COordinate Rotation Digital Computer) based VLSI architecture for the systolic processor element.

1.1 Systolic Array Processors

Contemporary parallel architectures may be grouped into three different classes [8] based on structure : vector processors, multiprocessor systems, and array processors. Vector processors and multi-processor systems belong to the domain of general-purpose computers while most array processors are designed for special-purpose applications.

Array processors, as a computing paradigm, are capable of meeting the real-time processing requirements of a variety of application domains. Locally interconnected computing networks such as systolic and wavefront processor arrays, due to their massive parallelism and regular dataflow, allow efficient implementation of a large

number of algorithms of practical significance; especially in the areas of image processing, signal processing, and robotics [51, 55].

Many definitions of systolic arrays exist in the literature [34, 54, 78]. Kung and Leiserson [52] define a systolic system as a “network of processors which rhythmically compute and pass data through the system”. Systolic arrays, as a class of pipelined array architectures, display regular and modular structures locally interconnected to allow a high degree of pipelining and synchronized multiprocessing capability [53]. The primary reasons for the use of systolic arrays in special-purpose processing are simple and regular design, concurrency and communication, with balanced computation and I/O [51].

1.2 Parallel Architectures for the SVD

While general-purpose computers require the design of complicated control units and optimized schemes for the allocation of machine resources, the design of array processors requires a good understanding of the relationship between parallel computing algorithms and optimal computing hardware/software structures. As most implementations of array processors tend to be VLSI/WSI, this optimization is of greater concern. Special-purpose arithmetic techniques can greatly improve hardware and performance efficiency.

The Singular Value Decomposition (SVD) is an important matrix factorization procedure used extensively in signal and image processing algorithms [3] and is very well suited to analyzing data matrices from sensor arrays [70]. The singular values can also be used to determine the rank of a matrix in a numerically reliable manner [36]. Furthermore, they can be used to find a good low-rank approximation to the original

matrix; a feature that has proven especially useful in image and signal processing problems [69].

Complex data matrices are known to occur frequently in engineering practice. In particular, several adaptive beam-forming algorithms [46, 64] which determine the direction or bearing of a signal source, require complex matrix factorizations and can benefit from a complex SVD array.

SVD algorithms require costly arithmetic operations such as division and square root in the computation of rotation parameters. Increased efficiency may be obtained through the use of hardware oriented arithmetic techniques that relate better to the algorithm [1, 71, 73]. Special VLSI structures have been proposed for the SVD [29, 62]. The COordinate Rotation Digital Computer (CORDIC) [80, 81] algorithms, which allow easy computation of inverse tangents and vector rotations, have proven extremely useful in this context [14, 29, 73].

Brent, Luk and VanLoan [6] proposed an expandable array of simple processors for 2×2 matrices, to compute the SVD of a larger matrix. The array uses the SVD-Jacobi method [32, 49] combined with a "parallel ordering" [5] scheme to exploit the parallelism inherent in the Jacobi method. Cavallaro and Luk [14] have demonstrated the use of CORDIC in a hardware and performance efficient architecture for the 2×2 processor.

Most parallel algorithms and architectures proposed for the SVD assume real input matrices. Deprettere and van der Veen [79] allowed for complex matrix elements but assumed a specialized matrix structure. Cavallaro and Elster [12] extended the Deprettere and van der Veen scheme to the SVD of an arbitrary complex 2×2 matrix.

1.3 Contributions of the Thesis

This thesis extends the Brent-Luk-VanLoan systolic array for real SVD to handle input matrices with complex elements. The expandable array structure of the Brent-Luk-VanLoan systolic array with the 2×2 processors is preserved. The SVD-Jacobi method with the two-angle, two-sided rotation scheme used in the Brent-Luk-VanLoan systolic array, is modified to a two-step, twelve-angle¹ two-sided rotation scheme to account for complex matrix elements.

The two-step SVD scheme is derived from simple two-sided unitary transformations developed to ensure hardware and performance efficiency. Each unitary transformation step in the diagonalization procedure, is identical in structure to permit pipelined systolic execution.

The two-sided unitary transformation step developed is shown to be efficiently implementable in hardware using CORDIC. A few modifications to the CORDIC scheme, necessary to ensure correct application of the unitary transformations at each step, are detailed. These modifications extend the range of vector rotations possible through CORDIC.

An architecture for the complex 2×2 processor, based on the CORDIC modules developed by Cavallaro and Luk [14] for the real 2×2 processor, is presented. An analysis of the area/time complexity of the complex 2×2 processor is compared with a similar analysis for the real 2×2 processor. Also, a performance comparison of the real and complex systolic SVD schemes is shown.

Results from the simulation of the algorithm using the Connection Machine², a SIMD (Single Instruction Multiple Data) [31] computer with good inter-processor

¹three left and three right rotation angles per step

²*Connection Machine*TM is a registered trademark of Thinking Machines Corporation, Cambridge, MA.

communication capability, are compared with LINPACK/EISPACK [26, 33] routines for validation. The simulation also shows the convergence behavior of the complex SVD scheme.

1.4 Overview of the Thesis

The next two chapters review previous work in the context of systolic arrays for SVD, CORDIC algorithms and Jacobi-type methods. In the following chapter, the serial and parallel SVD algorithms are presented. Jacobi methods for the eigenvalue and the singular value decompositions, which are fundamental to parallel SVD algorithms are also elaborated. The chapter concludes with a brief survey of the various parallel architectures that have been proposed for the SVD.

Chapter 3 presents the CORDIC algorithms in detail. The capabilities and the limitations of the CORDIC algorithms are discussed. Also, the use of CORDIC for the implementation of the real 2×2 SVD processor is motivated. The Cavallaro-Luk CORDIC based architectures for the real 2×2 SVD processor are then presented.

Further chapters contain the original contributions of the thesis. Chapter 4 presents the complex SVD problem. Various techniques useful in the development of a diagonalization scheme for a complex 2×2 matrix are developed. A two-step diagonalization scheme is then proposed for the SVD of a complex 2×2 matrix.

Chapter 5 explains the enhancements to the basic CORDIC algorithm necessary for the hardware implementation of the two-step diagonalization scheme proposed in Chapter 4. A performance efficient architecture based on CORDIC for the complex 2×2 processor is then proposed.

Chapter 6 reviews the systolic algorithm due to Brent-Luk-VanLoan and details modifications to the Brent-Luk-VanLoan systolic array necessary to efficiently im-

plement the two-step diagonalization scheme proposed in the previous chapter. The chapter ends with a comparison of the Brent-Luk-VanLoan systolic array with the proposed complex SVD array.

Chapter 7 discusses the simulation of the systolic array for complex SVD on the Connection Machine and the experimental validation of the algorithm. The behavior of the algorithm is observed to predict the number of sweeps required for convergence.

Finally, Chapter 8 presents the conclusions and summarizes the thesis. Extensions to the thesis and possible future work are also discussed.

Chapter 2

Parallel Architectures for Real SVD

In the introduction to the thesis, the importance of parallel architectures for real-time signal processing and the utility of the SVD as a matrix factorization technique was emphasized. In reviewing the fundamental concepts and previous work in this context, this chapter lays a foundation for the others that follow.

2.1 Singular Value Decomposition

A singular value decomposition (SVD) of a matrix $M \in C^{m \times n}$ is given by

$$M = U\Sigma V^H, \quad (2.1)$$

where $U \in C^{m \times m}$ and $V \in C^{n \times n}$ are unitary matrices and $\Sigma \in R^{m \times n}$ is a real non-negative “diagonal” matrix. Since $M^H = V\Sigma^T U^H$, we may assume $m \geq n$ without loss of generality. The singular values may be arranged in any order, for if $P \in R^{m \times m}$ and $Q \in R^{n \times n}$ are permutation matrices such that $P\Sigma Q$ remains “diagonal”, then

$$M = (UP^T)(P\Sigma Q)(Q^T V^H)$$

is also an SVD. It is customary to choose P and Q so that the singular values are arranged in non-increasing order:

$$\sigma_1 \geq \dots \geq \sigma_r > 0, \quad \sigma_{r+1} = \dots = 0,$$

where $r = \text{rank}(M)$.

If the matrices U , Σ and V are partitioned by columns as

$$U = [u_1, u_2, \dots, u_m], \Sigma = \text{diag}[\sigma_1, \sigma_2, \dots, \sigma_n] \text{ and } V = [v_1, v_2, \dots, v_n],$$

then σ_i is the i^{th} singular value of M , and u_i and v_i are the left and right singular vectors corresponding to σ_i . If M is real, then the unitary matrices U and V are real and hence orthogonal.

2.2 SVD Algorithms

The importance of the SVD as a matrix factorization technique is underscored by the variety of algorithms available. They range from serial algorithms to parallel Jacobi-type methods. On a conventional uniprocessor system, the most commonly used procedure is the Golub-Kahan-Reinsch [35, 37] SVD algorithm.

The Golub-Kahan-Reinsch algorithm is implemented both in LINPACK [26] and EISPACK [33]. The first step in the Golub-Kahan-Reinsch SVD algorithm is the bi-diagonalization of the matrix. This is followed by an iterative diagonalization of the bi-diagonal matrix to complete the SVD. The time complexity of the Golub-Kahan-Reinsch algorithm is $O(mn^2)$.

An improvement to the Golub-Kahan-Reinsch procedure was suggested by Chan [15]. The improved algorithm is more efficient for an $m \times n$ matrix, unless $m \approx n$. A comparison of the two procedures is detailed in [36]. Serial SVD algorithms that have been suggested for complex matrices are due to Businger and Golub [10] and Bunse-Gerstner and Gragg [9].

The advances in parallel architectures and algorithms have given rise to a number of parallel SVD schemes [5, 6, 30, 45, 65, 68]. Algorithms for the computation of the SVD, specific to some parallel architectures are also known [28, 59]. The parallel

architectures and algorithms are indispensable where real-time processing is required [72].

On linear systolic arrays, the most efficient SVD algorithm is the Jacobi-like algorithm given by Brent and Luk [5] which requires $O(mn \log n)$ time and $O(n)$ processors. Jacobi-type procedures are extremely amenable to parallel computation [66] as evidenced by the number of such schemes that have been proposed [5, 59, 65]. They have been applied to the symmetric eigenvalue problem [5], the QR-decomposition [57] and the Schur decomposition [74].

2.3 The SVD-Jacobi Method

The classical method of Jacobi uses a sequence of plane rotations to diagonalize a symmetric matrix $M \in R^{n \times n}$. A Jacobi rotation by an angle θ in the i, j plane is denoted by the matrix $J(i, j, \theta)$ where,

$$\begin{aligned} j_{pp} &= 1 & \forall (p \neq i, j), \\ j_{ii} &= \cos \phi, & j_{ij} &= \sin \phi, \\ j_{ji} &= -\sin \phi, & j_{jj} &= \cos \phi, \end{aligned} \tag{2.2}$$

and all other $j_{pq} = 0$.

The Jacobi method systematically reduces the quantity

$$\text{off}(M) = \sqrt{\sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij}^2}, \tag{2.3}$$

the “norm” of the off-diagonal elements. The basic step in a Jacobi procedure involves choosing an index pair (i, j) that satisfies $1 \leq i < j \leq n$, computing a cosine-sine pair (c, s) such that

$$\begin{bmatrix} m'_{ii} & m'_{ij} \\ m'_{ji} & m'_{jj} \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} m_{ii} & m_{ij} \\ m_{ji} & m_{jj} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{2.4}$$

is diagonal, and overwriting M with

$$M' = J^T M J,$$

where $J = J(i, j, \theta)$. The classical Jacobi algorithm maximizes the reduction of $\text{off}(M)$ in (2.4) by choosing (i, j) so that m_{ij}^2 is maximal.

The drawback of the classical Jacobi scheme is that the updates require $O(n)$ time while the search for optimal (i, j) is $O(n^2)$. This problem is overcome by fixing the sequence of sub-problems to be solved in advance. Common schemes are the cyclic-by-row and the cyclic-by-column procedures where the pair (i, j) is chosen in a row-by-row, or column-by-column fashion respectively. For example, if $n = 4$ the pair sequence is repeated as:

$$(i, j) = (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4), \dots$$

Formally, the cyclic-by-row ordering for the pair (i, j) may be expressed as

$$(i_0, j_0) = (1, 2),$$

$$(i_{k+1}, j_{k+1}) = \begin{cases} (i_k, j_k + 1), & \text{if } i_k < n - 1, j_k < n, \\ (i_k + 1, i_k + 2), & \text{if } i_k < n - 1, j_k = n, \\ (1, 2), & \text{if } i_k = n - 1, j_k = n, \end{cases}$$

and the cyclic-by-column ordering as

$$(i_0, j_0) = (1, 2),$$

$$(i_{k+1}, j_{k+1}) = \begin{cases} (i_k + 1, j_k), & \text{if } i_k < j_k - 1, j_k \leq n, \\ (1, j_k + 1), & \text{if } i_k = j_k - 1, j_k < n, \\ (1, 2), & \text{if } i_k = n - 1, j_k = n. \end{cases}$$

A “parallel ordering” described by Brent and Luk [5] is particularly useful in the context of parallel architectures and illustrated in the $n = 8$ case by

$$\begin{aligned}
 (p, q) = & (1, 2), (3, 4), (5, 6), (7, 8), \\
 & (1, 4), (2, 6), (3, 8), (5, 7), \\
 & (1, 6), (4, 8), (2, 7), (3, 5), \\
 & (1, 8), (6, 7), (4, 5), (2, 3), \\
 & (1, 7), (8, 5), (6, 3), (4, 2), \\
 & (1, 5), (7, 3), (8, 2), (6, 4), \\
 & (1, 3), (5, 2), (7, 4), (8, 6).
 \end{aligned}$$

For any even n , $n - 1$ steps are required for the Brent-Luk “parallel ordering” as shown below

$$\begin{array}{cccccccc}
 1 & & 3 & \rightarrow & 5 & \rightarrow & \cdots & \rightarrow & (n-3) & \rightarrow & (n-1) \\
 & & \nearrow & & & & & & & & \downarrow \\
 2 & \leftarrow & 4 & \leftarrow & 6 & \leftarrow & \cdots & \leftarrow & (n-2) & \leftarrow & n
 \end{array} \tag{2.5}$$

The rotation pairs in each column of (2.5) can be calculated concurrently. This fact was exploited by Brent and Luk [5] and later Brent, Luk and VanLoan [6] in the development of systolic arrays where a sweep of the “parallel ordering” could be executed in $O(n)$ time.

Jacobi methods can be readily extended to the SVD of arbitrary matrices. A two-sided Jacobi-SVD procedure was first suggested by Kogbetliantz [49]. The theoretical framework for the two-sided Jacobi SVD method for square matrices was provided by Forsythe and Henrici [32]. The proof and conditions for convergence for the two-sided cyclic-Jacobi-SVD are given in [32] and the asymptotic convergence rate was shown to be quadratic by Wilkinson [82].

Hansen [39] discusses the convergence properties associated with various orderings for the serial Jacobi method. He defines a certain “preference factor” for comparing

different ordering schemes. The “parallel ordering” is quite desirable in that it optimizes the “preference factor” as $n \rightarrow \infty$. The “parallel ordering”, like the cyclic orderings, is ultimately quadratically convergent. The proof of convergence for the “parallel ordering” was given by Park and Luk [60].

Theoretically, the SVD of a matrix M may be computed from the eigenvalue decomposition of $M^T M$. However, the numerical difficulties associated with the computation of $M^T M$ makes the approach impractical. Hestenes [43] suggested a “one-sided” approach, which applies the Jacobi method implicitly, to overcome the difficulty in the formation of $M^T M$. A discussion on the relative difficulties and accuracies of the different approaches to computing the SVD, and the justification for a systolic array can be found in [6].

2.4 SVD of a Real 2×2 Matrix

The Jacobi eigenvalue procedure was extended to the SVD of a square matrix $M \in C^{n \times n}$ by Kogbetliantz [49] and Forsythe and Henrici [32]. The matrix is diagonalized via a sequence of 2×2 SVDs. Corresponding to the two-sided rotation described in (2.4) for the eigenvalue decomposition of a symmetric 2×2 matrix, a similar scheme can be devised for the SVD of an arbitrary 2×2 matrix. Discussion of the SVD procedure for a complex 2×2 matrix is postponed until Chapter 4.

The real 2×2 SVD problem can be expressed as the computation of the cosine-sine pairs (c_l, s_l) and (c_r, s_r) such that

$$R(\theta_r)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_l) = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}, \quad (2.6)$$

where

$$R(\theta) = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \quad (2.7)$$

is a 2×2 rotation matrix and the input 2×2 matrix is

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}. \quad (2.8)$$

Several approaches to determining the angles θ_l and θ_r in (2.6) are mentioned in [6]. The approach suggested in [32] is to first determine θ_{sum} and θ_{diff} as solutions to

$$\theta_{sum} = (\theta_r + \theta_l) = \tan^{-1} \left(\frac{c+b}{d-a} \right), \quad (2.9)$$

$$\theta_{diff} = (\theta_r - \theta_l) = \tan^{-1} \left(\frac{c-b}{d+a} \right). \quad (2.10)$$

The two angles θ_l and θ_r , can then be separated from θ_{sum} and θ_{diff} . Convergence for the Jacobi-method with a cyclic ordering is guaranteed if (2.9) and (2.10) hold.

An alternative method is to symmetrize the 2×2 matrix by rotation through $\theta_l - \theta_r$ followed by diagonalization using (2.4). A real 2×2 matrix as defined in (2.8) can be symmetrized by a one-sided rotation as

$$R(\theta_{sym})^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} p & q \\ q & r \end{bmatrix}, \quad (2.11)$$

where θ_{sym} is defined as

$$\theta_{sym} = \tan^{-1} \left(\frac{b-c}{a+d} \right). \quad (2.12)$$

After symmetrization, the matrix can be diagonalized by (2.4) as

$$R(\theta_{diag})^T \begin{bmatrix} p & q \\ q & r \end{bmatrix} R(\theta_{diag}) = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}, \quad (2.13)$$

where

$$\theta_{diag} = \frac{1}{2} \tan^{-1} \left(\frac{2q}{r-p} \right). \quad (2.14)$$

2.5 Systolic Architectures for Real SVD

The advent of parallel processing and systolic arrays, combined with real-time signal processing requirements [72], has given rise to a variety of SVD arrays. The

architectures that have been proposed range from linear arrays through triangular configurations and mesh-connected processor structures.

Jacobi-type methods are easily adapted to matrix computations using processor arrays. Unfortunately, Jacobi-SVD algorithms are applicable only to square matrices. For an $m \times n$ matrix $M \in R^{m \times n}$, the obvious strategy is to first compute the QR-decomposition (QRD)

$$M = Q \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad (2.15)$$

where the matrix $Q \in R^{m \times m}$ is orthogonal and the matrix $R \in R^{n \times n}$ is upper triangular, and then apply an SVD procedure to R . This approach is particularly suitable for the case where $m \gg n$ [15].

QRD arrays have been thoroughly studied [2, 4, 34, 41, 47, 58, 65]. However, the interfacing of QRD and SVD arrays is a difficult problem. The QRD algorithm in [58] is the only algorithm implementable on the square array of Brent-Luk-VanLoan [6]. Luk [57] has proposed a triangular array that can compute both the QRD and the SVD of a $m \times n$, $m \geq n$, matrix. The array uses $\frac{1}{4}n^2 + O(n)$ processors and requires $O(m + nS)$ time, where S denotes the number of sweeps. The array is extended to a rectangular configuration with $\frac{1}{2}mn + O(m)$ processors for the computation of singular vectors.

The systolic array of Brent-Luk [5] uses the one-sided orthogonalization method due to Hestenes [43] on a linearly connected mesh of $O(n)$ processors and requires $O(mn \log n)$ steps to compute a singular value. A two-dimensional array, with $O(mn)$ processors and a non-planar interconnection structure, that requires $O(n \log m)$ time was also proposed. The method is quadratically convergent and the conjecture is that 6 to 10 iterations provide for sufficient numerical accuracy.

In [6], Brent-Luk-VanLoan describe a similar architecture of $O(n^2)$ processors that implements a cyclic Jacobi-method for the SVD in $O(m + n \log n)$ steps. With reference to the arrays of Brent-Luk-VanLoan, Schreiber in [67], suggested methods to cope with problems that do not match the array size.

In [68], Schreiber proposed a kn -processor design which reduces a dense matrix to upper triangular form of bandwidth $k + 1$ in time $O(mn/k)$. A $k(k + 1)$ processor array from [41] is used to implement a SVD iteration for $(k + 1)$ -diagonal matrices, analogous to the Golub-Reinsch iteration for bi-diagonal matrices, in $6n + O(k)$ time.

Ipsen [45] suggested systolic arrays for the SVD of an $m \times n$, $m \geq n$, matrix A of bandwidth w . After matrix A is reduced to a bi-diagonal form B by means of Givens plane rotations [36], the singular values of B are computed by the Golub-Reinsch [37] iteration. $O(wn)$ processors that can compute or apply a plane rotation accomplish the reduction to bi-diagonal form in $O(np)$ steps, where p is the number of superdiagonals. A constant number of processors is then needed to determine each singular value in $6n$ steps. The singular vectors are computed by rerouting the rotations through the arrays used for the reduction to bi-diagonal form, or by employing another array of $O(wm)$ processors.

In a typical SVD algorithm, costly square root and division operations are required to apply plane rotations. The co-ordinate rotation algorithms, CORDIC, which allow easy computation of inverse tangents and vector rotations, have proven extremely useful in this context by effectively mapping the algorithm to hardware. The next chapter details the essentials of CORDIC algorithms and presents various hardware and performance efficient architectures that have been proposed for the implementation of special-purposé SVD arrays.

Chapter 3

CORDIC Architectures for Real SVD

Special-purpose hardware oriented arithmetic techniques can improve performance of scientific algorithms. These techniques are indispensable in the context of hardware design for special-purpose applications. This chapter reviews the CORDIC algorithms and architectures which have proven extremely useful in the computation of the SVD.

3.1 CORDIC Algorithms

The COordinate Rotation DIgital Computer (CORDIC) algorithms were first presented in 1959 by Volder [80]. A similar idea, particularly effective in decimal radix computations, was presented by Meggitt [61] in 1962. Daggett [18] discussed the use of CORDIC for decimal-binary conversions. Further theoretical work was done by Walther [81] in 1971 to realize a unified algorithm and demonstrate the applicability of CORDIC to various transcendental and hyperbolic functions. Cochran [16] benchmarked various algorithms and found that CORDIC techniques surpass alternative methods in scientific calculator applications.

The CORDIC algorithms allow fast iterative hardware calculation of \sin , \cos , \arctan , \sinh , \cosh , $\operatorname{arctanh}$, products, quotients, square roots, and conversion between binary and mixed radix number systems. The CORDIC algorithms have some limitations which are discussed later in this chapter. However, these limitations do not overshadow the utility of the technique. By exploiting the properties and tailoring

the algorithms suitably, it is possible to design efficient hardware/software structures that present significant gains over conventional architectures. Real-time signal processing concerns [1], combined with the performance and hardware advantage in the VLSI setting, makes CORDIC an attractive alternative to traditional arithmetic units for special-purpose hardware design.

Plane rotations and generation of rotation angles through inverse tangent calculations were shown to be fundamental to SVD algorithms in the previous chapter. In a conventional sequential computer, the calculation of rotation angles through costly square root and division operations, or computation of sines/cosines in software, proves expensive. Also, matrix-vector products involve costly multiplication and division operations.

In the context of special-purpose SVD architectures, primitive CORDIC operations like vector rotations and inverse tangent calculations help increase efficiency by more effectively mapping the algorithm to hardware [71, 73]. Special VLSI structures have been shown possible for the SVD [14, 29, 62].

Coordinate Systems

The range of the CORDIC algorithms is best expressed through the use of a generalized polar coordinate system. The CORDIC algorithms of interest relate to a two-dimensional coordinate system. Higher dimensions were considered by Delosme in [22, 20].

Let (x, y) be the planar orthogonal coordinates for a point P and introduce a generalized polar coordinate system (R, A) by

$$\begin{cases} R = \sqrt{x^2 + my^2}, \\ A = \left(\frac{1}{\sqrt{m}}\right) \tan^{-1}\left(\frac{y\sqrt{m}}{x}\right), \end{cases} \quad \begin{cases} x = R \cos(A\sqrt{m}), \\ y = \left(\frac{R}{\sqrt{m}}\right) \sin(A\sqrt{m}). \end{cases} \quad (3.1)$$

Here m is a fixed constant whose value is one of 1, -1 or 0. When $m = 1$, (3.1) reduces to the familiar interconversion relations between the orthogonal coordinate system and the polar coordinate system. We need to impose an interpretation for the cases when $m = 0$ and $m = -1$. Precisely,

$$A = \left(\frac{y}{x}\right) \text{ for } m = 0, \quad A = \tanh^{-1}\left(\frac{y}{x}\right) \text{ for } m = -1. \quad (3.2)$$

The resulting radius (R) and angle (A) expressions for the different coordinate systems characterized by m are tabulated in Table 3.1.

For simplicity, we always assume $x \geq 0$, and further $x \geq |y| \geq 0$ for $m = -1$. It is easily seen that

$$A = \frac{S}{2R^2},$$

where S is the area of the domain surrounded by x axis, the radius vector OP and the curve of constant radius R passing through P (Figure 3.1).

Fundamental Transformations

Take a linear transformation from a point $P_j = (x_j, y_j)$ to $P_{j+1} = (x_{j+1}, y_{j+1})$ given by

$$\begin{cases} x_{j+1} = x_j + m\delta_j y_j, \\ y_{j+1} = y_j - \delta_j x_j, \end{cases} \quad (3.3)$$

Coordinate Systems		
Mode	Radius	Angle
General	$\sqrt{x^2 + my^2}$	$\left(\frac{1}{\sqrt{m}}\right) \tan^{-1}\left(\frac{y\sqrt{m}}{x}\right)$
Circular	$\sqrt{x^2 + y^2}$	$\tan^{-1}\left(\frac{y}{x}\right)$
Linear	x	$\frac{y}{x}$
Hyperbolic	$\sqrt{x^2 - y^2}$	$\tanh^{-1}\left(\frac{y}{x}\right)$

Table 3.1: Radius and Angle in Coordinate Systems

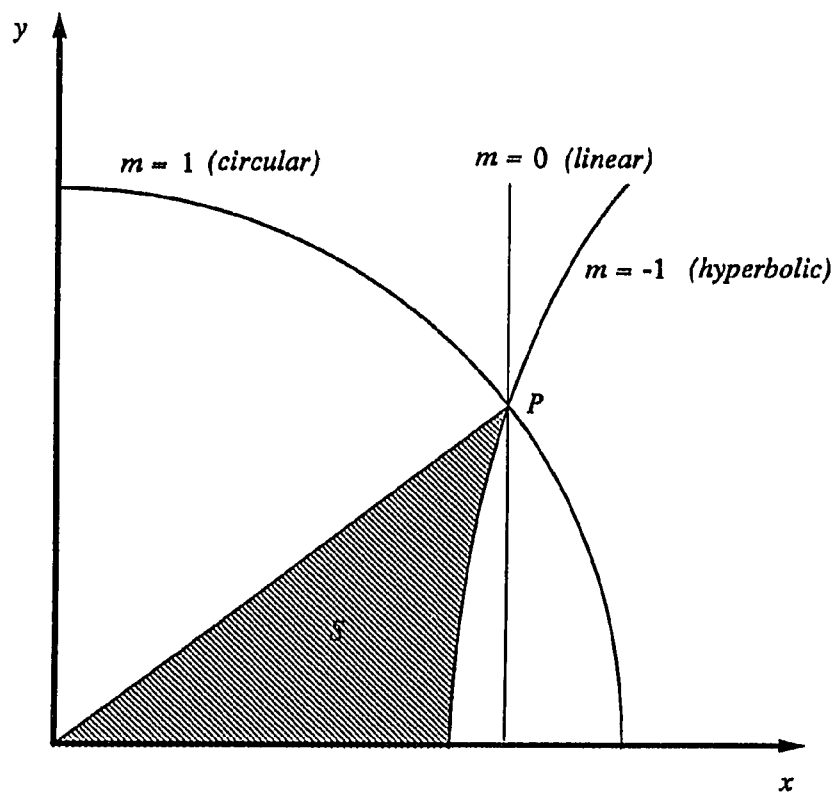


Figure 3.1: Coordinate Systems

where m is the parameter of the coordinate system (3.1) and δ_j is an arbitrary constant. The transformation (3.3) gives

$$\begin{aligned} A_{j+1} &= A_j - \alpha_j, \\ R_{j+1} &= R_j \times k_j, \end{aligned}$$

in the generalized polar coordinate system (3.1), where

$$\begin{cases} \alpha_j = \left(\frac{1}{\sqrt{m}}\right) \tan^{-1}(\sqrt{m}\delta_j), \\ k_j = \sqrt{1 + m\delta_j^2}. \end{cases} \quad (3.4)$$

Starting from $P_0 = (x_0, y_0)$, iterate the transformations (3.3) under a suitable sequence of constants $\delta_0, \delta_1, \dots, \delta_{n-1}$ until $P_n = (x_n, y_n)$. Then

$$\begin{aligned} A_n &= A_0 - \alpha_n, \\ R_n &= R_0 \times K_n, \end{aligned}$$

where

$$\alpha_n = \sum_{j=0}^{n-1} \alpha_j, \quad K_n = \prod_{j=0}^{n-1} k_j. \quad (3.5)$$

Introducing a third variable z and transforming it as

$$z_{j+1} = z_j + \alpha_j \quad (3.6)$$

simultaneously with (3.3), the final values are given by

$$\begin{cases} x_n = K_n \left[x_0 \cos(\alpha_n \sqrt{m}) + \left(\frac{y_0}{\sqrt{m}}\right) \sin(\alpha_n \sqrt{m}) \right], \\ y_n = K_n \left[y_0 \cos(\alpha_n \sqrt{m}) - \left(\frac{x_0}{\sqrt{m}}\right) \sin(\alpha_n \sqrt{m}) \right], \\ z_n = z_0 + \alpha_n, \end{cases}$$

where α_n and K_n are given by (3.5).

CORDIC Operation Modes

Consider the iteration of the transformations (3.3) and (3.6) under a suitable sequence of constants $\{\delta_j\}$ in the following two ways

Case I : y is forced to zero,

Case II : z is forced to zero.

The final values of the variables x , y and z are shown in Figure 3.2, where

$$K_{\pm 1} = \prod_{j=0}^{\infty} \sqrt{1 \pm \delta_j^2}.$$

Observing that

$$\sqrt{(t+c)^2 - (t-c)^2} = 2\sqrt{c}\sqrt{t},$$

$$e^t = \cosh t + \sinh t,$$

and

$$\frac{1}{2} \log \frac{t}{c} = \tanh^{-1} \left(\frac{t+c}{t-c} \right),$$

Figure 3.2 contains all elementary standard functions such as square root, log, exp, sin, cos, and arctan as well as multiplication and division. Iteration of Case I for $m = 0$ is essentially the non-restoring division algorithm.

The CORDIC transformations are quite amenable to computer arithmetic. The key contribution of Volder [80] and Walther [81] was to set δ_j as a power of the machine radix. Digital computers use binary arithmetic. It is therefore convenient to choose

$$\delta_j = \pm 2^{-j} \text{ (or } \pm 2^{-j-1} \text{)}, \quad j = 0, 1, 2, \dots, n, \quad (3.7)$$

where n is the word length or bit-precision of the machine. When $m = -1$, slight modification is necessary for convergence [44]. Let

$$\begin{cases} \epsilon_j = 2^{-j}, & \beta_j = \left(\frac{1}{\sqrt{m}} \right) \tan^{-1} (2^{-j} \sqrt{m}) \\ \delta_j = \pm \epsilon_j, & \alpha_j = \text{sign}(\delta_j) \beta_j \end{cases} \quad (3.8)$$

The constants β_j may be pre-calculated and are necessary only for j up to $n/3$, where n is the bit-precision, since

$$\beta_j = \epsilon_j + \frac{m}{3} \epsilon_j^3 + \frac{1}{5} \epsilon_j^5 + \dots$$

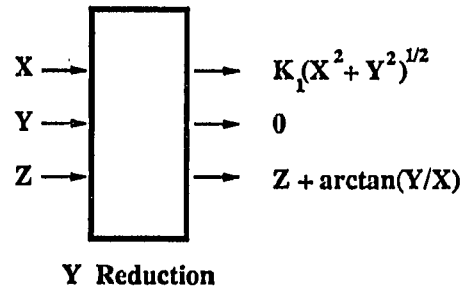
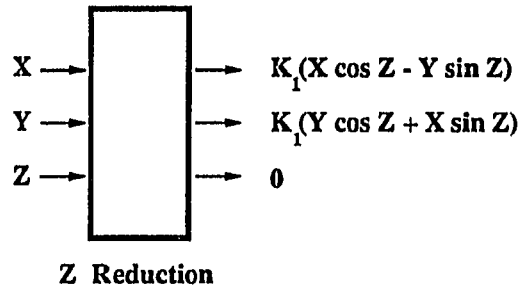
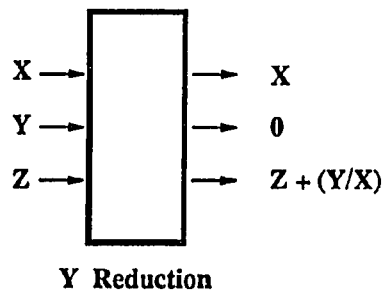
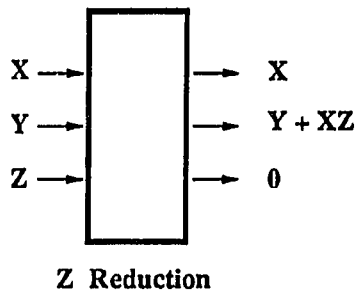
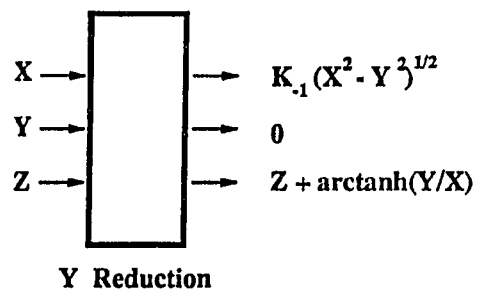
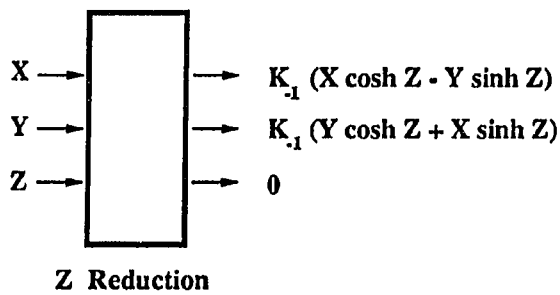
CIRCULAR MODELINEAR MODEHYPERBOLIC MODE

Figure 3.2: Results of CORDIC Iterations

may be replaced by ϵ_j , if ϵ_j^3 and higher terms are negligible.

The transformations (3.3) and (3.6) are now modified as

$$\begin{array}{l} \text{for } \delta_j > 0 \\ \left\{ \begin{array}{l} x_{j+1} = x_j + my_j 2^{-j} \\ y_{j+1} = y_j - x_j 2^{-j} \\ z_{j+1} = z_j + \beta_j \end{array} \right. \quad (3.9) \end{array} \quad \begin{array}{l} \text{for } \delta_j < 0 \\ \left\{ \begin{array}{l} x_{j+1} = x_j - my_j 2^{-j} \\ y_{j+1} = y_j + x_j 2^{-j} \\ z_{j+1} = z_j - \beta_j \end{array} \right. \quad (3.10) \end{array}$$

In Case I, select (3.9) if $y_j \geq 0$, and (3.10) otherwise. In Case II, select (3.9) if $z_j < 0$, and (3.10) otherwise. The CORDIC operation modes of Cases I and II are also referred to as y -reduction and z -reduction respectively, for obvious reasons. The conditions for the choice of transformations (3.9) or (3.10) is tabulated in Table 3.2.

It is remarkable that the transformation (3.9) or (3.10) is possible only by addition, subtraction, shifting and the use of constants. Thus, CORDIC algorithms greatly simplify the design of arithmetic units for special-purpose hardware and VLSI implementation.

Convergence Issues

Walther [81] has shown that the domain of convergence of the CORDIC algorithms is limited by

$$\beta_j - \sum_{k=j+1}^{n-1} \beta_k \leq \beta_{n-1} \quad (j = 0, 1, 2, \dots), \quad (3.11)$$

Transformation	CORDIC operation mode	
	Case I	Case II
Eqn. (3.9)	$y_j \geq 0$	$z_j < 0$
Eqn. (3.10)	$y_j < 0$	$z_j \geq 0$

Table 3.2: Transformations for CORDIC Operation Modes

where n is the number of iterations of transformations (3.9) or (3.10) used.

When $m = 1$, it is easy to verify (3.11) for $\beta_j = \tan^{-1}(2^{-j})$, since $\tan^{-1}(x)$ is convex in $x \geq 0$, i.e.,

$$\tan^{-1}(2^{-j}) < 2 \tan^{-1}(2^{-(j+1)}).$$

Thus, for the circular mode, the transformations (3.9) and (3.10) always converge, when

$$z_0 < \sum_{j=0}^{\infty} \tan^{-1}(2^{-j}) = 1.74\dots > \frac{\pi}{2}, \quad (3.12)$$

which covers the closed right half-plane.

When $m = -1$, the sequence $\beta_j = \tanh^{-1}(2^{-j})$ ($j = 1, 2, \dots$) does not satisfy the convergence condition (3.11), since $\tanh^{-1}(x)$ is concave in $x \geq 0$, i.e.,

$$\tanh^{-1}(2^{-j}) > 2 \tanh^{-1}(2^{-(j+1)}).$$

Walther [81] presented an empirical method to solve the hyperbolic convergence problem by repeating certain iterations. The method is based on the fact that

$$\beta_j - \left(\sum_{k=j+1}^{n-1} \beta_k \right) - \beta_{3j+1} < \beta_{n-1} \quad \text{for } \beta_j = \tanh^{-1}(2^{-j}) \quad (j \geq 1). \quad (3.13)$$

As β_j decreases with j , the convergence condition (3.11) will be satisfied if iterations at

$$j = 4, 13, 40, 121, \dots$$

are repeated. The general formula of the above sequence is given by the recurrence relation

$$j_n = 3j_{n-1} + 1, \quad j_0 = 1.$$

The transformations (3.9) and (3.10) converge if and only if the initial values (x_0, y_0) satisfy the condition

$$\left| \tanh^{-1}\left(\frac{y_0}{x_0}\right) \right| < C = \left(\sum_{j=1}^{\infty} + \sum_{j=4,13,\dots} \right) \tanh^{-1}(2^{-j}) = 1.118\dots$$

A discussion on the convergence region for the hyperbolic mode and a proof for (3.13) can be found in [44].

Circular Mode

In the context of SVD algorithms, application of plane rotations and generation of rotation angles are of utmost interest. CORDIC in the circular mode ($m = 1$), allows easy conversion between orthogonal and polar coordinates. The CORDIC operation modes of y -reduction and z -reduction have added utility in the circular mode.

Y-Reduction for Inverse Tangent

In the circular mode, the y -reduction can be used to compute inverse tangents. Consider the results of transformations (3.9) and (3.10) for Case I (y -reduction) and $m = 1$ (circular mode) shown in Figure 3.2,

$$\begin{cases} x = K_1 \sqrt{x_0^2 + y_0^2}, \\ y = 0, \\ z = z_0 + \tan^{-1}(y_0/x_0). \end{cases} \quad (3.14)$$

If z_0 is chosen to be zero then (3.14) can be used to compute the quantity $\tan^{-1}(y_0/x_0)$. The result is available directly without the requirement of re-scaling.

Z-Reduction for Rotations

In the circular mode, the z -reduction can be used to yield a vector rotation or the sine and cosine of a given angle. Again, consider the results of transformations (3.9)

and (3.10) for Case II (z -reduction) and $m = 1$ (circular mode) shown in Figure 3.2,

$$\begin{cases} x = K_1 (x_0 \cos z_0 - y_0 \sin z_0), \\ y = K_1 (y_0 \cos z_0 + x_0 \sin z_0), \\ z = 0. \end{cases} \quad (3.15)$$

The values of x and y in (3.15) represent rotation of the vector (x_0, y_0) in the two-dimensional plane by the angle z_0 . However, unlike y -reduction, the final values are scaled by K_1 .

As a special case, the z -reduction can be used to calculate the sine and cosine of the angle z_0 . Suppose $x_0 = K_1^{-1}$ and $y_0 = 0$, then (3.15) may be re-written as

$$\begin{cases} x = K_1 K_1^{-1} \cos(z_0) = \cos(z_0), \\ y = K_1 K_1^{-1} \sin(z_0) = \sin(z_0), \\ z = 0. \end{cases} \quad (3.16)$$

Scale Factor Considerations

The CORDIC formulation is not yet complete since the vector (x_0, y_0) is not only rotated but also scaled at each iteration. This scaling is by a constant k_j given by (3.4). Thus, the final values of the variables x and y need to be adjusted for the z -reduction (Case II) CORDIC operation mode and the variable x for the y -reduction (Case I) CORDIC operation mode if desired.

From (3.4), it is clear that scaling is not a concern as far as the linear mode ($m = 0$) is concerned. Since the CORDIC mode of interest, as far as computing the SVD is considered, is the circular mode; further discussion is restricted to the case when $m = 1$. The basic ideas, though, easily extend to the hyperbolic mode.

In a digital computer with a finite word length or bit-precision n , only n iterations of the transformations (3.9) and (3.10) are required. This is due to the fact that n

iterations guarantee n bits of accuracy and additional iterations prove extraneous. However, $\log n$ bits are necessary to guard against round-off errors.

After n iterations, the values of the variables x and y for the z -reduction (Case II) CORDIC operation mode are given by

$$\begin{aligned}x_n &= K_n x_{final}, \\y_n &= K_n y_{final},\end{aligned}$$

where x_{final} and y_{final} are the desired final values and K_n is given by (3.5).

In many cases, fewer than n iterations are necessary for z to converge to zero. The storage of K_n for all possible values of n is necessary if early termination is permitted. This limitation of the CORDIC algorithms is presented in the discussion of techniques for latency reduction by Bridge, Fisher and Reynolds [7].

For the circular mode ($m = 1$), substituting (3.4) with (3.7) in (3.5) and using the fact that from (3.8)

$$\tan(\beta_j) = 2^{-j}$$

we have

$$K_n = \prod_{j=0}^{n-1} k_j = \prod_{j=0}^{n-1} \sqrt{1 + 2^{-2j}} = \prod_{j=0}^{n-1} \sec \beta_j. \quad (3.17)$$

As a last step in the CORDIC z -reduction operation, a scale factor correction needs to be performed to yield

$$\begin{aligned}K_{SFC}x_n &= K_{SFC}K_n x_{final} \approx Cx_{final}, \\K_{SFC}y_n &= K_{SFC}K_n y_{final} \approx Cy_{final},\end{aligned} \quad (3.18)$$

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

Single Scale Factor Correction

The single scale factor correction techniques proposed in the literature fall into two classes. They include either special scale factor compensation iterations [24, 40] or modified repetitive angle sequences [1, 21]

The most direct scheme, due to Walther [81], involves a multiplication by K_n^{-1} using the CORDIC hardware in the linear mode. The scheme is costly since it may require a full CORDIC “cycle” of n shifts and n additions. However, the binary representation of K_n^{-1} can be used to determine which shifts and additions truly need to be performed. The CORDIC processor is modified to perform special iterations for both the x and y variables of the form

$$x \leftarrow x + x_n 2^{-j}, \quad (3.19)$$

where selected multiples of x_n and y_n are accumulated. Approximately $(n/2)$ iterations are required to correct for the scale factor.

Haviland and Tuszynski [40] proposed a method whereby the special scale factor correction iterations,

$$x \leftarrow x - x 2^{-j} \quad (3.20)$$

are performed for both the x and y variables. This scheme also causes a multiplication of x_n and y_n by K_n^{-1} . Again, approximately $(n/2)$ iterations are required for this method.

Ahmed [1], seeks to make the constant, C (3.18), a power of the machine radix by repeating certain full CORDIC iterations. A final shift is used to clear the remaining scale constant. The proposed scheme uses almost n extra CORDIC iterations to achieve $C = 2$. As an extra benefit the method extends the domain of convergence of the CORDIC algorithm. However, as all CORDIC operations can be made to fall

within the basic CORDIC domain of convergence, the time penalty caused by almost n extra iterations does not make Ahmed's method attractive for the SVD.

Delosme [21] combines the methods of Ahmed and Haviland and Tuszynski by repeating both CORDIC rotation iterations and special scale iterations to produce a low overhead scale factor correction scheme. In this method, approximately $(n/4)$ iterations are required and a $C = 2$ is generated.

Yang and Böhme [83] suggested a scheme for single scale factor correction requiring $n_k = \lceil n/4 \rceil$ additional iterations. A computer program was devised to systematically examine all relevant combinations of the shift sequences $S(i) = (i = 0, 1, \dots, n - 1)$ with differences $S(i+1) - S(i) \in \{0, 1, 2\}$ for more optimization freedom. For efficient scaling correction, they proposed an additional sequence n_k shift-add operations,

$$2^{-T(0)} \prod_{j=1}^{n_k} [1 + \eta(j)2^{-T(j)}] K_n = 1 + \Delta K \quad (T(j) \in I, \eta(j) = \pm 1).$$

These additional scaling iterations are parametrized by the set of signed integers $\{T(0), \eta(1)T(1), \dots, \eta(n_k)T(n_k)\}$ with $|\Delta K| \ll 2^{-n}$.

The above schemes require numerical methods to determine the appropriate sequence for a given word length. The number of iterations is chosen to reduce the approximation error to less than 2^{-n} . A detailed discussion on the various single scale factor correction schemes and a systematic approach to the two-sided scale factor correction as required for the SVD can be found in [11].

Cavallaro-Luk Two-Sided Scale Factor Correction

In the computation of the SVD, two vector rotations are performed for the diagonalization of the real 2×2 matrix (2.6). If the correction for the scale factor is postponed to the end of the second vector rotation, then the new scale factor is a square of the

single vector rotation scale factor K_n . From (3.17),

$$K_n^{-2} = \prod_{j=0}^{n-1} k_j^{-2} = \prod_{j=0}^{n-1} \frac{1}{\sqrt{1+2^{-2j}}} = \prod_{j=0}^{n-1} \cos^2 \beta_j, \quad (3.21)$$

the observation was made that each term in (3.21) resembles the special scale factor iterations shown in (3.20). If the CORDIC processor performs special iterations of the form (3.20), then the scale factor correction constant will be

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

where

$$C = 2 \quad \text{and} \quad J = \{1, 3, 5, \dots, (2\lceil \frac{n}{4} \rceil - 1)\} \quad \text{for } n > 0. \quad (3.22)$$

The scale factor as in (3.18) is removed, 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 is possible from (3.22) for any value of n . The Cavallaro-Luk two-sided rotation scheme greatly improves performance of the CORDIC two-sided vector rotation and results in a factor of 2 speedup over two applications of Delosme's single scale factor correction scheme.

3.2 Architecture for a Basic CORDIC Processor

In § 3.1, CORDIC algorithms were shown to greatly simplify the design of VLSI and special-purpose hardware. Efficient architectures, for the hardware implementation of the CORDIC recurrence relations (3.9) and (3.10), which minimize area/time complexity are necessary to exploit the performance advantages of the CORDIC algorithms to the fullest.

Ahmed [1] has investigated different architectures for a basic CORDIC processor capable of implementing the CORDIC recurrence relations in hardware. A fully parallel fixed-point CORDIC processor architecture due to Ahmed is shown in Figure 3.3.

Three parallel data paths are provided for the x , y and z recurrence equations. The processor is composed of several registers to hold the final and temporary values of x , y and z . For a fixed-point implementation, shifters are used in the x and y data paths to produce multiplication by 2^{-j} .

A read-only memory (ROM) is used to store the angles shown in Table 3.1 for the three CORDIC modes *viz.*, the circular ($m = 1$), the linear ($m = 0$) and the hyperbolic ($m = -1$) modes. Three fixed-point adders provide the additions required in the CORDIC recurrence relations (3.9) and (3.10).

Finally, a control unit oversees the overall sequencing and angle selection depending on which of the six CORDIC operational modes is chosen. The control unit is responsible for choosing between (3.9) and (3.10), based on the values in the y and z registers for the y -reduction and z -reduction operation modes respectively (Table 3.2).

3.3 Cavallaro-Luk Architectures for the Real 2×2 Processor

The SVD-Jacobi method (§ 2.3) requires the diagonalization of a 2×2 matrix as a basic step. The procedures for diagonalization were detailed in § 2.4. Based on the methods described in § 2.4 and using the basic fixed-point CORDIC processor architecture shown in § 3.2, Cavallaro and Luk [13] proposed four architectures, each with a different area/time complexity, for a real 2×2 CORDIC SVD processor. CORDIC has also been applied to the real SVD by Delosme [23]

To recapitulate, two diagonalization procedures for a real 2×2 matrix were presented in § 2.4. The first method is a direct two-angle two-sided rotation (2.6,2.9),

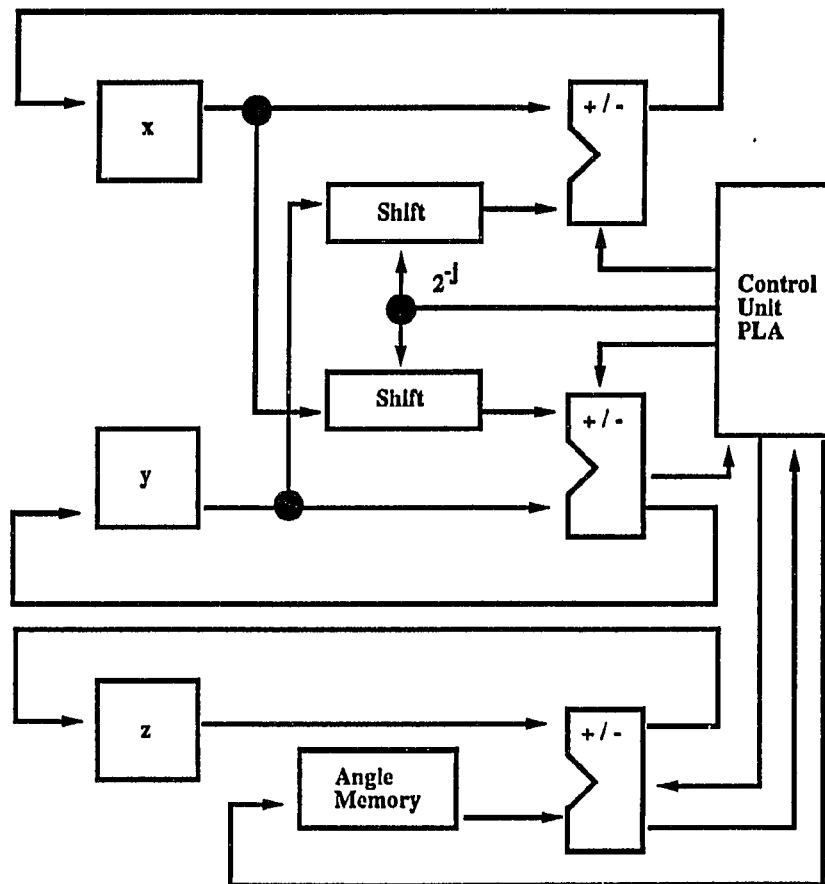


Figure 3.3: Parallel Fixed-point CORDIC Module

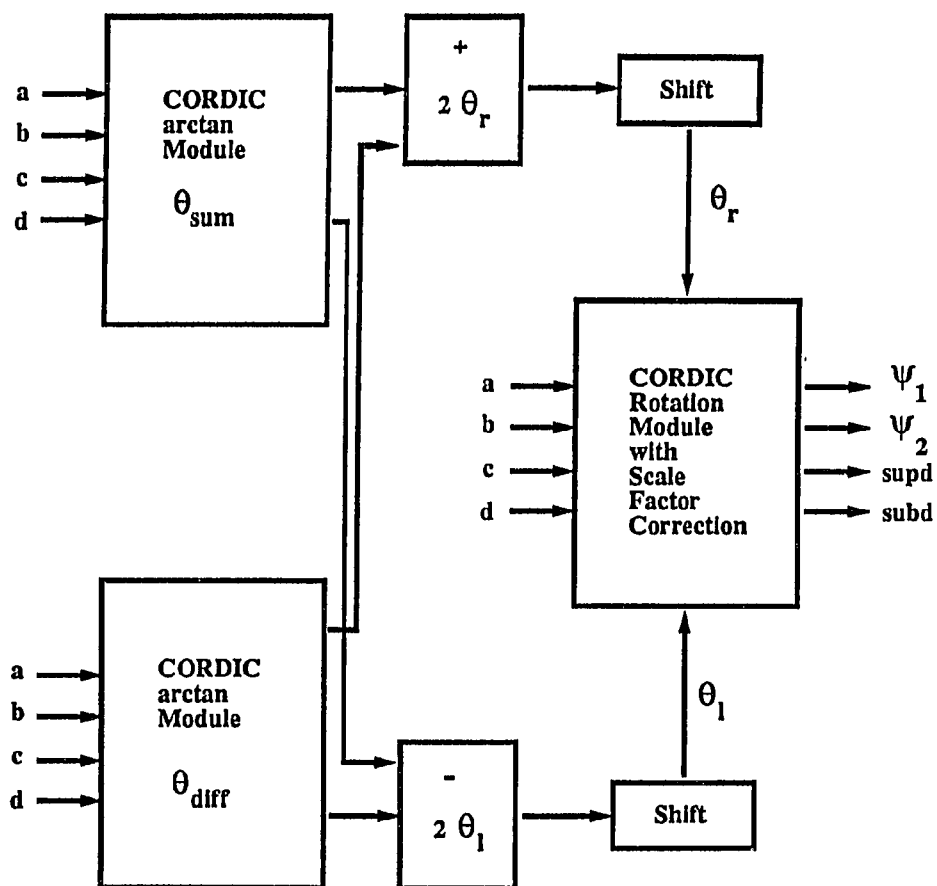


Figure 3.4: The Parallel Fixed-point SVD Processor Architecture

and the latter is a two-step method which requires symmetrization of the 2×2 matrix (2.11,2.12) followed by a one-angle two-sided rotation (2.13,2.14) to complete the diagonalization.

The first three architectures, *viz.* the Symmetrization-diagonalization method architecture, the Approximation method architecture and the Semi-parallel method architecture implement diagonalization using the two-step method while the fourth *viz.* the Parallel-diagonalization method implements the direct two angle rotation method. The fixed-point CORDIC architecture for the Parallel-diagonalization method due to Cavallaro and Luk is shown in Figure 3.4.

A detailed discussion on the motivations for the different approaches and architectures can be found in [11]. A area/time complexity analysis of the different architectures as presented in [11] is tabulated in Table 3.3 where T_C is the time required for one CORDIC vector rotation and A_C is the area occupied by a basic fixed-point CORDIC processor. The extra time of $0.25T_C$ is for two-sided scale factor correction.

A VLSI implementation of the real 2×2 processor for the Brent-Luk-VanLoan systolic array which uses CORDIC arithmetic has been reported in the literature [42]. A detailed analysis of the implementation and other issues relating to the use of fixed-point CORDIC for SVD computations is discussed in [50].

Architecture	Area	Time
Symm.-Diag.	$3A_C$	$4.25T_C$
Approx.	$3A_C$	$3.25T_C$
Semi-parallel	$4A_C$	$3.25T_C$
Parallel-Diag.	$2A_C$	$3.25T_C$

Table 3.3: Area/Time Complexity for Fixed-point SVD Architectures

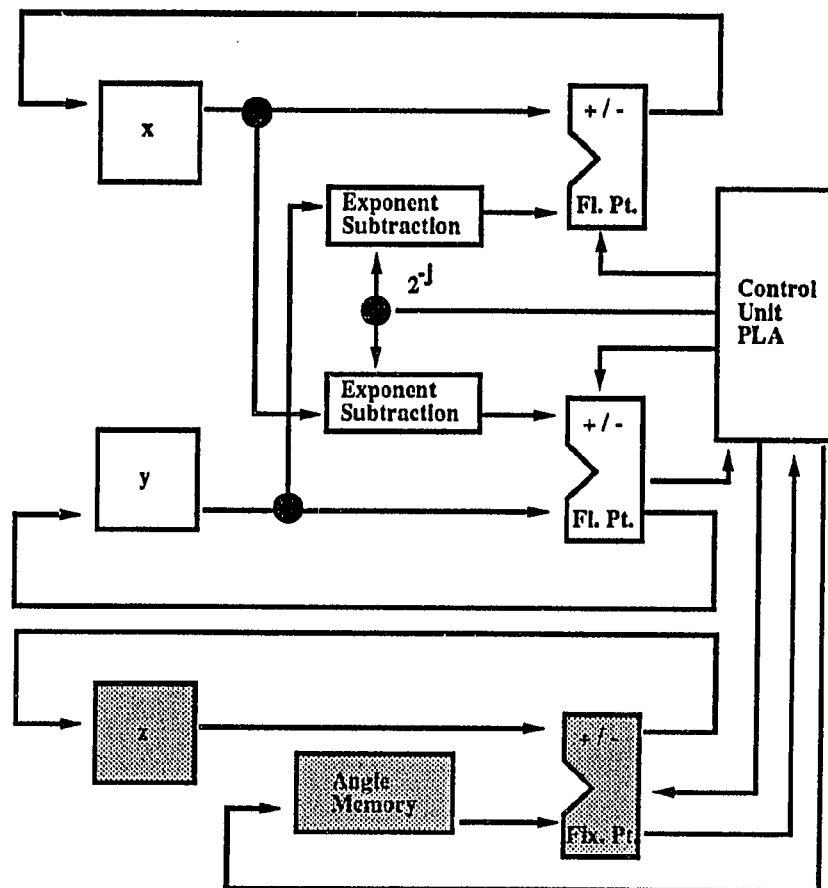


Figure 3.5: Parallel Floating-point CORDIC Module

The CORDIC algorithms were originally described for fixed-point data. However, for practical systems, a floating-point format is preferable. Walther [81] has dealt with the conversion from fixed-point to floating-point formats. Ahmed [1] points out that the shifting required in the fixed-point algorithm gets converted to exponent subtraction and mantissa alignment in floating-point.

Muller [63], Johnsson [48] and Ercegovic and Lang [27] have addressed the issue of floating-point CORDIC. Floating-point architectures were also considered in [11]. Cavallaro and Luk proposed a hybrid architecture that uses fixed-point angle memory with floating-point data paths (Figure 3.5).

In the next chapter, the SVD of an arbitrary matrix with possibly complex data elements is discussed. The SVD-Jacobi method when extended to a matrix with complex data, requires the diagonalization of a complex 2×2 matrix. A methodology for the diagonalization of a complex 2×2 matrix which is efficiently implementable in hardware, especially using CORDIC, is developed.

Chapter 4

The SVD of a Complex Matrix

Complex arithmetic and matrix transformations require a significantly greater number of computational steps than the corresponding real operations. Following a review of techniques known for complex matrix transformations, a hardware and performance efficient scheme for the diagonalization of complex matrices is proposed.

4.1 Complex Number Representation

Stating Euler's Formula as:

$$e^{i\theta} = \cos \theta + i \sin \theta , \quad (4.1)$$

a complex number, $z = z_r + i z_i$ may be written as

$$z = R_z e^{i\theta_z} , \quad (4.2)$$

where

$$R_z = \sqrt{z_r^2 + z_i^2} , \quad \theta_z = \tan^{-1} \left(\frac{z_i}{z_r} \right) . \quad (4.3)$$

Equation (4.3) is a polar co-ordinate system representation of z in the complex plane and the range of θ_z is

$$0 \leq \theta_z \leq 2\pi . \quad (4.4)$$

An alternate polar coordinate representation can be defined where the range of θ_z is

restricted to the principal values of *arctangent*

$$-\frac{\pi}{2} \leq \theta_z \leq \frac{\pi}{2}, \quad (4.5)$$

if (4.3) is modified as

$$R_z = \text{sign}(z_r) \sqrt{z_r^2 + z_i^2}, \quad \theta_z = \tan^{-1} \left(\frac{z_i}{z_r} \right). \quad (4.6)$$

4.2 Complex Givens Rotation

The Givens rotation [36] is an important technique which is used to selectively introduce a zero into a matrix. Givens rotations can be used in the QR Decomposition and are similar to the rotations used in the Jacobi [32] method for the symmetric eigenvalue decomposition (2.4) and the SVD (2.6). A real Givens rotation is given by

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad (4.7)$$

where

$$\theta = \tan^{-1} \left(\frac{b}{a} \right), \quad c = \cos \theta, \quad s = \sin \theta \quad \text{and} \quad r = \sqrt{a^2 + b^2}.$$

The Givens rotation can be generalized to handle the case of complex arithmetic.

Let

$$\begin{bmatrix} a_r + i a_i \\ b_r + i b_i \end{bmatrix} = \begin{bmatrix} A e^{i\theta_a} \\ B e^{i\theta_b} \end{bmatrix}$$

be an arbitrary complex vector. A complex Givens rotation can be described by two rotation angles as formulated in Coleman and Van Loan [17] as

$$\begin{bmatrix} \cos \theta_1 & \sin \theta_1 (e^{i\theta_2}) \\ -\sin_1 (e^{-i\theta_2}) & \cos \theta_1 \end{bmatrix} \begin{bmatrix} a_r + i a_i \\ b_r + i b_i \end{bmatrix} = \begin{bmatrix} r_r + i r_i \\ 0 \end{bmatrix}. \quad (4.8)$$

The above rotation matrix is derived by first applying the simple unitary transformation

$$U = \begin{bmatrix} e^{-i\theta_a} & 0 \\ 0 & e^{-i\theta_b} \end{bmatrix} \quad (4.9)$$

to convert the complex numbers to real values. This is followed by a real Givens rotation (4.7) to zero out the second component. However, in order to avoid four complex rotations, the complex conjugate of (4.9) is applied to the left side of the real Givens rotation, giving the complex Givens rotation in (4.8).

In the expanded form (4.8) may be written as

$$\begin{bmatrix} e^{i\theta_a} & 0 \\ 0 & e^{i\theta_b} \end{bmatrix} \begin{bmatrix} \cos \theta_1 & \sin \theta_1 \\ -\sin \theta_1 & \cos \theta_1 \end{bmatrix} \begin{bmatrix} e^{-i\theta_a} & 0 \\ 0 & e^{-i\theta_b} \end{bmatrix} = \begin{bmatrix} \cos \theta_1 & \sin \theta_1 (e^{i\theta_2}) \\ -\sin \theta_1 (e^{-i\theta_2}) & \cos \theta_1 \end{bmatrix}.$$

The angles θ_1 and θ_2 can be determined from the input vector as follows

$$A = \sqrt{a_r^2 + a_i^2}, \quad \theta_a = \tan^{-1} \left(\frac{a_i}{a_r} \right), \quad (4.10)$$

$$B = \sqrt{b_r^2 + b_i^2}, \quad \theta_b = \tan^{-1} \left(\frac{b_i}{b_r} \right).$$

Then from the above angles and radii,

$$\theta_1 = \tan^{-1} \left(\frac{B}{A} \right) \quad \text{and} \quad \theta_2 = \theta_a - \theta_b. \quad (4.11)$$

4.3 QRD of a Complex 2×2 Matrix

The QR-Decomposition of a real $m \times n$ matrix (2.15) was discussed in § 2.5. The QR-decomposition (QRD) of a $m \times n$ matrix $M \in C^{m \times n}$, is given by

$$M = Q \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad (4.12)$$

where the matrix $Q \in C^{m \times m}$ is unitary and the matrix $R \in C^{n \times n}$ is upper triangular.

Let

$$\mathcal{M} \triangleq \begin{bmatrix} a_r + i a_i & b_r + i b_i \\ c_r + i c_i & d_r + i d_i \end{bmatrix} = \begin{bmatrix} A e^{i\theta_a} & B e^{i\theta_b} \\ C e^{i\theta_c} & D e^{i\theta_d} \end{bmatrix}, \quad (4.13)$$

be a complex 2×2 matrix. Using the complex Givens rotation [17] as described in § 4.2, the QR decomposition of a complex 2×2 matrix (4.13) can be computed as

$$\begin{bmatrix} c_\alpha & s_\alpha e^{i\theta_\beta} \\ -s_\alpha e^{-i\theta_\beta} & c_\alpha \end{bmatrix} \begin{bmatrix} A e^{i\theta_a} & B e^{i\theta_b} \\ C e^{i\theta_c} & D e^{i\theta_d} \end{bmatrix} = \begin{bmatrix} W e^{i\theta_\omega} & X e^{i\theta_x} \\ 0 & Z e^{i\theta_z} \end{bmatrix}, \quad (4.14)$$

where

$$\theta_\beta = \theta_\alpha - \theta_c \quad \text{and} \quad \tan \alpha = \frac{C}{A}.$$

Deprettere and van der Veen [79] define a three angle complex rotation

$$R(\theta, \phi_1, \phi_2) \triangleq \begin{bmatrix} e^{i\phi_1} & 0 \\ 0 & e^{i\phi_2} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \quad (4.15)$$

where $c = \cos \theta$ and $s = \sin \theta$. The QR decomposition of a complex 2×2 matrix \mathcal{M} (4.13) can also be defined in terms of (4.15) as

$$\begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} R(\theta, \phi_1, \phi_2) = \begin{bmatrix} We^{i\theta_w} & Xe^{i\theta_x} \\ 0 & Ze^{i\theta_z} \end{bmatrix}, \quad (4.16)$$

from which

$$\tan \theta = \frac{s}{c} = \frac{e^{i\phi_1} Ce^{i\theta_c}}{e^{i\phi_2} De^{i\theta_d}}. \quad (4.17)$$

To ensure that $\tan \theta$ is real, ϕ_1 and ϕ_2 are chosen to make the numerator with respect to the denominator in (4.17), real. This requires that

$$\phi_1 = -\theta_c, \quad \phi_2 = -\theta_d, \quad (4.18)$$

and

$$\theta = \tan^{-1} \left(\frac{C}{D} \right) \quad (4.19)$$

in (4.16). Given that (4.18) and (4.19) hold, (4.16) can be rewritten as

$$\begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} R(\theta, \phi_1, \phi_2) = \begin{bmatrix} We^{i\theta_w} & Xe^{i\theta_x} \\ 0 & Z \end{bmatrix}, \quad (4.20)$$

where $Z = \sqrt{C^2 + D^2}$.

Lotkin [56] and Greenstadt [38] have also suggested transformations for the triangularization of a 2×2 matrix to be used in the Jacobi [32] method for the Eigenvalue Decomposition of arbitrary matrices.

4.4 SVD of a Complex 2×2 Matrix

The complex SVD can be approached by using the SVD-Jacobi algorithm [32, 49] discussed in § 2.3. As a basic step in the complex SVD-Jacobi algorithm, a 2×2 matrix, possibly with complex data elements, is transformed into a real diagonal matrix.

Several two-sided unitary transformations have been suggested in the literature [32, 49] to diagonalize a complex 2×2 matrix. The Forsythe and Henrici [32] two-sided unitary transformation to diagonalize a complex 2×2 matrix is

$$\begin{bmatrix} c_\phi e^{i\theta_\alpha} & -s_\phi e^{i\theta_\beta} \\ s_\phi e^{i\theta_\gamma} & c_\phi e^{i\theta_\delta} \end{bmatrix} \begin{bmatrix} A e^{i\theta_a} & B e^{i\theta_b} \\ C e^{i\theta_c} & D e^{i\theta_d} \end{bmatrix} \begin{bmatrix} c_\psi e^{i\theta_\epsilon} & s_\psi e^{i\theta_\eta} \\ -s_\psi e^{i\theta_\zeta} & c_\psi e^{i\theta_\omega} \end{bmatrix} = \begin{bmatrix} W e^{i\theta_w} & 0 \\ 0 & Z e^{i\theta_z} \end{bmatrix}, \quad (4.21)$$

where

$$\tan(\theta_\alpha - \theta_\beta) = -\frac{AC \sin(\theta_a - \theta_c) + BD \sin(\theta_b - \theta_d)}{AC \cos(\theta_a - \theta_c) + BD \cos(\theta_b - \theta_d)}, \quad (4.22)$$

$$\tan(\theta_\eta - \theta_\omega) = -\frac{AB \sin(\theta_a - \theta_b) + CD \sin(\theta_c - \theta_d)}{AB \cos(\theta_a - \theta_b) + CD \cos(\theta_c - \theta_d)},$$

$$\tan(\theta_\phi - \theta_\psi) = -\frac{B e^{i(\theta_a + \theta_\omega + \theta_b)} + C e^{i(\theta_\beta + \theta_\eta + \theta_c)}}{D e^{i(\theta_\beta + \theta_\omega + \theta_d)} - A e^{i(\theta_a + \theta_\eta + \theta_a)}},$$

and

$$\tan(\theta_\phi + \theta_\psi) = -\frac{B e^{i(\theta_a + \theta_\omega + \theta_b)} - C e^{i(\theta_\beta + \theta_\eta + \theta_c)}}{D e^{i(\theta_\beta + \theta_\omega + \theta_d)} + A e^{i(\theta_a + \theta_\eta + \theta_a)}}.$$

When the matrix \mathcal{M} is real, condition (4.22) simplifies to

$$\theta_\alpha = \theta_\beta = \theta_\gamma = \theta_\delta = 0, \quad (4.23)$$

and

$$\tan(\theta_\phi - \theta_\psi) = -\left(\frac{B+C}{D-A}\right), \quad (4.24)$$

$$\tan(\theta_\phi + \theta_\psi) = -\left(\frac{B-C}{D+A}\right).$$

This result is identical to the direct two-angle, two-sided rotation, diagonalization scheme (2.6,2.9), described in § 2.4 for the SVD of a real 2×2 matrix.

Another scheme for the diagonalization of an arbitrary 2×2 matrix is due to Kogbetliantz [49]. He defines a two-sided unitary transformation to compute the SVD of a complex 2×2 matrix as

$$\begin{bmatrix} \gamma^{\frac{1}{2}}c_{\zeta} & \gamma^{\frac{1}{2}}s_{\zeta} \\ -\gamma^{\frac{1}{2}}s_{\zeta'} & \gamma^{\frac{1}{2}}c_{\zeta'} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} \gamma^{\frac{1}{2}}c_z & -\gamma^{\frac{1}{2}}s_{z'} \\ \gamma^{\frac{1}{2}}s_z & \gamma^{\frac{1}{2}}c_{z'} \end{bmatrix} = \begin{bmatrix} We^{i\theta_w} & 0 \\ 0 & Ze^{i\theta_z} \end{bmatrix}, \quad (4.25)$$

where

$$\begin{aligned} z &= \psi + i\phi, \quad z' = \psi - i\phi, \\ \zeta &= \xi + i\eta, \quad \zeta' = \xi - i\eta, \end{aligned}$$

and

$$\gamma = \operatorname{sech} 2\phi.$$

The left and right transformation matrices in (4.25) are unitary since

$$\gamma(|c|^2 + |s|^2) = 1.$$

There are four real equations in the four real unknowns ψ , ϕ , ξ , and η for (4.25) to hold.

Deprettere and van der Veen [79] considered input matrices with a specialized structure as

$$\mathcal{S} = \begin{bmatrix} A & Be^{i\theta_b} \\ 0 & D \end{bmatrix} \quad (4.26)$$

for a CORDIC SVD array based on the SVD-Jacobi method. The diagonalization of \mathcal{S} is accomplished in two steps. In the first step, a transition to an all real element data matrix is made. Formally,

$$\begin{bmatrix} e^{-i\theta_{\phi}} & 0 \\ 0 & e^{i\theta_{\phi}} \end{bmatrix} \begin{bmatrix} A & Be^{i\theta_b} \\ 0 & D \end{bmatrix} \begin{bmatrix} e^{i\theta_{\phi}} & 0 \\ 0 & e^{-i\theta_{\phi}} \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & D \end{bmatrix}, \quad (4.27)$$

from which $\theta_{\phi} = \theta_b/2$. Next, the real 2×2 matrix diagonalization scheme discussed in § 2.4 is used to complete the SVD. Precisely, the transformation

$$R(\theta_r)^T \begin{bmatrix} A & B \\ 0 & D \end{bmatrix} R(\theta_l) = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$

is used, where $R(\theta)$ is given by (2.7), and

$$\begin{aligned}\tan(\theta_r - \theta_l) &= \left(\frac{B}{D - A} \right), \\ \tan(\theta_r + \theta_l) &= \left(\frac{B}{D + A} \right).\end{aligned}\tag{4.28}$$

A generalization of the complex CORDIC SVD scheme in van der Veen and Deprettere [79] is presented in [12]. An input matrix \mathcal{M} of the form (4.13) is assumed. Several transformations are performed to diagonalize \mathcal{M} .

First a complex Givens rotation (4.8) is applied to introduce a zero into \mathcal{M} . This transforms the matrix as

$$\begin{bmatrix} c_1 & s_1(c_2 + is_2) \\ -s_1(c_2 - is_2) & c_1 \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} = \begin{bmatrix} We^{i\theta_w} & Xe^{i\theta_x} \\ 0 & Ze^{i\theta_z} \end{bmatrix}.\tag{4.29}$$

A unitary transformation

$$U = \begin{bmatrix} e^{i\theta_\gamma} & 0 \\ 0 & e^{i\theta_\delta} \end{bmatrix},\tag{4.30}$$

where $\theta_\gamma = -\theta_w$ and $\theta_\delta = -\theta_z$, is then applied to make the diagonal elements real.

This transforms the result of (4.29) as

$$\begin{bmatrix} e^{i\theta_\gamma} & 0 \\ 0 & e^{i\theta_\delta} \end{bmatrix} \begin{bmatrix} We^{i\theta_w} & Xe^{i\theta_x} \\ 0 & Ze^{i\theta_z} \end{bmatrix} = \begin{bmatrix} W & Xe^{i(\theta_x + \theta_\gamma)} \\ 0 & Z \end{bmatrix} = \begin{bmatrix} W & X'e^{i\theta_{x'}} \\ 0 & Z \end{bmatrix}.\tag{4.31}$$

Next, a two-sided unitary transformation

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta_\epsilon} \end{bmatrix} \begin{bmatrix} W & X'e^{i\theta_{x'}} \\ 0 & Z \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta_\zeta} \end{bmatrix} = \begin{bmatrix} W & X' \\ 0 & Z \end{bmatrix},\tag{4.32}$$

where $\theta_\epsilon = \theta_{x'}$ and $\theta_\zeta = -\theta_{x'}$, is applied to make the 2×2 matrix real. Finally, a real 2×2 SVD is performed using (2.13) with

$$\begin{aligned}\tan(\theta_r - \theta_l) &= \left(\frac{X'}{Z - W} \right), \\ \tan(\theta_r + \theta_l) &= \left(\frac{X'}{Z + W} \right).\end{aligned}\tag{4.33}$$

4.5 Two-sided Unitary Transformations

The various methods proposed for the SVD of a complex 2×2 matrix in § 4.4 have shortcomings. They are either, too cumbersome to implement in special-purpose VLSI using CORDIC or traditional arithmetic units like the Forsythe-Henrici scheme (4.21) and the scheme due to Kogbetliantz (4.25), or they assume a specialized matrix structure as in the Deprettere-van der Veen scheme (4.27,4.28). The scheme due to Cavallaro-Elster (4.29-4.33), though implementable in CORDIC, does not efficiently adapt to systolic computation.

Two-sided rotations are used in the diagonalization of a real 2×2 matrix in the Brent-Luk-Van Loan systolic array [6]. In the development of a systolic scheme to diagonalize a complex matrix, it is important to express the methods of the previous sections as two-sided unitary rotations/transformations. This has been the primary motivation for the transformations developed in this section.

Additionally, the two-sided transformations need to be structured in a manner that is easily amenable to implementation in hardware, preferably using the CORDIC processor architectures discussed in § 3.2 and § 3.3. The discussion on implementation using CORDIC is postponed until § 5.2.

Any 2×2 unitary matrix may be expressed as

$$\begin{bmatrix} c_\phi e^{i\theta_\alpha} & s_\phi e^{i\theta_\beta} \\ -s_\phi e^{i\theta_\gamma} & c_\phi e^{i\theta_\delta} \end{bmatrix}, \quad (4.34)$$

where $\theta_\alpha, \theta_\beta, \theta_\gamma, \theta_\delta$ and θ_ϕ are real numbers with

$$\theta_\alpha - \theta_\beta - \theta_\gamma + \theta_\delta \equiv 0 \pmod{2\pi}. \quad (4.35)$$

The transformations described below, each, assume an input data \mathcal{M} matrix of the form (4.13). Also, from (4.34) and (4.35) the equations (4.36-4.40) are, clearly, unitary transformations. The transformations described below are each identified

by an alphabetic character to preserve ease of notation and at the same time hint at the nature of the transformation. The transformations defined below are by no means special or unique; they are presented mainly to motivate the derivation of more complicated transformations.

The \mathcal{I} Transformation

An \mathcal{I} transformation, is a simple two-sided unitary transformation that interchanges the arguments of the elements of \mathcal{M} , along a diagonal. It is expressed as

$$\begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{-i\theta_\alpha} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{-i\theta_\alpha} \end{bmatrix} = \begin{bmatrix} Ae^{i\theta_d} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_a} \end{bmatrix} \quad (4.36)$$

or

$$\begin{bmatrix} e^{i\theta_\beta} & 0 \\ 0 & e^{-i\theta_\beta} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{-i\theta_\beta} & 0 \\ 0 & e^{i\theta_\beta} \end{bmatrix} = \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_c} \\ Ce^{i\theta_b} & De^{i\theta_d} \end{bmatrix},$$

where

$$\theta_\alpha = \frac{(\theta_d - \theta_a)}{2} \text{ and } \theta_\beta = \frac{(\theta_c - \theta_b)}{2}.$$

The \mathcal{R} and \mathcal{C} Transformations

Given an arbitrary complex 2×2 matrix, a two-sided unitary transformation can be defined to convert any two elements of \mathcal{M} to real values. If the elements belong to the same row then, the transformation \mathcal{R} is defined as

$$\begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{i\theta_\alpha} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\beta} & 0 \\ 0 & e^{-i\theta_\beta} \end{bmatrix} = \begin{bmatrix} A & B \\ Ce^{i\theta_y} & De^{i\theta_x} \end{bmatrix} \quad (4.37)$$

or

$$\begin{bmatrix} e^{i\theta_\gamma} & 0 \\ 0 & e^{i\theta_\gamma} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\delta} & 0 \\ 0 & e^{-i\theta_\delta} \end{bmatrix} = \begin{bmatrix} Ae^{i\theta_w} & Be^{i\theta_x} \\ C & D \end{bmatrix},$$

where

$$\theta_\alpha = \frac{-(\theta_b + \theta_a)}{2}, \theta_\beta = \frac{(\theta_b - \theta_a)}{2}, \theta_\gamma = \frac{-(\theta_d + \theta_c)}{2} \text{ and } \theta_\delta = \frac{(\theta_d - \theta_c)}{2}.$$

If the elements belong to the same column then, the transformation \mathcal{C} is defined

as

$$\begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{-i\theta_\alpha} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\beta} & 0 \\ 0 & e^{i\theta_\beta} \end{bmatrix} = \begin{bmatrix} A & Be^{i\theta_x} \\ C & De^{i\theta_x} \end{bmatrix} \quad (4.38)$$

or

$$\begin{bmatrix} e^{i\theta_\gamma} & 0 \\ 0 & e^{-i\theta_\gamma} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\delta} & 0 \\ 0 & e^{i\theta_\delta} \end{bmatrix} = \begin{bmatrix} Ae^{i\theta_w} & B \\ Ce^{i\theta_y} & D \end{bmatrix},$$

where

$$\theta_\alpha = \frac{(\theta_c - \theta_a)}{2}, \quad \theta_\beta = \frac{-(\theta_c + \theta_a)}{2}, \quad \theta_\gamma = \frac{(\theta_d - \theta_b)}{2} \text{ and } \theta_\delta = \frac{-(\theta_d + \theta_b)}{2}.$$

The \mathcal{D} Transformation

If the elements are on the diagonal (main or off) then, the transformation \mathcal{D} is defined

as

$$\begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{i\theta_\alpha} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\beta} & 0 \\ 0 & e^{-i\theta_\beta} \end{bmatrix} = \begin{bmatrix} A & Be^{i\theta_x} \\ Ce^{i\theta_y} & D \end{bmatrix} \quad (4.39)$$

or

$$\begin{bmatrix} e^{i\theta_\gamma} & 0 \\ 0 & e^{i\theta_\gamma} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\delta} & 0 \\ 0 & e^{-i\theta_\delta} \end{bmatrix} = \begin{bmatrix} Ae^{i\theta_w} & B \\ C & De^{i\theta_x} \end{bmatrix},$$

where

$$\theta_\alpha = \frac{-(\theta_d + \theta_a)}{2}, \quad \theta_\beta = \frac{(\theta_d - \theta_a)}{2}, \quad \theta_\gamma = \frac{-(\theta_c + \theta_b)}{2} \text{ and } \theta_\delta = \frac{(\theta_d - \theta_b)}{2}.$$

Unlike the transformations \mathcal{R} (4.37) and \mathcal{C} (4.38), the angles θ_α and θ_γ in \mathcal{D} (4.39) can be interchanged with θ_β and θ_δ respectively, to yield the same effect ³.

The \mathcal{Q} Transformation

Any combination of the \mathcal{I} , \mathcal{R} , \mathcal{C} and \mathcal{D} transformations can be used with a real two-sided rotation to yield a useful two-sided unitary transformation. The transformation

³However, the subsidiary angles θ_w and θ_x , or θ_x and θ_y will be different.

\mathcal{Q} is defined as

$$\begin{bmatrix} c_\phi e^{i\theta_\alpha} & -s_\phi e^{i\theta_\beta} \\ s_\phi e^{i\theta_\alpha} & c_\phi e^{i\theta_\beta} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} c_\psi e^{i\theta_\gamma} & s_\psi e^{i\theta_\gamma} \\ -s_\psi e^{i\theta_\delta} & c_\psi e^{i\theta_\delta} \end{bmatrix}, \quad (4.40)$$

where θ_α , θ_β , θ_γ , and θ_δ are chosen according to the combination of (4.36), (4.37), (4.38) and (4.39) desired.

If either θ_ϕ or θ_ψ is zero, and the other angle is chosen as in (4.7), a zero can be introduced into the 2×2 matrix at a desired position by choosing an appropriate combination of the sub-transformations \mathcal{I} , \mathcal{R} , \mathcal{C} and \mathcal{D} to transform \mathcal{M} . Also, if the sub-transformations chosen render the matrix real, then angles θ_ϕ and θ_ψ may be chosen as in (4.24) to diagonalize \mathcal{M} .

The transformations \mathcal{I} , \mathcal{C} , \mathcal{D} , \mathcal{R} and \mathcal{Q} can be implemented easily in CORDIC as will be explained in § 5.2. The \mathcal{Q} transformation is used as a basis for the two-step diagonalization scheme presented in § 4.6.

4.6 Two-step SVD of a Complex 2×2 Matrix

The transformations developed in § 4.5 can be used to derive a diagonalization procedure for an arbitrary complex 2×2 matrix \mathcal{M} as defined in (4.13). The conditions for (4.21) as expressed by (4.22) were shown to be necessary and sufficient for the diagonalization of an arbitrary complex 2×2 matrix \mathcal{M} . It is clear from the nature of the transformations developed in § 4.5, that an arbitrary complex 2×2 matrix cannot always be diagonalized by just one \mathcal{Q} transformation.

However, two \mathcal{Q} transformations are sufficient to compute the SVD. The first \mathcal{Q} transformation essentially performs a QR decomposition of \mathcal{M} . The second completes the diagonalization. To illustrate the steps in the diagonalization, each \mathcal{Q} transformation is presented as a combination of sub-transformations used.

The first sub-transformation is an \mathcal{R} transformation which renders the bottom row of \mathcal{M} real. It is followed by a real Givens rotation which zeros the lower-left (2,1) element. This completes the first \mathcal{Q} transformation which is defined as

$$\begin{bmatrix} c_\phi e^{i\theta_\alpha} & -s_\phi e^{i\theta_\beta} \\ s_\phi e^{i\theta_\alpha} & c_\phi e^{i\theta_\beta} \end{bmatrix} \begin{bmatrix} A e^{i\theta_a} & B e^{i\theta_b} \\ C e^{i\theta_c} & D e^{i\theta_d} \end{bmatrix} \begin{bmatrix} c_\psi e^{i\theta_\gamma} & s_\psi e^{i\theta_\gamma} \\ -s_\psi e^{i\theta_\delta} & c_\psi e^{i\theta_\delta} \end{bmatrix} = \begin{bmatrix} W e^{i\theta_w} & X e^{i\theta_x} \\ 0 & Z \end{bmatrix}, \quad (4.41)$$

where

$$\begin{aligned} \theta_\alpha &= \theta_\beta = \frac{-(\theta_d + \theta_c)}{2}, \\ \theta_\gamma &= -\theta_\delta = \frac{(\theta_d - \theta_c)}{2}, \end{aligned}$$

and

$$\theta_\phi = 0, \quad \theta_\psi = \tan^{-1}\left(\frac{C}{D}\right).$$

Next, a \mathcal{D} and an \mathcal{I} transformation are combined with a two-sided rotation (2.6,2.9) to generate the \mathcal{Q} transformation for the second step. The \mathcal{D} transformation converts the main diagonal elements to real values and the \mathcal{I} transformation takes advantage of the fact that the lower-left (2,1) element is zero, to convert the upper-right (1,2) element to a real value. After the \mathcal{D} and \mathcal{I} transformations are applied, the 2×2 matrix is real. Equation (2.6) with (2.9) is then used to diagonalize the real matrix. Thus, the second \mathcal{Q} transformation can be written as

$$\begin{bmatrix} c_\lambda e^{i\theta_\xi} & -s_\lambda e^{i\theta_\eta} \\ s_\lambda e^{i\theta_\xi} & c_\lambda e^{i\theta_\eta} \end{bmatrix} \begin{bmatrix} W e^{i\theta_w} & X e^{i\theta_x} \\ 0 & Z \end{bmatrix} \begin{bmatrix} c_\rho e^{i\theta_\zeta} & s_\rho e^{i\theta_\zeta} \\ -s_\rho e^{i\theta_\omega} & c_\rho e^{i\theta_\omega} \end{bmatrix} = \begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix}. \quad (4.42)$$

Unlike the first \mathcal{Q} transformation⁴, there are two possible sets of values for the unitary transformation angles. This is due to the fact that the left and right unitary angles in a \mathcal{D} transformation are interchangeable. The choice is between

$$\theta_\xi = -\left(\frac{\theta_x + \theta_w}{2}\right), \quad \theta_\eta = \left(\frac{\theta_x - \theta_w}{2}\right), \quad (4.43)$$

⁴The QRD can be accomplished with a \mathcal{Q} transformation which uses a \mathcal{C} sub-transformation along with a suitable choice of θ_ϕ and θ_ψ . It is immaterial how the first \mathcal{Q} transformation is structured as far as CORDIC implementation is concerned.

$$\theta_\zeta = \left(\frac{\theta_x - \theta_w}{2} \right), \quad \theta_\omega = \left(\frac{\theta_w - \theta_x}{2} \right),$$

and

$$\begin{aligned} \theta_\xi &= -\left(\frac{\theta_x}{2} \right), & \theta_\eta &= \left(\frac{\theta_x}{2} \right), \\ \theta_\zeta &= \left(\frac{\theta_x}{2} \right) - \theta_w, & \theta_\omega &= -\left(\frac{\theta_x}{2} \right). \end{aligned} \tag{4.44}$$

However, the rotation angles for the second \mathcal{Q} transformation are given by

$$\tan(\theta_\lambda - \theta_\rho) = -\left(\frac{X}{Z - W} \right),$$

and

$$\tan(\theta_\lambda + \theta_\rho) = -\left(\frac{X}{Z + W} \right),$$

no matter which of (4.43) or (4.44) is chosen for the unitary angles.

The reasons for the choice of (4.43) over (4.44) relates to the implementation of the \mathcal{Q} transformation using CORDIC (§ 5.2). However, it is easy to see that

$$|\theta_\xi| \leq \frac{\pi}{2}, \quad |\theta_\eta| \leq \frac{\pi}{2}, \tag{4.45}$$

$$|\theta_\zeta| \leq \frac{\pi}{2}, \quad \text{and} \quad |\theta_\omega| \leq \frac{\pi}{2},$$

if the arguments θ_w and θ_x are computed using (4.6) from the corresponding orthogonal representations; a fact of considerable importance as far as CORDIC implementation of the \mathcal{Q} transformation is concerned, since the convergence limits of the CORDIC algorithms (in the circular mode) extend only to the right half-plane.

In the next chapter, the issue of implementing the \mathcal{Q} transformation using CORDIC is examined. Some modifications to the basic CORDIC scheme are necessitated by the limitation in the convergence range. An architecture for the complex 2×2 processor is also proposed.

Chapter 5

CORDIC for Complex SVD

In Chapter 4, a two-step diagonalization scheme for the SVD of an arbitrary matrix was presented. The scheme was tailored towards easy implementation using the CORDIC primitives of vector rotations and inverse tangent calculations, introduced in § 3.1. However, a few modifications to the basic CORDIC procedure are necessary to handle complex data.

5.1 Modified CORDIC for Complex Arithmetic

Use of CORDIC algorithms for complex arithmetic has been discussed by Hitotumatu [44]. He described CORDIC based procedures for the multiplication and division of two arbitrary complex numbers and a procedure to compute the square root of a complex number.

From (3.12), it can be seen that the convergence region of the CORDIC transformations in the circular mode, covers the closed right half plane. Therefore, it is convenient to restrict the argument of a complex number to the interval $[-\frac{\pi}{2}, +\frac{\pi}{2}]$, and allow negative moduli (4.6).

Restating (4.6), a complex number, $z = z_r + i z_i$ may be written as $z = R_z e^{i\theta_z}$, where

$$R_z = \text{sign}(z_r) \sqrt{z_r^2 + z_i^2}, \quad \text{and} \quad \theta_z = \tan^{-1} \left(\frac{z_i}{z_r} \right),$$

with $-\pi/2 \leq \theta_z \leq \pi/2$.

Let $a + ib$ and $c + id$, be two arbitrary complex numbers and the product of $a + ib$ and $c + id$ be $p + iq$. Formally,

$$\begin{aligned} p + iq &= (a + ib) \times (c + id) \\ &= (ac - bd) + i(ad + bc) \end{aligned} \tag{5.1}$$

The CORDIC based algorithm for computing the product of two complex numbers, due to Hitotumatu [44], is shown in Table 5.1. The CORDIC multiplication scheme requires the use of traditional multiplication hardware or CORDIC in the linear mode ($m = 0$). It is less efficient than the direct method that involves the formation of four products of real numbers (5.1), as it requires 3 product calculations and 2 CORDIC operations (including scale factor correction).

From the discussion on CORDIC in § 3.1, it is clear that CORDIC operations require $O(n)$ additions, where n is the word-length or bit-precision of the hardware, as do multiplications. The area/time complexity of CORDIC operations and multiplications is thus, similar when multiplications are computed through a series of shift and add operations; although this may not always be the case.

Architectures based on CORDIC for operations on complex data have also been proposed in the literature. A CORDIC architecture for the application of the complex Givens rotation (4.8), is described in [12]. Figure 5.1 shows the generation of the angles for the complex Givens rotation using CORDIC arctan modules (y -reduction in the circular mode), and Figure 5.2 shows the application of complex Givens rotation using CORDIC vector rotation modules (z -reduction in the circular mode).

Another CORDIC architecture for complex arithmetic is due to van der Veen and Deprettere [79]. The architecture as shown in Figure 5.3 applies the 3-angle complex rotation described by (4.15). It is composed of three vector rotation modules. In both architectures, operations are performed on complex data with the number being represented by a real vector composed of the real and imaginary parts.

Transformation to Polar Coordinates.

Take initial values as

$$\begin{aligned}x_0 &= |c|, \\y_0 &= \pm d \text{ (+ if } c \geq 0, \text{ and } - \text{ if } c < 0), \\z_0 &= 0.\end{aligned}$$

Perform CORDIC Y – Reduction (circular mode). Then,

$$\begin{aligned}x_n &= K_1 \sqrt{c^2 + d^2}, \\y_n &= 0, \\z_n &= \theta_{cd} = \tan^{-1} \left(\frac{d}{c} \right) \text{ where } \left(-\frac{\pi}{2} \leq \theta_{cd} \leq \frac{\pi}{2} \right).\end{aligned}$$

Replace

$$x_n \text{ by } -x_n \text{ if } p < 0.$$

Do anticipatory scale factor correction

$$R_p = K_1^{-2} x_n.$$

Rotation

Initialize

$$\begin{aligned}x_0 &= a \times R_p, \\y_0 &= b \times R_p, \\z_0 &= \theta_{cd}.\end{aligned}$$

Perform CORDIC Z – Reduction (circular mode). Then,

$$\begin{cases} x_n = K_1 K_1^{-2} K_1 \sqrt{c^2 + d^2} (a \cos \theta_{cd} - b \sin \theta_{cd}) = ac - bd, \\ y_n = K_1 K_1^{-2} K_1 \sqrt{c^2 + d^2} (a \sin \theta_{cd} + b \cos \theta_{cd}) = ad + bc. \end{cases}$$

Table 5.1: Complex Multiplication with CORDIC

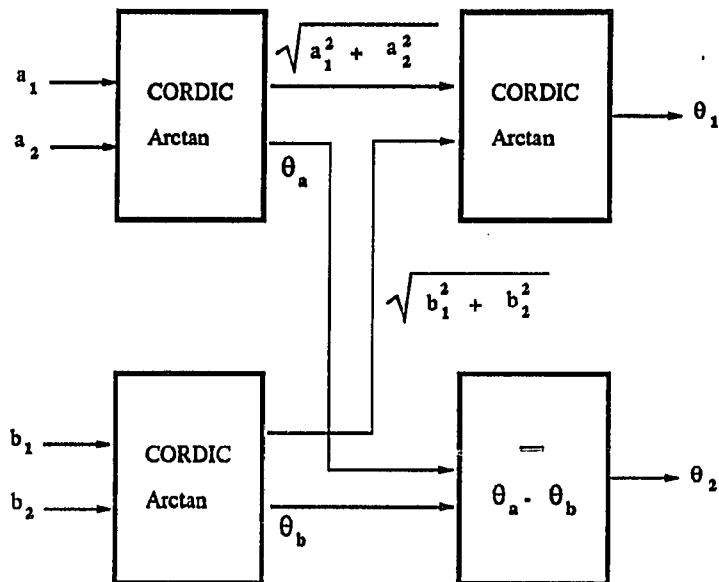


Figure 5.1: CORDIC Complex Givens Rotation Angle Calculation

The complex CORDIC architectures in [12] and [79] do not explicitly address the issue of scale factor correction. From earlier discussion in § 3.1, it is apparent that this can be crucial to the performance and accuracy of the architecture, especially if single scale factor correction is necessitated.

5.2 CORDIC for the Q Transformation

In § 4.5, the Q transformation (4.40) was presented. A two-step diagonalization scheme for the SVD of a complex 2×2 matrix was then derived, with each step as a Q transformation. The structuring of the Q transformation is designed for efficient implementation in CORDIC.

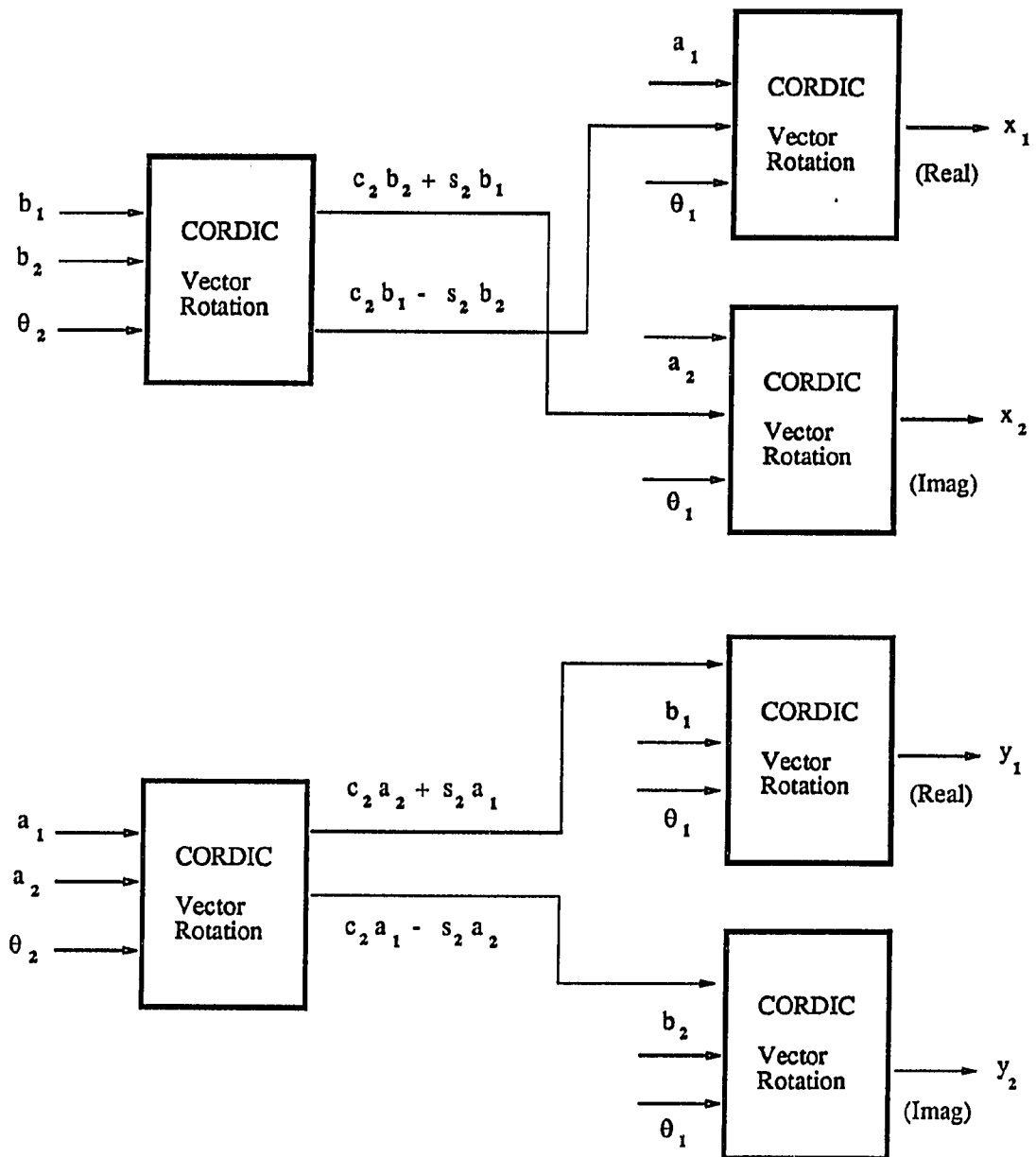


Figure 5.2: CORDIC Complex Givens Rotation Application

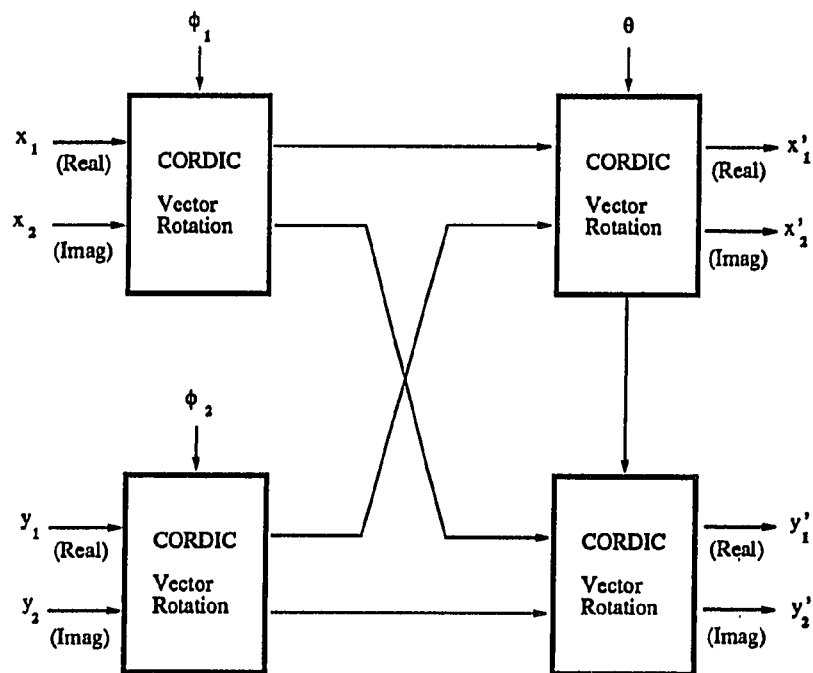


Figure 5.3: Deprettere and van der Veen Complex Rotation in CORDIC

The Q transformation was motivated primarily by two considerations. Firstly, the multiplication of two arbitrary complex numbers, as demonstrated by Hitotumatu's algorithm (Table 5.1) is more efficiently done using traditional arithmetic units.

If the multiplication algorithm in Table 5.1 is examined, the need for multiplication hardware arises at two points, *viz.*, first at the scale factor correction step, and secondly, at the initialization of data for the rotation step. If the two steps could somehow be circumvented, through suitably structuring a transformation, then Hitotumatu's algorithm can be adapted to efficiently compute complex vector rotations.

The scale factor correction may be handled using any of the schemes discussed in § 3.1. And, the multiplication required at the initialization of data for the rotation

step, could be avoided if the modulus of one of the two complex numbers involved, is known to be unity. However, for numerical reasons the computation of the modulus is to be avoided altogether.

Secondly, the CORDIC primitives of use are vector rotations and inverse tangent calculations. From § 3.1, we know that vector rotations are not complete until the results are corrected for the scale factor introduced by the CORDIC iterations. In the discussion on scale factor considerations in § 3.1, it was mentioned that a two-sided scale factor correction [14] is preferable over a single scale factor correction, both for the systematic approach and the performance advantage of the scheme.

Therefore, any transformation used must be amenable to two-sided scale factor corrections; i.e. only an even number of rotational transformations should be used at each step. If at any step in the diagonalization procedure a one-sided rotation is employed, then a more costly single scale factor correction is necessitated.

The \mathcal{Q} transformation

$$\begin{bmatrix} c_\phi e^{i\theta_\alpha} & -s_\phi e^{i\theta_\beta} \\ s_\phi e^{i\theta_\alpha} & c_\phi e^{i\theta_\beta} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} c_\psi e^{i\theta_\gamma} & s_\psi e^{i\theta_\gamma} \\ -s_\psi e^{i\theta_\delta} & c_\psi e^{i\theta_\delta} \end{bmatrix}, \quad (5.2)$$

is recalled to present a scheme for the implementation using CORDIC. The \mathcal{Q} transformation may be rewritten as

$$\begin{bmatrix} c_\phi & -s_\phi \\ s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{i\theta_\beta} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\gamma} & 0 \\ 0 & e^{i\theta_\delta} \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi \\ -s_\psi & c_\psi \end{bmatrix}, \quad (5.3)$$

to hint at a procedure for the use of CORDIC in the application of the \mathcal{Q} transformation.

As indicated by (5.3), the \mathcal{Q} transformation can be performed in two steps. First, the inner two-sided unitary transformation is completed, followed by the two-sided rotational transformation. The inner unitary transformation, essentially affects only the arguments of the complex data elements. Precisely,

$$\begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{i\theta_\beta} \end{bmatrix} \begin{bmatrix} Ae^{i\theta_a} & Be^{i\theta_b} \\ Ce^{i\theta_c} & De^{i\theta_d} \end{bmatrix} \begin{bmatrix} e^{i\theta_\gamma} & 0 \\ 0 & e^{i\theta_\delta} \end{bmatrix} = \begin{bmatrix} Ae^{i\theta'_a} & Be^{i\theta'_b} \\ Ce^{i\theta'_c} & De^{i\theta'_d} \end{bmatrix} \quad (5.4)$$

where

$$\theta'_a = \theta_a + \theta_\alpha + \theta_\gamma, \quad \theta'_b = \theta_b + \theta_\alpha + \theta_\delta,$$

and

$$\theta'_c = \theta_c + \theta_\beta + \theta_\gamma, \quad \theta'_d = \theta_d + \theta_\beta + \theta_\delta.$$

We assume that the input matrix has the complex data elements represented in orthogonal coordinates. Since the inner transformation affects only the arguments of the complex data, it is convenient to compute the polar coordinate representations of the data elements. The unitary transformation angles can then be used to modify the arguments appropriately. A transformation back to the orthogonal coordinate system completes the unitary transformation.

Let

$$m = m_r + im_i = Me^{\theta_m}$$

be representative of a data element in the 2×2 complex matrix and θ_u be the shift in the argument necessitated by the unitary transformation⁵. The exact procedure in CORDIC to implement the inner unitary transformation, is given in Table 5.2.

Once the inner rotation of the \mathcal{Q} transformation has been completed, the outer two-sided rotation needs to be computed. From the arithmetic of complex numbers, it is easy to observe that the two-sided rotation may be applied independently to the real and imaginary parts of the data elements. Thus, the outer transformation may be computed as

$$\begin{aligned} & \begin{bmatrix} c_\phi & -s_\phi \\ s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} a'_r + ia'_i & b'_r + ib'_i \\ c'_r + ic'_i & d'_r + id'_i \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi \\ -s_\psi & c_\psi \end{bmatrix} \\ &= \begin{bmatrix} c_\phi & -s_\phi \\ s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} a'_r & b'_r \\ c'_r & d'_r \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi \\ -s_\psi & c_\psi \end{bmatrix} + i \left(\begin{bmatrix} c_\phi & -s_\phi \\ s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} a'_i & b'_i \\ c'_i & d'_i \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi \\ -s_\psi & c_\psi \end{bmatrix} \right), \end{aligned}$$

⁵The quantities M and θ_m are computed as per the modified polar coordinates (4.6)

Transformation to Polar Coordinates.

Take initial values as

$$\begin{aligned} x_0 &= |m_r|, \\ y_0 &= \pm m_i \text{ (+ if } m_r \geq 0, \text{ and - if } m_r < 0), \\ z_0 &= 0. \end{aligned}$$

Perform CORDIC Y – Reduction (circular mode). Then,

$$\begin{aligned} x_n &= K_1 \sqrt{m_r^2 + m_i^2}, \\ y_n &= 0, \\ z_n &= \theta_m = \tan^{-1} \left(\frac{m_i}{m_r} \right) \text{ where } \left(-\frac{\pi}{2} \leq \theta_m \leq \frac{\pi}{2} \right). \end{aligned}$$

Replace

$$x_n \text{ by } -x_n \text{ if } m_r < 0.$$

Rotation

Update

$$z_0 = z_n + \theta_u.$$

Perform CORDIC Z – Reduction (circular mode). Then,

$$\begin{cases} x_n = K_1^2 \sqrt{m_r^2 + m_i^2} [\cos(\theta_m + \theta_u) - \sin(\theta_m + \theta_u)], \\ y_n = K_1^2 \sqrt{m_r^2 + m_i^2} [\sin(\theta_m + \theta_u) + \cos(\theta_m + \theta_u)]. \end{cases}$$

Two – Sided Scale Factor Correction

Perform scale factor iterations

$$\left. \begin{aligned} x &\leftarrow x - x 2^{-j} \\ y &\leftarrow y - y 2^{-j} \end{aligned} \right\} \text{ for } J = \{1, 3, 5, \dots, (2\lceil \frac{\pi}{4} \rceil - 1)\}$$

followed by a final right shift ($C = 2$).

The final values are

$$\begin{cases} x_n = \sqrt{m_r^2 + m_i^2} [\cos(\theta_m + \theta_u) - \sin(\theta_m + \theta_u)] = \text{Re} \{ M e^{(\theta_m + \theta_u)} \}, \\ y_n = \sqrt{m_r^2 + m_i^2} [\sin(\theta_m + \theta_u) + \cos(\theta_m + \theta_u)] = \text{Im} \{ M e^{(\theta_m + \theta_u)} \}. \end{cases}$$

Table 5.2: CORDIC Application of the Unitary Transformation

where

$$\begin{bmatrix} a'_r + i a'_i & b'_r + i b'_i \\ c'_r + i c'_i & d'_r + i d'_i \end{bmatrix}$$

is the result of the inner two-sided unitary transformation. The application of the outer transformation, for the real or imaginary parts, involves two vector rotations. The vectors are given by the rows for the left rotation and by the columns for the right rotation. Scale factor correction is postponed until both the left and right rotations are applied to employ two-sided scale factor correction.

The update of the z register by θ_u at the beginning of the rotation step in Table 5.2, is not a straight-forward addition. The problem arises because the update could possibly rotate the data element into a different quadrant of the complex plane. Figure 5.4 illustrates the problem for the case when the data element m lies in the first quadrant (right half-plane) and the update causes the argument to increase beyond $\pi/2$, rotating m into the second quadrant (left half-plane).

In the modified polar coordinate representation (4.6) of a complex number, the range of the argument is restricted to the principal values of *arctangent*. For purposes of mathematical correctness, information about the half-plane of the complex data element must be stored along with the value of the argument. This information may be encoded by an additional bit representing the half-plane of the argument.

A set of pre-processing rules can then be derived, to handle the addition of complex arguments, at the beginning of the rotation step in Table 5.2. The rules are tabulated in Table 5.3, where H is the bit-value representing the half-plane of the argument, R the sign of modulus, and θ the value of the argument restricted to the closed right half-plane (Figure 5.5).

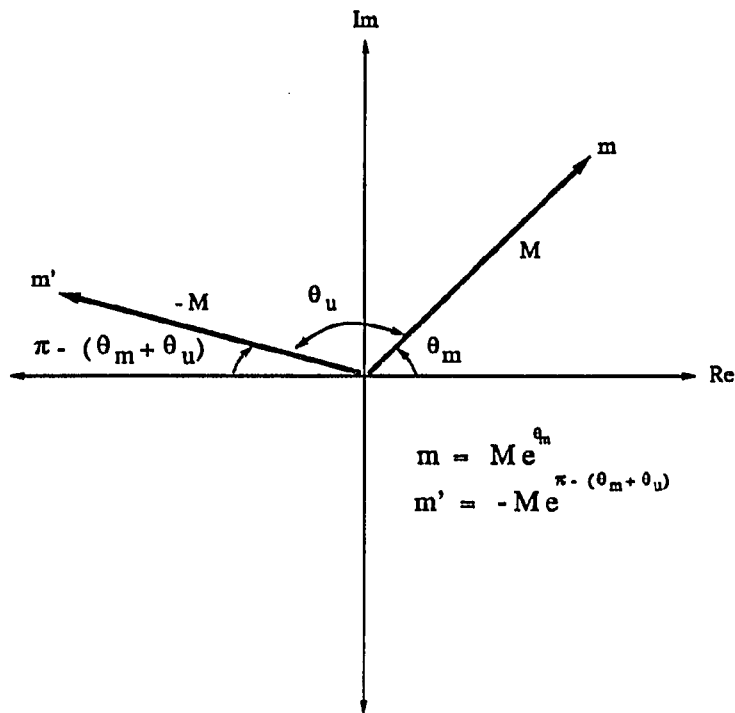


Figure 5.4: Rotation in the Complex Plane

To handle subtraction of arguments, a unary negation is also defined in Table 5.3. The update affects not only the z variable, but also the x variable, because the modified polar coordinate representation (4.6) allows for negative moduli.

In addition to applying the Q transformation, a scheme for generating the rotation and unitary angles is also required. From the discussion in § 4.5, it can be seen that any unitary angle that may be required, can be obtained at the end of the transformation step in Table 5.2. Rotation angles, can be computed using the CORDIC in the inverse tangent (circular mode y -reduction).

<i>Unary Negation</i>			
H/R	H ⁻ /R ⁻	Arg	Arg ⁻
h	h	θ	$-\theta$
<i>Addition</i>			
if $ \theta_1 + \theta_2 \leq \pi/2$			
$\theta_{sum} = \theta_1 + \theta_2$			
H ₁ /R ₁	H ₂ /R ₂	H _{sum} /R _{sum}	
h	h	h	
h	\bar{h}	1	
<i>else</i>			
$\theta_{sum} = -sign(\theta_1 + \theta_2) (\pi - \theta_1 + \theta_2)$			
H ₁ /R ₁	H ₂ /R ₂	H _{sum} /R _{sum}	
h	h	1	
h	\bar{h}	0	

Table 5.3: Pre-Processing Rules for Argument Addition

5.3 An Architecture for the Complex 2×2 Processor

In § 4.6 a two-step diagonalization scheme was presented. The scheme is composed of two Q transformations. § 5.2 discussed the use of CORDIC in generating angles for and applying the Q transformation. Based on these previous discussions an architecture for a complex 2×2 processor can be derived.

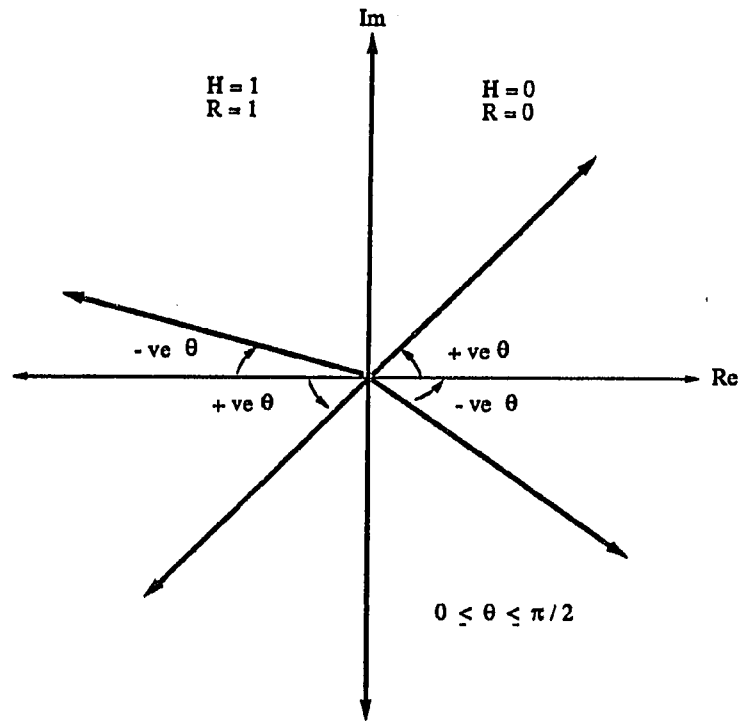


Figure 5.5: Angle and Modulus in Modified Polar Coordinates

Figures 5.6 and 5.7 show the various stages of computation and the respective CORDIC modules involved in each of the two \mathcal{Q} transformations of the two-step diagonalization scheme presented in § 4.6. The CORDIC polar transformation module essentially performs the transformation step in Table 5.2 and the CORDIC complex rotation module performs the rotation step in Table 5.2. The CORDIC complex rotation module uses the pre-processing rules described in Table 5.3 to compute the argument and sign of the modulus for the result.

In the application of the \mathcal{Q} transformation using CORDIC, a basic CORDIC processor like those presented in § 3.2, is required for each complex data element. However, additional complexity in control is required to handle extra computations required for complex data manipulations (Tables 5.2 & 5.3). To achieve maximum parallelism in

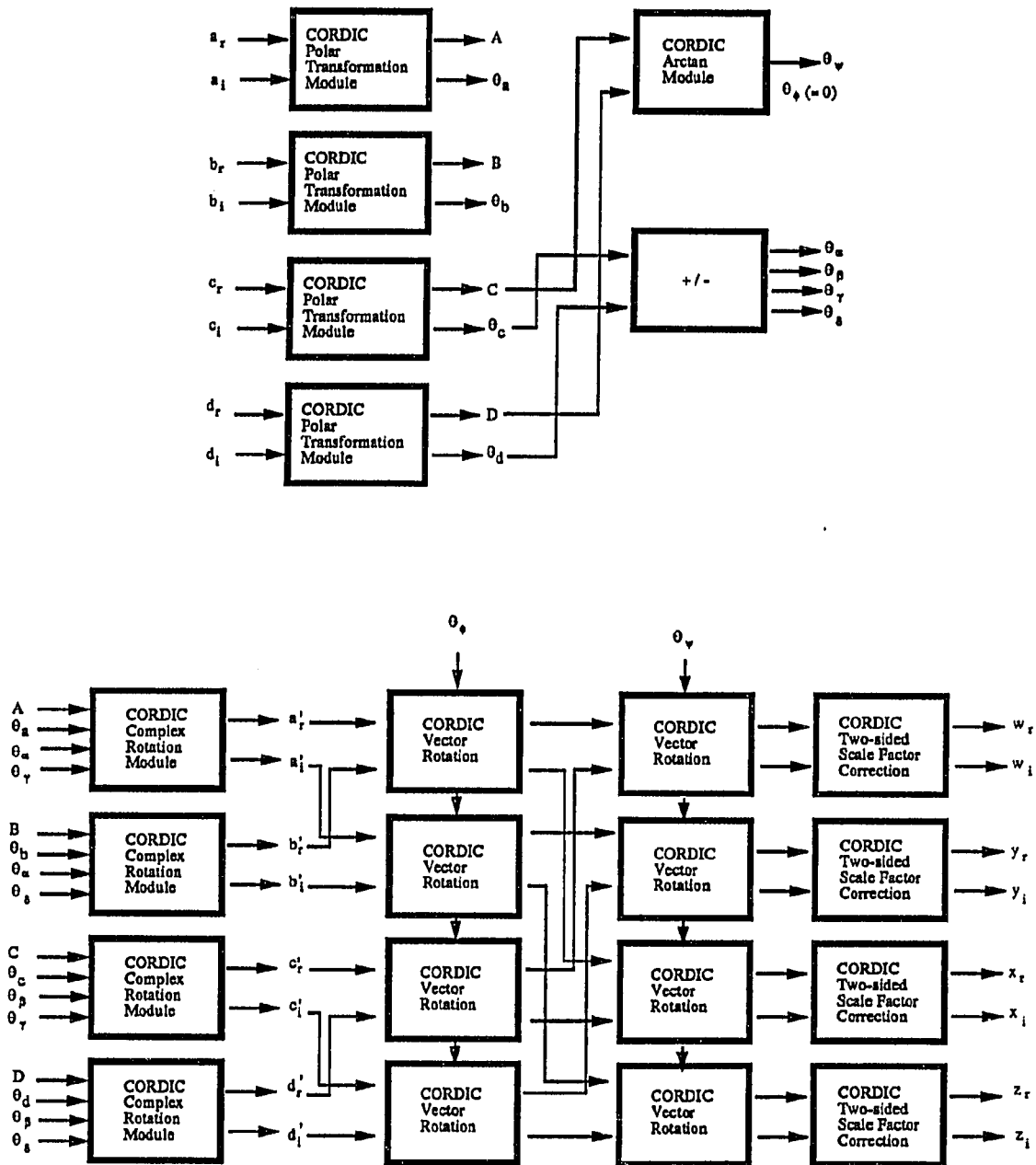


Figure 5.6: Diagonalization Step - I in CORDIC

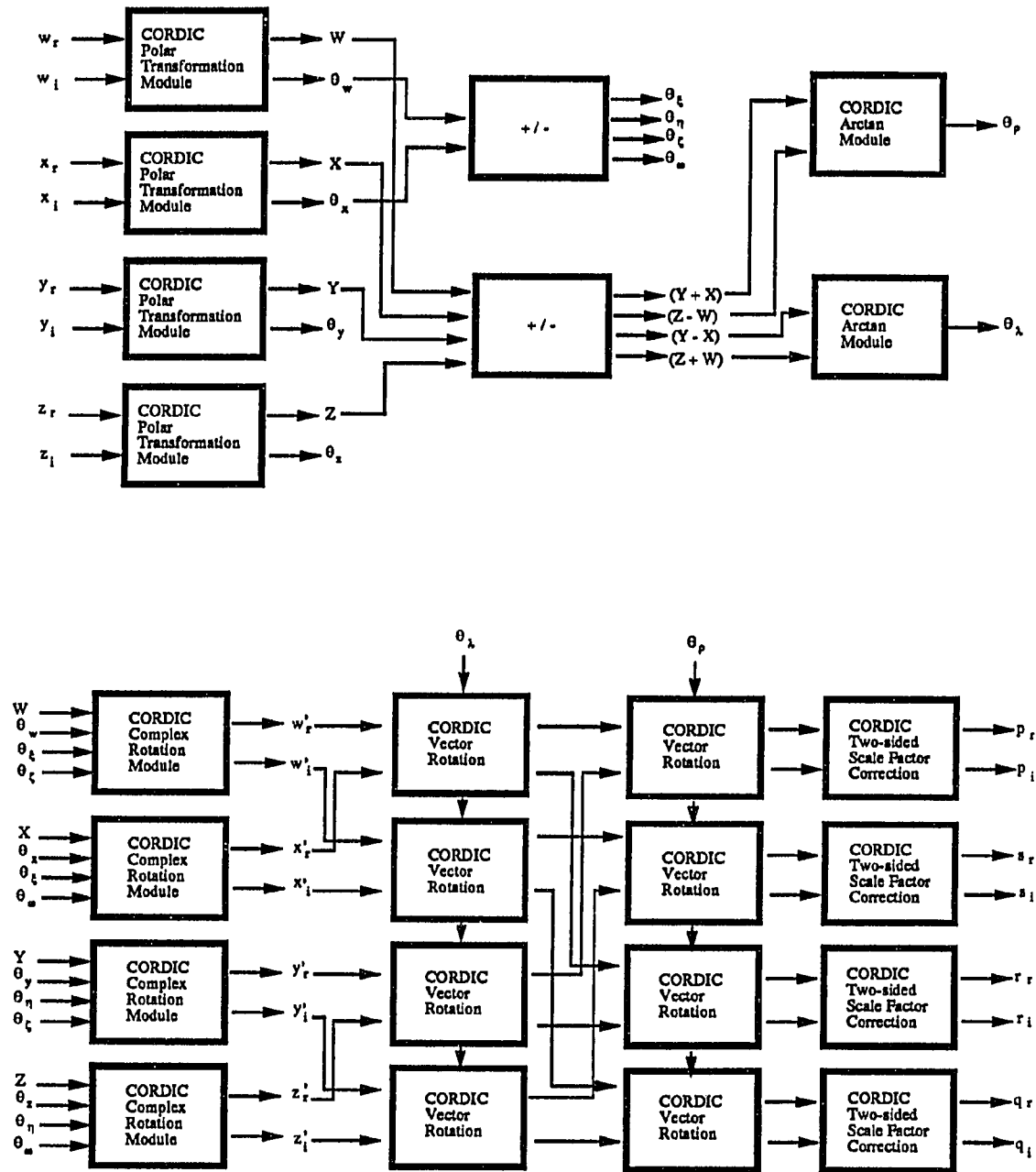


Figure 5.7: Diagonalization Step - II in CORDIC

the application of the Q transformation for the complex 2×2 problem, four CORDIC modules are needed.

The choice of floating-point (Figure 3.3) vs. fixed-point data path (Figure 3.5) CORDIC modules is not obvious. An argument for the floating-point data path CORDIC module may be made, considering that the negation of variables is necessitated in the application of the Q transformation using CORDIC (Tables 5.2 & 5.3).

The reason lies in the fact that, unary negation of floating-point data requires only the inversion of the sign bit, which can be done by the control unit implicitly; while a similar operation for fixed-point data requires the use of arithmetic hardware to compute the two's complement. However, floating-point operations have greater latency compared to fixed-point data manipulations.

Figures 5.6 and 5.7, indicate an adjacency in the pattern of communication of data and results of CORDIC iterations between the CORDIC modules. Therefore, it is more efficient to have register banks shared by neighboring CORDIC modules rather than a centralized register bank.

Furthermore, the results of some CORDIC operations, like the rotation angles are needed by all the CORDIC modules. A data bus linking the four register banks is also required. An architecture for the complex 2×2 processor is shown in Figure 5.8. Tables 5.4 and 5.5 detail the computation sequence and CORDIC module assignments to execute the two-step diagonalization scheme on the architecture in Figure 5.8.

5.4 Area/Time Analysis for the Complex 2×2 Processor

The CORDIC complex 2×2 processor as shown in Figure 5.8, is composed of four CORDIC modules and a central control unit. The CORDIC modules are similar to

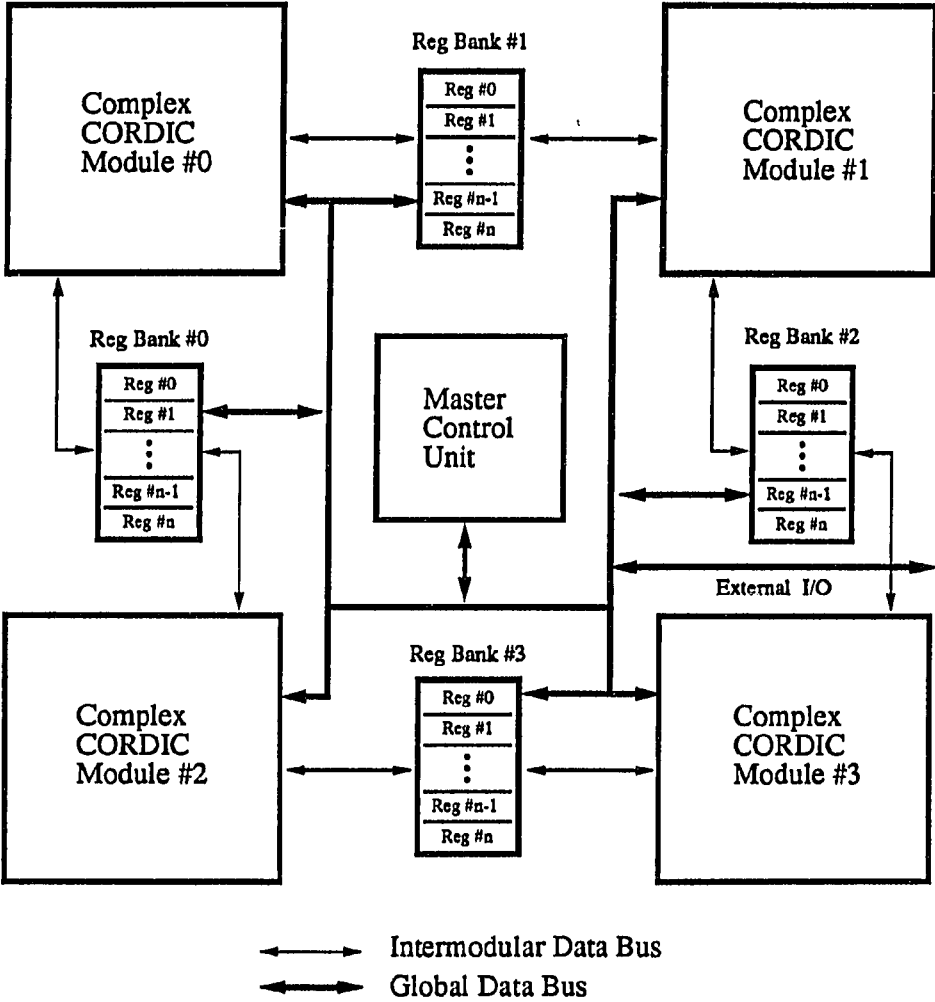


Figure 5.8: Complex 2 × 2 Processor Architecture

Step - I (First Q transformation)

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Polar Transformation	a_r, a_i	A, θ_a
CM #1	Polar Transformation	b_r, b_i	B, θ_b
CM #2	Polar Transformation	c_r, c_i	C, θ_c
CM #3	Polar Transformation	d_r, d_i	D, θ_d

CM#2 & CM#3 exchange C, θ_c , and D, θ_d .

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #2	Add, Subtract, Shift	θ_c, θ_d	$\left(\frac{\theta_d + \theta_c}{2}\right), \left(\frac{\theta_d - \theta_c}{2}\right)$
CM #3	Inverse Tangent	C, D	$\tan^{-1}\left(\frac{C}{D}\right)$

Broadcast results of CM #2 and CM #3 to other CORDIC modules

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Complex Rotation, 2-sided SFC	a_r, a_i	a'_r, a'_i
CM #1	Complex Rotation, 2-sided SFC	b_r, b_i	b'_r, b'_i
CM #2	Complex Rotation, 2-sided SFC	c_r, c_i	c'_r, c'_i
CM #3	Complex Rotation, 2-sided SFC	d_r, d_i	d'_r, d'_i

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Vector Rotation	a'_r, b'_r	a''_r, b''_r
CM #1	Vector Rotation	a'_i, b'_i	a''_i, b''_i
CM #2	Vector Rotation	c'_r, d'_r	c''_r, d''_r
CM #3	Vector Rotation	c'_i, d'_i	c''_i, d''_i

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Vector Rotation, 2-sided SFC	a''_r, c''_r	w_r, y_r
CM #1	Vector Rotation, 2-sided SFC	a''_i, c''_i	w_i, y_i
CM #2	Vector Rotation, 2-sided SFC	b''_r, d''_r	x_r, z_r
CM #3	Vector Rotation, 2-sided SFC	b''_i, d''_i	x_i, z_i

Table 5.4: CORDIC SVD Processor Algorithm - Step I

Step - II (Second Q transformation)

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Polar Transformation	w_r, w_i	W, θ_w
CM #1	Polar Transformation	y_r, x_i	X, θ_x
CM #2	Polar Transformation	y_r, y_i	Y, θ_y
CM #3	Polar Transformation	z_r, z_i	Z, θ_z

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Subtract, Shift	θ_w, θ_x	$\left(\frac{\theta_x - \theta_w}{2}, \left(\frac{\theta_w - \theta_x}{2}\right)\right)$
CM #1	Add, Shift	θ_w, θ_x	$\left(\frac{\theta_w + \theta_x}{2}\right)$
CM #2	Add, Subtract, Shift	W, Z	$Z + W, Z - W$
CM #3	Add, Subtract, Shift	X, Y	$Y + X, Y - X$

Broadcast results to CORDIC modules that require them

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #2	Inverse Tangent	W, Z	$\tan^{-1}\left(\frac{Y-X}{Z+W}\right)$
CM #3	Inverse Tangent	X, Y	$\tan^{-1}\left(\frac{Y+X}{Z-W}\right)$

Broadcast results of CM #2 and CM #3 to other CORDIC modules

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Complex Rotation, 2-sided SFC	w_r, w_i	w_r, w_i
CM #1	Complex Rotation, 2-sided SFC	x_r, x_i	x_r, x_i
CM #2	Complex Rotation, 2-sided SFC	y_r, y_i	y_r, y_i
CM #3	Complex Rotation, 2-sided SFC	z_r, z_i	z_r, z_i

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Vector Rotation	w_r, x_r	w_r'', x_r''
CM #1	Vector Rotation	w_i, x_i	w_i'', x_i''
CM #2	Vector Rotation	y_r, z_r	y_r'', z_r''
CM #3	Vector Rotation	y_i, z_i	y_i'', z_i''

Compute in Parallel

CORDIC Module	Computation	Inputs	Result(s)
CM #0	Vector Rotation, 2-sided SFC	w_r'', y_r''	w_r, y_r
CM #1	Vector Rotation, 2-sided SFC	w_i'', y_i''	w_i, y_i
CM #2	Vector Rotation, 2-sided SFC	x_r'', z_r''	x_r, z_r
CM #3	Vector Rotation, 2-sided SFC	x_i'', z_i''	x_i, z_i

Table 5.5: CORDIC SVD Processor Algorithm - Step II

those described in § 3.2 and § 3.3, with modifications in the control units to implement the schemes in Tables 5.2 & 5.3 for complex arithmetic using CORDIC.

The total execution time for the two-step diagonalization method can be computed from Tables 5.4 & 5.5. Let T_{Q_1} and T_{Q_2} , be the time required to compute the first and second Q transformations in the computation of the SVD. The time complexity analysis presented below is indicative of the maximum parallelism that can be exploited in the CORDIC SVD algorithm using the architecture of Figure 5.8.

From Table 5.4 we have,

$$T_{Q_1} = T_{PT} + T_{IT} + T_{CR} + 2T_{VR} + 2T_{TSFC},$$

where T_{PT} , T_{IT} , T_{CR} , T_{VR} and T_{TSFC} , are the times to compute the polar transformation of a complex number represented in orthogonal coordinates using CORDIC, the inverse tangent, the complex rotation, the vector rotation and the two-sided scale factor correction, respectively.

All of the times, except for T_{TSFC} , mentioned above are about the same as the time required to compute one CORDIC vector rotation. As in [11], we define this time for a vector rotation using CORDIC to be T_C . T_C is based on the time required to compute an addition (T_{ADD}), the basic operation in CORDIC. T_{ADD} depends on whether floating-point or fixed-point data paths are used and the technology of the implementation.

All further analysis assumes fixed-point data paths. For fixed-point data paths $T_C = n T_{ADD}$ where n is the word length and from [14], we know that $T_{TSFC} \approx 0.25T_C$. Thus,

$$T_{Q_1} \approx 5.5T_C.$$

A similar analysis for T_{Q_2} from Table 5.5, yields

$$\begin{aligned} T_{Q_2} &= T_{PT} + T_{IT} + T_{CR} + 2T_{VR} + 2T_{TSFC}, \\ &\approx 5.5T_C. \end{aligned}$$

The total time complexity of the two-step diagonalization method is therefore

$$\begin{aligned} T_{CSVD} &= T_{Q_2} + T_{Q_2}, \\ &\approx 11T_C. \end{aligned}$$

The area complexity of the architecture of Figure 5.8 can be expressed in terms of the area of a basic CORDIC processor, A_C . Assuming that the area complexity of the central control unit and the register banks is negligible compared to that of a CORDIC module, A_{CSVD} , the area complexity of the complex CORDIC SVD processor, can be derived as

$$A_{CSVD} \approx 4A_C.$$

The above assumption is reasonable in that each CORDIC module requires a barrel shifter of $O(n^2)$ area complexity with three adders, angle ROM, storage, control and moderately complex interconnection buses (Figure 3.3).

In the next chapter, the two-step diagonalization scheme proposed in § 4.6, is used along with the SVD-Jacobi method in deriving a systolic architecture for the SVD of a complex $n \times n$ matrix. The systolic array is based on the Brent-Luk-VanLoan array for the SVD of a real matrix. Some performance advantage is gained from the identity of the steps in the diagonalization scheme.

Chapter 6

A Systolic Array for Complex SVD

The two-step diagonalization scheme for the SVD of an arbitrary complex 2×2 matrix (§ 4.6) may be used as a basic step in the Jacobi-SVD method described in § 2.3 to compute the SVD of larger square matrices. The most effective systolic array known to implement the Jacobi-SVD method for real matrices is due to Brent, Luk and VanLoan. This chapter proposes an enhancement of the Brent-Luk-VanLoan systolic array for computing the SVD of arbitrary complex square matrices.

6.1 The Brent-Luk-VanLoan Systolic Array

On linear systolic arrays, the most efficient SVD algorithm is the Jacobi-like algorithm given by Brent and Luk [5]. The array implements a one-sided orthogonalization method due to Hestenes [43] and requires $O(mn \log n)$ time and $O(n)$ processors to compute the SVD of a real $m \times n$ matrix.

The Brent-Luk “parallel ordering” [5] described in § 2.3, was extended to a square systolic array architecture with $O(n^2)$ processors by Brent, Luk and VanLoan [6]. The array implements the SVD-Jacobi method discussed in § 2.3. It is capable of executing a sweep of the SVD-Jacobi method in $O(n)$ time and is conjectured to require $O(\log n)$ sweeps for convergence. The proof of convergence for the Jacobi-SVD procedure with “parallel ordering” is due to Park and Luk [60].

The Brent-Luk-VanLoan systolic array is primarily intended to compute the SVD

of a real $n \times n$ matrix, although the SVD of an $m \times n$ matrix can be computed in $m + O(n \log n)$ time. The array as described in [6] is similar to the array proposed in [5] for the eigenvalue decomposition of a symmetric matrix.

The Brent-Luk-VanLoan systolic array is an expandable, mesh-connected array of processors, where each processor contains a 2×2 sub-matrix of the input matrix $M \in R^{n \times n}$. Assuming that n is even, the Brent-Luk-VanLoan systolic array is a square array of $n/2 \times n/2$ processors. Before the computation begins, processor P_{ij} contains

$$\begin{bmatrix} m_{2i-1,2j-1} & m_{2i-1,2j} \\ m_{2j,2j-1} & m_{2j,2j} \end{bmatrix}$$

where $(i, j = 1, \dots, \frac{n}{2})$.

Each processor P_{ij} is connected to its “diagonally” nearest neighbors $P_{i\pm 1, j\pm 1}$, $(1 < i, j < \frac{n}{2})$. The Brent-Luk-VanLoan systolic array, with 16 processors for $n = 8$, is shown in Figure 6.1. The interconnections between the processors facilitate data exchange to implement the “parallel ordering” of Brent-Luk. The processor communication links for data interchange are shown in Figure 6.2.

The Brent-Luk “parallel ordering” permits Jacobi rotations (2.2) to be applied, in parallel, in groups of $n/2$. The angles for the $n/2$ Jacobi rotations are generated by the $n/2$ processors on the main diagonal of the array. The diagonal processors P_{ii} ($i = 1, \dots, \frac{n}{2}$) in the array have a more important role in the computation of the SVD when compared to the off-diagonal processors P_{ij} ($i \neq j, 1 \leq i, j \leq \frac{n}{2}$).

The application of a two-sided Jacobi rotation affects only the row and column of the diagonal processor generating the angles. In an idealized situation, the diagonal processors may broadcast the rotation angles, along the row and the column corresponding to their position in the array, in constant time. Each off-diagonal processor

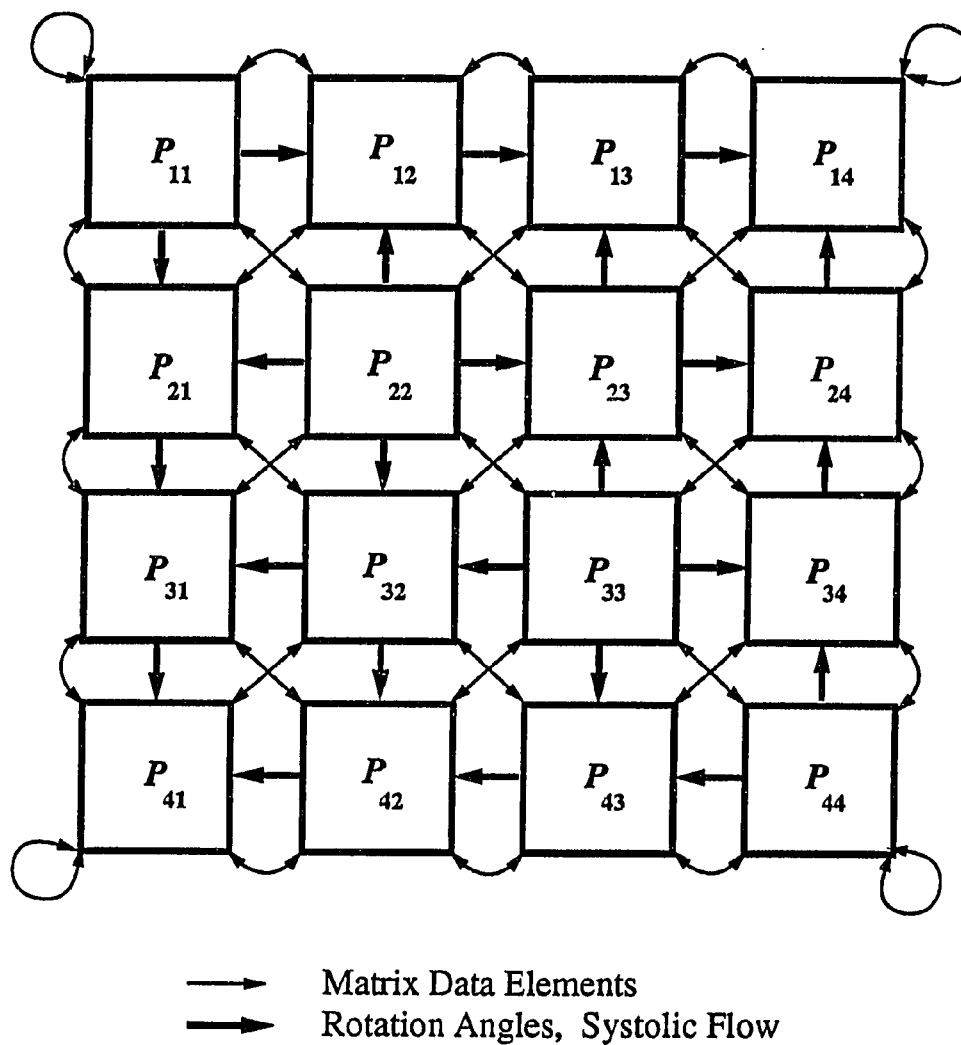


Figure 6.1: The Brent-Luk-VanLoan Systolic Array

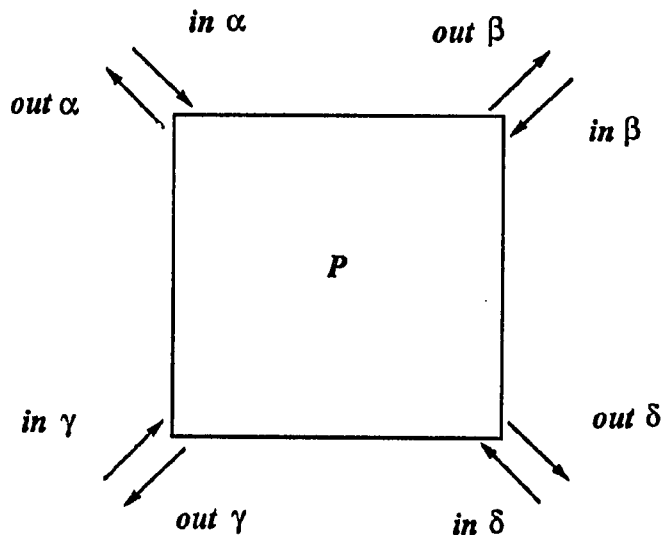


Figure 6.2: Interprocessor Communication Links for Processors

applies a two-sided rotation using the angles generated by the diagonal processors, in the same row and column with respect to its location in the array.

In general, a processor P_{ij} contains four real data elements

$$\begin{bmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{bmatrix}.$$

At each time step, the diagonal processors P_{ii} compute the rotation pairs (c_i^L, s_i^L) and (c_i^R, s_i^R) to annihilate their off-diagonal elements β_{ii} and γ_{ii} . This is essentially the SVD of the real 2×2 matrix stored at the diagonal processors and can be done using the methods discussed in § 2.4.

The annihilation of the off-diagonal elements at the diagonal processors may be expressed as

$$\begin{bmatrix} c_i^L & -s_i^L \\ s_i^L & c_i^L \end{bmatrix} \begin{bmatrix} \alpha_{ii} & \beta_{ii} \\ \gamma_{ii} & \delta_{ii} \end{bmatrix} \begin{bmatrix} c_i^R & s_i^R \\ -s_i^R & c_i^R \end{bmatrix} = \begin{bmatrix} \alpha'_{ii} & 0 \\ 0 & \delta'_{ii} \end{bmatrix}. \quad (6.1)$$

The two-sided Jacobi rotations represented by (6.1), which annihilate β_{ii} and γ_{ii} for $(i = 1, \dots, \frac{n}{2})$, are completed when the off-diagonal processors P_{ij} ($i \neq j$), each

perform the transformation

$$\begin{bmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{bmatrix} \Rightarrow \begin{bmatrix} \alpha'_{ij} & \beta'_{ij} \\ \gamma'_{ij} & \delta'_{ij} \end{bmatrix},$$

where

$$\begin{bmatrix} c_i^L & -s_i^L \\ s_i^L & c_i^L \end{bmatrix} \begin{bmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{bmatrix} \begin{bmatrix} c_j^R & s_j^R \\ -s_j^R & c_j^R \end{bmatrix} = \begin{bmatrix} \alpha'_{ij} & \beta'_{ij} \\ \gamma'_{ij} & \delta'_{ij} \end{bmatrix}. \quad (6.2)$$

It is clear from (6.1) and (6.2) that, for the completion of the $n/2$ parallel two-sided Jacobi rotations in constant time, the rotation angles θ_i^L and θ_i^R generated by a diagonal processor P_{ii} , must be broadcast along the i^{th} row and the i^{th} column respectively, in constant time. The broadcast would permit an off-diagonal processor P_{ij} to have access to the rotation angles θ_i^L and θ_j^R when required.

A step in the "parallel ordering" (2.5) is complete when columns and corresponding rows are interchanged between adjacent processors so that a new set of n off-diagonal elements is ready to be annihilated by the diagonal processors during the next computational step. The communication links at each processor (Figure 6.2) are interconnected as shown in Figure 6.1 to facilitate the data exchange as required by the "parallel ordering".

Formally, the interconnections may be specified as

$$\text{out } \alpha_{i,j} \Leftrightarrow \begin{cases} \text{in } \alpha_{i,j} & \text{if } i=1, j=1 \\ \text{in } \beta_{i,j-1} & \text{if } i=1, j>1 \\ \text{in } \gamma_{i-1,j} & \text{if } i>1, j=1 \\ \text{in } \delta_{i-1,j-1} & \text{if } i>1, j>1 \end{cases}, \quad \text{out } \beta_{i,j} \Leftrightarrow \begin{cases} \text{in } \alpha_{i,j+1} & \text{if } i=1, j<\frac{n}{2} \\ \text{in } \beta_{i,j} & \text{if } i=1, j=\frac{n}{2} \\ \text{in } \gamma_{i-1,j+1} & \text{if } i>1, j<\frac{n}{2} \\ \text{in } \delta_{i-1,j} & \text{if } i>1, j=\frac{n}{2} \end{cases}, \quad (6.3)$$

and

$$\text{out } \gamma_{i,j} \Leftrightarrow \begin{cases} \text{in } \alpha_{i+1,j} & \text{if } i<\frac{n}{2}, j=1 \\ \text{in } \beta_{i-1,j+1} & \text{if } i<\frac{n}{2}, j>1 \\ \text{in } \gamma_{i,j} & \text{if } i=\frac{n}{2}, j=1 \\ \text{in } \delta_{i,j+1} & \text{if } i=\frac{n}{2}, j>1 \end{cases}, \quad \text{out } \delta_{i,j} \Leftrightarrow \begin{cases} \text{in } \alpha_{i+1,j+1} & \text{if } i<\frac{n}{2}, j<\frac{n}{2} \\ \text{in } \beta_{i+1,j} & \text{if } i<\frac{n}{2}, j=\frac{n}{2} \\ \text{in } \gamma_{i,j+1} & \text{if } i=\frac{n}{2}, j<\frac{n}{2} \\ \text{in } \delta_{i,j} & \text{if } i=\frac{n}{2}, j=\frac{n}{2} \end{cases}.$$

Since a processor in the array stores a 2×2 sub-matrix, each processor must perform some diagonal data interchange to ensure that the "parallel ordering" is properly

implemented. This interchange depends on the location index of the processor P_{ij} in the array. The algorithm is shown in Table 6.1.

if $i = 1$ and $j = 1$ then $\begin{bmatrix} \text{out } \alpha \leftarrow \alpha; & \text{out } \beta \leftarrow \beta \\ \text{out } \gamma \leftarrow \gamma; & \text{out } \delta \leftarrow \delta \end{bmatrix}$
else if $i = 1$ then $\begin{bmatrix} \text{out } \alpha \leftarrow \beta; & \text{out } \beta \leftarrow \alpha \\ \text{out } \gamma \leftarrow \delta; & \text{out } \delta \leftarrow \gamma \end{bmatrix}$
else if $j = 1$ then $\begin{bmatrix} \text{out } \alpha \leftarrow \gamma; & \text{out } \beta \leftarrow \delta \\ \text{out } \gamma \leftarrow \alpha; & \text{out } \delta \leftarrow \beta \end{bmatrix}$
else then $\begin{bmatrix} \text{out } \alpha \leftarrow \delta; & \text{out } \beta \leftarrow \gamma \\ \text{out } \gamma \leftarrow \beta; & \text{out } \delta \leftarrow \alpha \end{bmatrix}$
{wait for outputs to propagate to inputs of adjacent processors}
then $\begin{bmatrix} \text{in } \alpha \leftarrow \alpha; & \text{in } \beta \leftarrow \beta \\ \text{in } \gamma \leftarrow \gamma; & \text{in } \delta \leftarrow \delta \end{bmatrix}$

Table 6.1: Algorithm Interchange

It cannot realistically be expected that the rotation angles can be broadcast in constant time for any array size. However, by assuming that the rotation parameters can be transmitted between adjacent processors in constant time, Brent-Luk-VanLoan specify a scheme to stagger computations that precludes the need for broadcast of rotation parameters, but still completes a sweep of the “parallel ordering” in $O(n)$ time.

Let $\Delta_{ij} = |i - j|$ denote the distance of processor P_{ij} from the diagonal. The operation of processor P_{ij} is then delayed by Δ_{ij} time units relative to the operation

of the diagonal processors. This is to allow sufficient time for the rotation parameters to be propagated at unit speed along each row and column of the processor array.

Also, a processor cannot commence a rotation until data from earlier rotations are available on all its input lines. Processor P_{ij} needs data from its four neighbors $P_{i\pm 1, j\pm 1}$ ($1 < i, j < \frac{n}{2}$). Dependencies for other processors can be seen from Figure 6.1. Since

$$|\Delta_{ij} - \Delta_{i\pm 1, j\pm 1}| \leq 2,$$

it is sufficient for processor P_{ij} to be idle for two time steps while waiting for processors $P_{i\pm 1, j\pm 1}$ to complete their possibly delayed steps. Figure 6.3 illustrates the staggering of computations to avoid broadcast of rotation parameters. The communication links for the processors have to be augmented to handle the systolic flow of the rotation angles. The processors are modified, depending on their position relative to the diagonal, as shown in Figure 6.4.

The algorithm, due to Brent-Luk-VanLoan, for the computation at each processor of the array, assuming that computation begins at time step $T = 0$, is given in Table 6.2. Also, it is assumed that each time step has non-overlapping read and write phases. The result of a write step T is available at the read phase of step $T + 1$, $T + 2$, and $T + 3$ in a neighboring processor, but does not interfere with a read step at T in a neighboring processor.

Figure 6.5 shows the timing of the systolic data and rotation angle exchange on the Brent-Luk-VanLoan systolic array with staggered computations. The numbers on the communication links correspond to the first time steps at which data is available on various processor input lines.

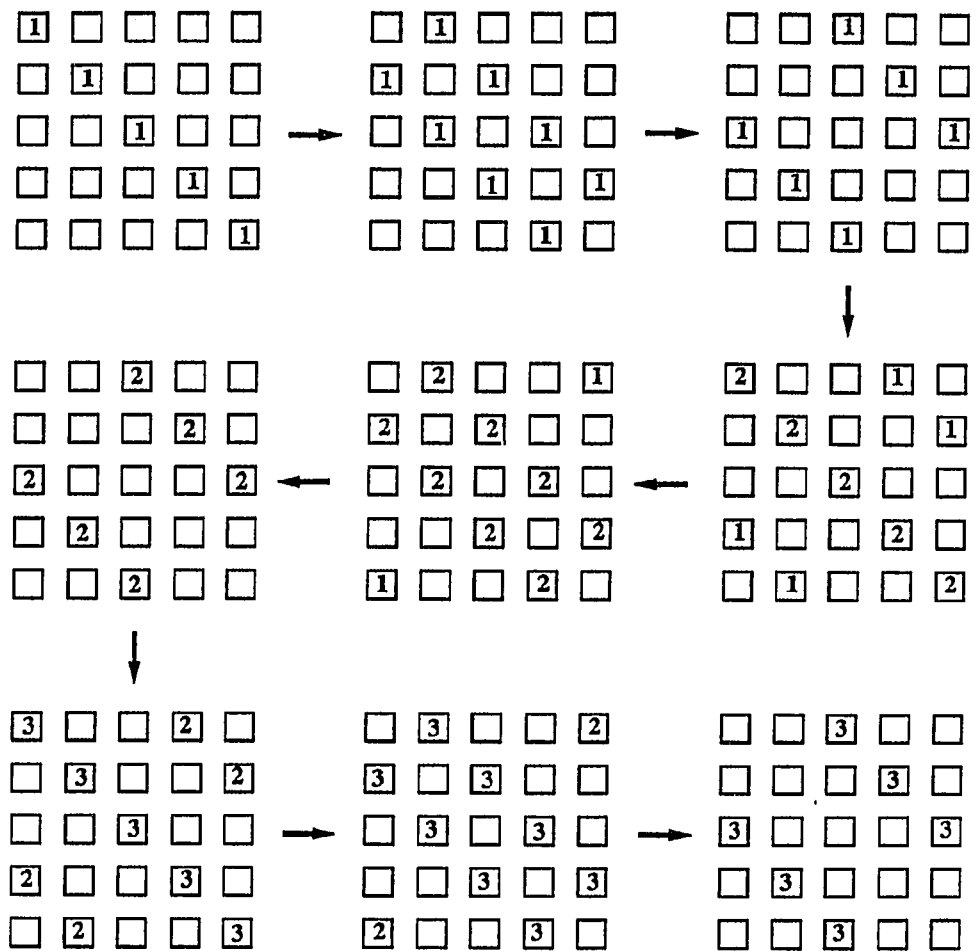


Figure 6.3: Staggering of Computations in the BLV Array

```

if ( $T \geq \Delta$ ) and ( $T - \Delta \equiv 0 \pmod{3}$ ) then
  begin
    if  $T \neq \Delta$  then  $\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \leftarrow \begin{bmatrix} \text{in } \alpha & \text{in } \beta \\ \text{in } \gamma & \text{in } \delta \end{bmatrix}$ ;

    if  $\Delta = 0$  then {Diagonal Processor}
      Use (2.13) to determine  $\theta_L, \theta_R$  and  $\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \rightarrow \begin{bmatrix} \alpha' & 0 \\ 0 & \delta' \end{bmatrix}$ 

    else {Off - Diagonal Processor}
      begin
         $\theta_L \leftarrow \text{in } h; \theta_R \leftarrow \text{in } v$ ;

        
$$\begin{bmatrix} c_i^L & -s_i^L \\ s_i^L & c_i^L \end{bmatrix} \begin{bmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{bmatrix} \begin{bmatrix} c_j^R & s_j^R \\ -s_j^R & c_j^R \end{bmatrix} = \begin{bmatrix} \alpha'_{ij} & \beta'_{ij} \\ \gamma'_{ij} & \delta'_{ij} \end{bmatrix}$$


        end;
        out  $h \leftarrow \theta_L$ ; out  $v \leftarrow \theta_R$ ;
        if  $i > j$  then set out  $\beta$  as in Algorithm Interchange;
        if  $i < j$  then set out  $\gamma$  as in Algorithm Interchange;
      end
    else if ( $T \geq \Delta$ ) and ( $T - \Delta \equiv 1 \pmod{3}$ ) then
      begin
        if ( $i = 1$ ) or ( $j = 1$ ) then set out  $\alpha$  as in Algorithm Interchange;
        if ( $i = \frac{n}{2}$ ) or ( $j = \frac{n}{2}$ ) then set out  $\alpha$  as in Algorithm Interchange;
      end
    else if ( $T \geq \Delta$ ) and ( $T - \Delta \equiv 2 \pmod{3}$ ) then
      begin
        if ( $i > 1$ ) or ( $j > 1$ ) then set out  $\alpha$  as in Algorithm Interchange;
        if ( $i \leq j$ ) then set out  $\beta$  as in Algorithm Interchange;
        if ( $i \geq j$ ) then set out  $\gamma$  as in Algorithm Interchange;
        if ( $i < \frac{n}{2}$ ) or ( $j < \frac{n}{2}$ ) then set out  $\delta$  as in Algorithm Interchange;
      end
    else
      Do nothing this time step.
  end

```

Table 6.2: BLV Array : Processor Algorithm

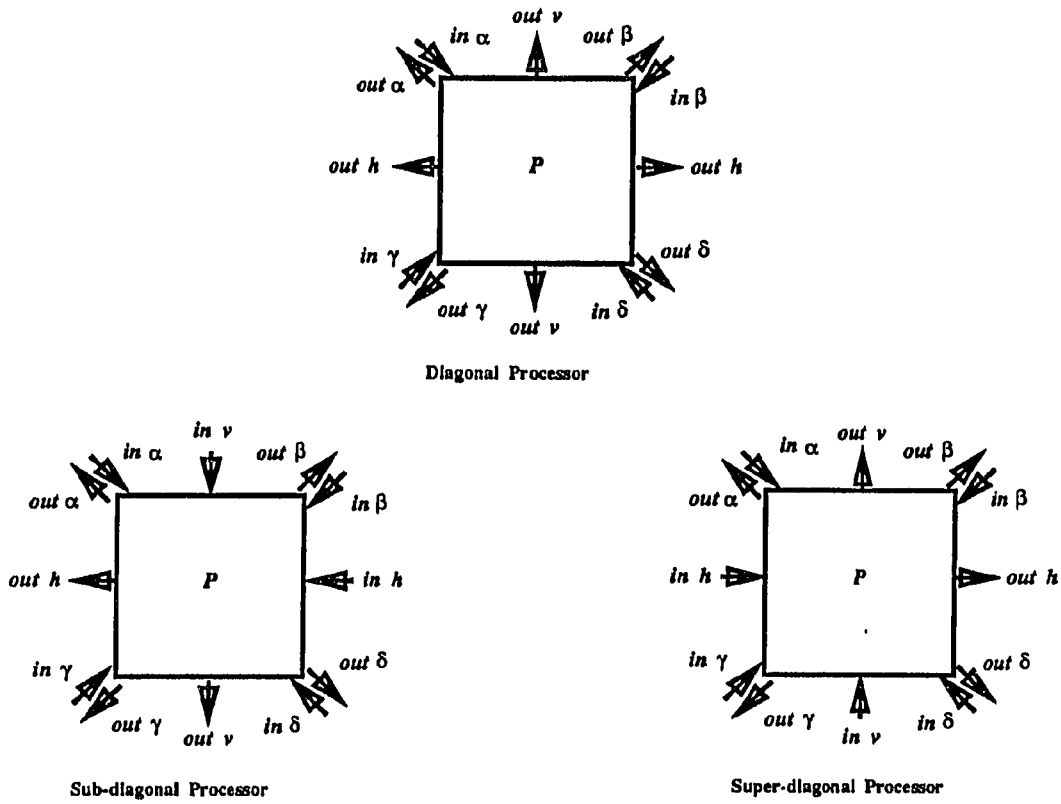


Figure 6.4: Processors for the BLV Array

6.2 A Systolic Array for Complex SVD

Although the Brent-Luk-VanLoan systolic array can be adapted in a straightforward manner to compute the SVD of a square, arbitrary complex matrix and each processor, now diagonalizes a complex 2×2 matrix as opposed to a real 2×2 matrix, there are some fundamental differences.

With the use of the two-step diagonalization scheme proposed in this thesis in § 4.6, and the CORDIC implementation of the scheme as presented in § 5.2, it is clear that the number of rotational parameters that are generated at each step of the Jacobi-SVD method is significantly greater than the real SVD case.

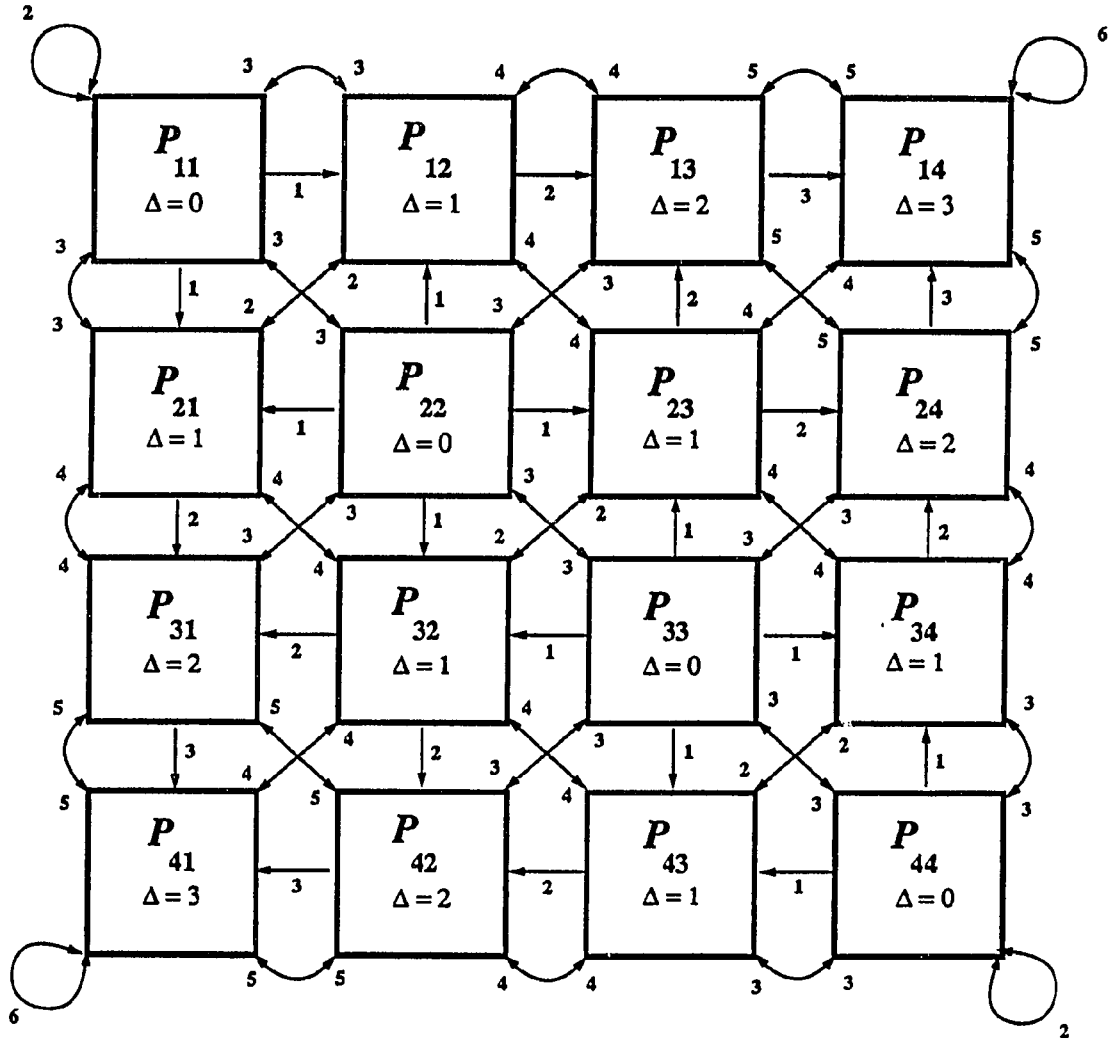


Figure 6.5: Data Exchange Timing for the BLV Array

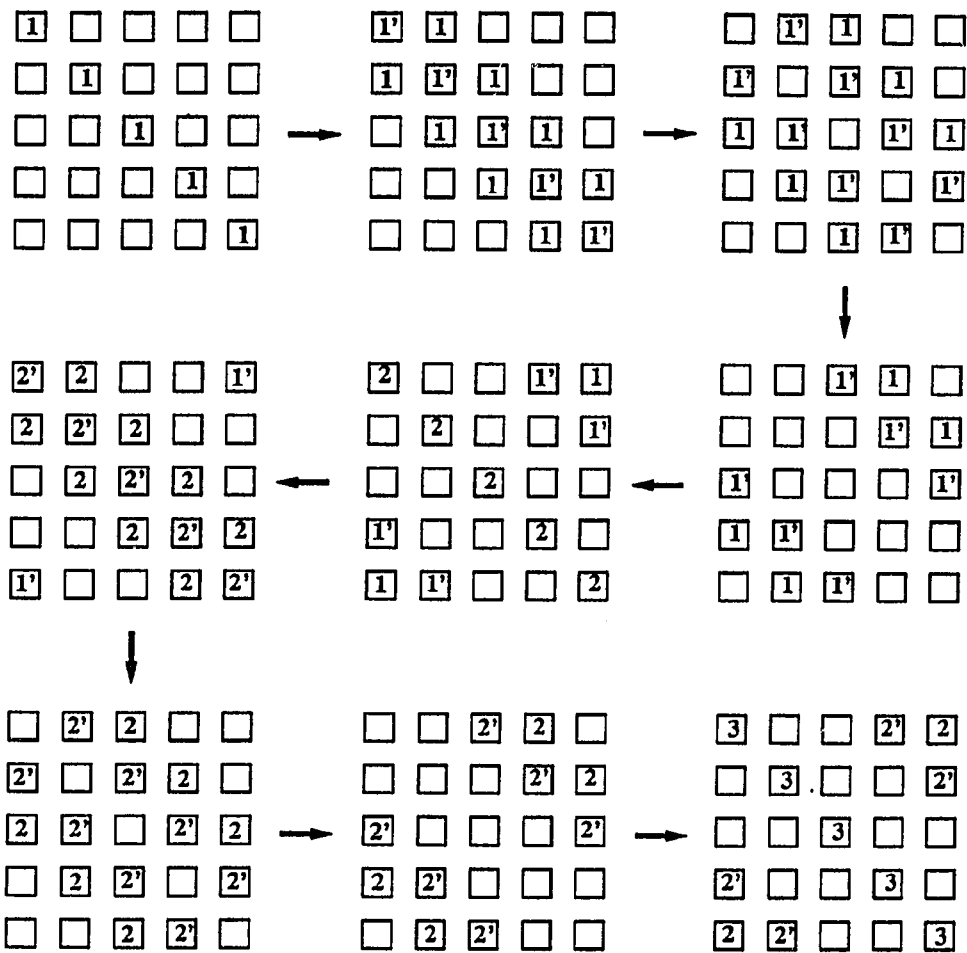


Figure 6.6: Staggering of Computations on the Complex SVD Array

The two-step diagonalization scheme was shown to be composed of two Q transformations. Each Q transformation requires the generation and application of six angles, four unitary angles and two rotational angles. Thus, the total count of the angles amounts to twelve in all; eight unitary and four rotational angles.

In a direct adaptation of the Brent-Luk-VanLoan systolic array for systolic computation of the SVD of a complex matrix, six angles must be propagated along both the rows and columns of processors on the main diagonal. These processors are responsible for the generation of the angles, in addition to applying them to diagonalize the 2×2 matrices stored on them. Also, while the diagonal processors are computing the second Q transformation, the immediately off-diagonal processors are idle; even though the rotational parameters needed for the application of the first Q transformation are available at the diagonal processors.

The realization that a single step computation of the SVD (of complex 2×2 matrices) using (4.21 or 4.25) would be infeasible for practical computation on special-purpose hardware motivated the repeated use of a powerful but implementable transformation (the Q transformation) and the two-step diagonalization scheme (§ 4.6). Also, the identity of the steps in the two-step diagonalization scheme was intended to prompt an overlapping of computation across the array.

The novel scheme proposed then, is to chase the first and second Q transformations down the diagonals, one behind the other as shown in Figure 6.6. Thus, while the processors on the main diagonal are still computing the parameters for the second Q transformation step, the immediately off-diagonal processors are applying the first Q transformation.

The pipelining of Q transformations as shown in Figure 6.6, imposes a problem in that now data can only be exchanged a computation step later than in the Brent-

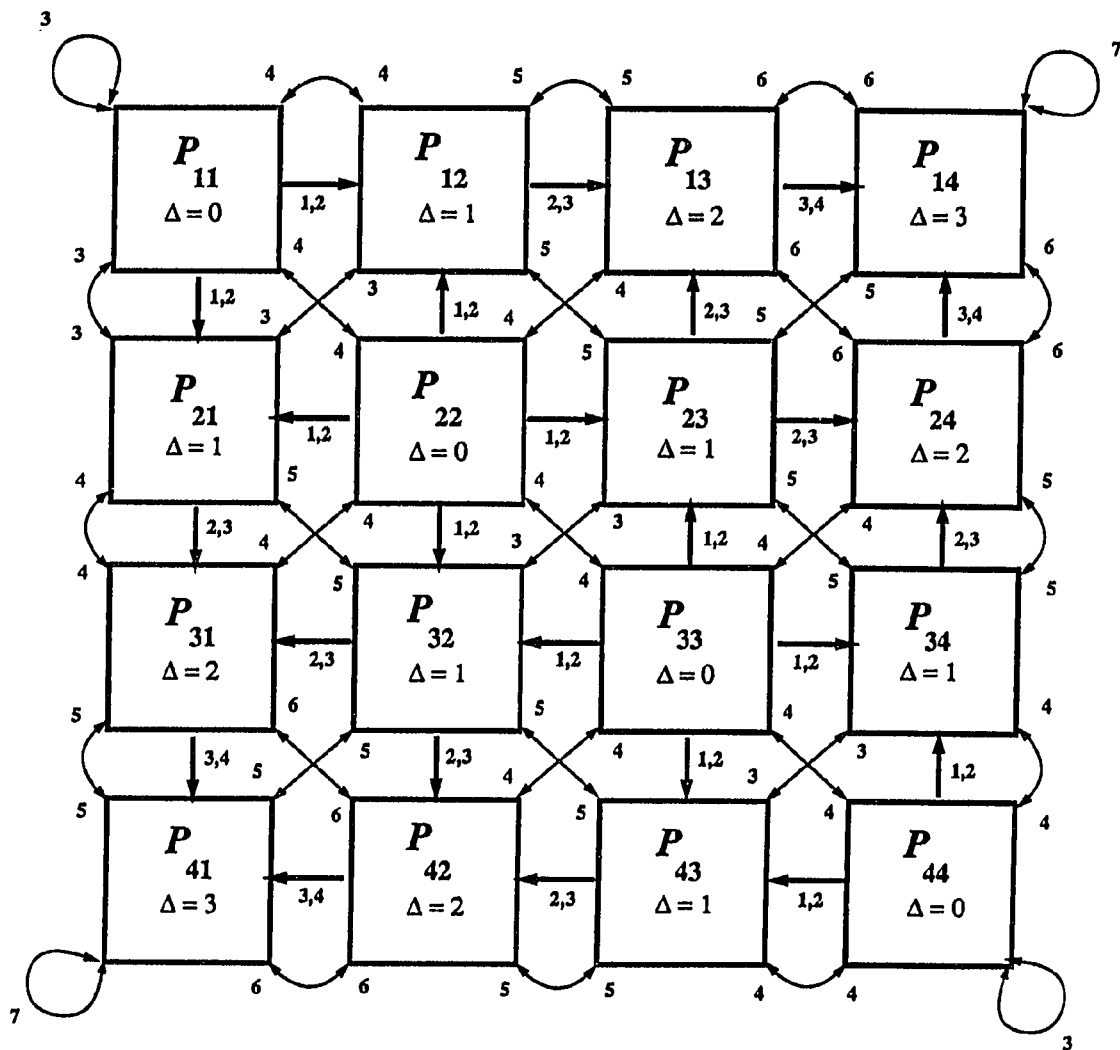


Figure 6.7: Data Exchange Timing for the Complex SVD Array

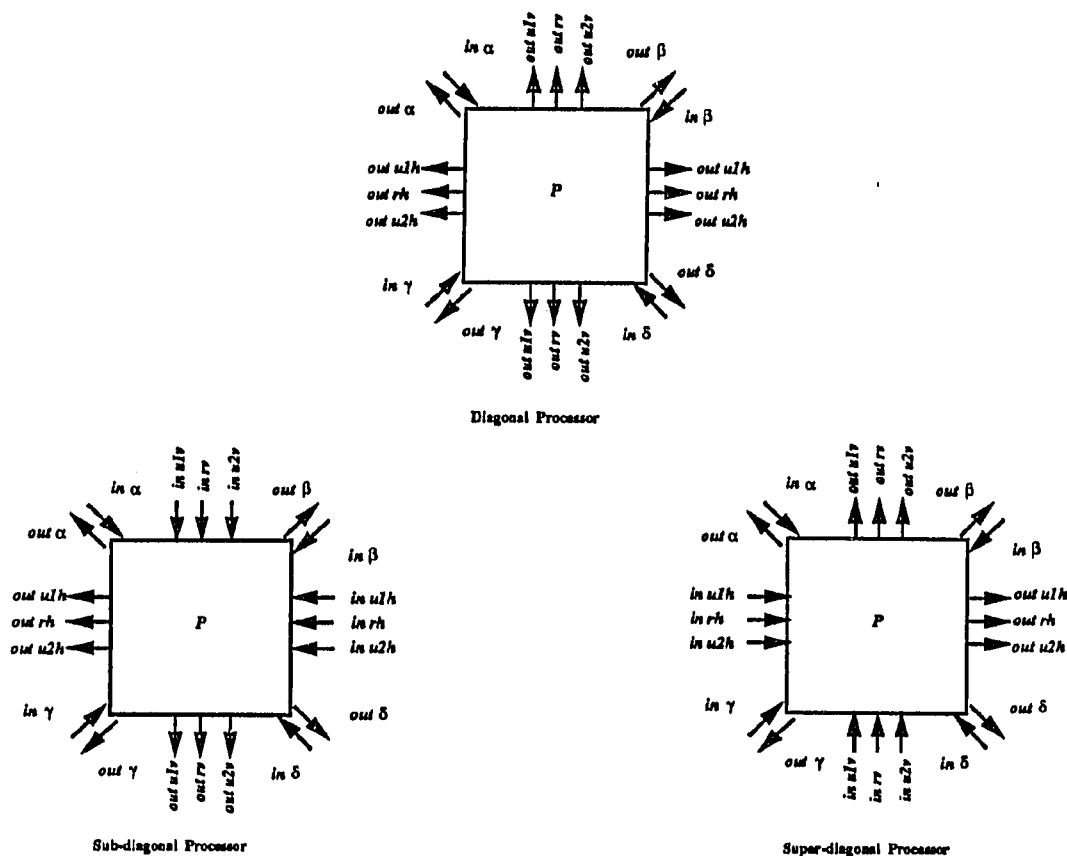


Figure 6.8: Processors for the Complex SVD Array

Luk-VanLoan systolic array. The availability of data to implement the data exchange as per the “parallel ordering” of Brent-Luk, is shown in Figure 6.7.

The change in the timing of data exchange also affects the processor algorithm (Table 6.2). The algorithm that governs the behavior of a processor in the complex SVD array is shown in Table 6.3.

The added systolicity and pipelining due to the staggering of computations, improves performance and reduces the communication load per step for the propagation of rotational parameters through the array. The processor utilization increases to 50% from 33% for the Brent-Luk-VanLoan systolic array due to the fact that processors

```

if ( $T \geq \Delta$ ) and ( $T - \Delta \equiv 0 \pmod{4}$ ) then
  begin
    if  $T \neq \Delta$  then  $\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \leftarrow \begin{bmatrix} \text{in } \alpha & \text{in } \beta \\ \text{in } \gamma & \text{in } \delta \end{bmatrix}$ ;

    if  $\Delta = 0$  then {Diagonal Processor}
      Compute first  $Q$  transformation using (4.41)
      Generate  $\theta_\alpha, \theta_\beta, \theta_\gamma, \theta_\delta, \theta_\phi$  and  $\theta_\psi$ .
    else {Off - Diagonal Processor}
      begin
         $\theta_\alpha \leftarrow \text{in } u1h$ ;  $\theta_\beta \leftarrow \text{in } u2h$ ;  $\theta_\phi \leftarrow \text{in } rh$ ;
         $\theta_\gamma \leftarrow \text{in } ulv$ ;  $\theta_\delta \leftarrow \text{in } u2v$ ;  $\theta_\psi \leftarrow \text{in } rv$ ;
        Apply  $Q$  transformation.
         $\text{out } u1h \leftarrow \theta_\alpha$ ;  $\text{out } u2h \leftarrow \theta_\beta$ ;  $\text{out } rh \leftarrow \theta_\phi$ ;
         $\text{out } ulv \leftarrow \theta_\gamma$ ;  $\text{out } u2v \leftarrow \theta_\delta$ ;  $\text{out } rv \leftarrow \theta_\psi$ ;
      end;
    end
  else if ( $T \geq \Delta$ ) and ( $T - \Delta \equiv 1 \pmod{4}$ ) then
    begin
      if  $\Delta = 0$  then {Diagonal Processor}
        Compute second  $Q$  transformation using (4.42)
        Generate  $\theta_\xi, \theta_\eta, \theta_\zeta, \theta_\omega, \theta_\lambda$  and  $\theta_\rho$ .
      else {Off - Diagonal Processor}
        begin
           $\theta_\xi \leftarrow \text{in } u1h$ ;  $\theta_\eta \leftarrow \text{in } u2h$ ;  $\theta_\lambda \leftarrow \text{in } rh$ ;
           $\theta_\zeta \leftarrow \text{in } ulv$ ;  $\theta_\omega \leftarrow \text{in } u2v$ ;  $\theta_\rho \leftarrow \text{in } rv$ ;
          Apply  $Q$  transformation.
           $\text{out } u1h \leftarrow \theta_\xi$ ;  $\text{out } u2h \leftarrow \theta_\eta$ ;  $\text{out } rh \leftarrow \theta_\lambda$ ;
           $\text{out } ulv \leftarrow \theta_\zeta$ ;  $\text{out } u2v \leftarrow \theta_\omega$ ;  $\text{out } rv \leftarrow \theta_\rho$ ;
        end;
        if  $i > j$  then set  $\text{out } \beta$  as in Algorithm Interchange;
        if  $i < j$  then set  $\text{out } \gamma$  as in Algorithm Interchange;
      end
    else if ( $T \geq \Delta$ ) and ( $T - \Delta \equiv 2 \pmod{4}$ ) then
      begin
        if ( $i = 1$ ) or ( $j = 1$ ) then set  $\text{out } \alpha$  as in Algorithm Interchange;
        if ( $i = \frac{n}{2}$ ) or ( $j = \frac{n}{2}$ ) then set  $\text{out } \alpha$  as in Algorithm Interchange;
      end
    else if ( $T \geq \Delta$ ) and ( $T - \Delta \equiv 3 \pmod{4}$ ) then
      begin
        if ( $i > 1$ ) or ( $j > 1$ ) then set  $\text{out } \alpha$  as in Algorithm Interchange;
        if ( $i \leq j$ ) then set  $\text{out } \beta$  as in Algorithm Interchange;
        if ( $i \geq j$ ) then set  $\text{out } \gamma$  as in Algorithm Interchange;
        if ( $i < \frac{n}{2}$ ) or ( $j < \frac{n}{2}$ ) then set  $\text{out } \delta$  as in Algorithm Interchange;
      end
    else
      Do nothing this time step.
  end

```

Table 6.3: Complex SVD : Processor Algorithm

along any diagonal are now active twice very four computation steps as opposed to once every three computation steps in the Brent-Luk-VanLoan systolic array.

The communication links for the processors in the complex SVD array need additional links to handle the increase in the number of rotational parameters exchanged. The processors for the Brent-Luk-VanLoan systolic array (Figure 6.4) are modified for the complex SVD array as shown in Figure 6.8.

6.3 Q Transformations for the EVD of Hermitian Matrices

Cavallaro and Elster [12] discuss the extension of the Brent-Luk-VanLoan systolic array [5] for the eigenvalue decomposition (EVD) of a Hermitian matrix. The basic step in the algorithm for computation of the eigenvalues involves the symmetric diagonalization of a complex 2×2 matrix. Precisely,

$$\begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{-i\theta_\alpha} \end{bmatrix} \begin{bmatrix} A & Be^{i\theta_b} \\ Be^{i\theta_b} & D \end{bmatrix} \begin{bmatrix} e^{-i\theta_\alpha} & 0 \\ 0 & e^{i\theta_\alpha} \end{bmatrix} = \begin{bmatrix} A & B \\ B & D \end{bmatrix}, \quad (6.4)$$

where

$$\alpha = -\frac{\theta_b}{2},$$

followed by a real 2×2 SVD (2.13) to complete the diagonalization of the 2×2 Hermitian matrix.

The eigenvalue decomposition of a 2×2 Hermitian matrix can be computed through the use of a Q transformation (4.40), where

$$\alpha = \frac{\theta_b}{2}, \quad \beta = \frac{-\theta_b}{2}, \quad \gamma = \frac{-\theta_b}{2} \text{ and } \delta = \frac{\theta_b}{2},$$

$$\tan(\theta_\phi - \theta_\psi) = -\left(\frac{B}{D - A}\right),$$

and

$$\tan(\theta_\phi + \theta_\psi) = -\left(\frac{B}{D + A}\right),$$

since (6.4) can be expressed as a combination of an \mathcal{R} and a \mathcal{C} transformation.

The scheme for the computation of the EVD on the architecture of Figure 5.8 is similar to the computation of the \mathcal{Q} transformation for Step - II of the SVD scheme (Table 5.5). A data exchange timing similar to Figure 6.5 is then required.

6.4 Performance of the Systolic SVD Arrays

In comparing the relative performance of the Brent-Luk-VanLoan systolic array and the complex SVD array, the convergence of the SVD-Jacobi method and the time to compute a sweep of the “parallel ordering” are the determining factors. For reasons that will be made clear in the next chapter, it may be assumed that the convergence behavior of the SVD-Jacobi method on the Brent-Luk-VanLoan systolic array with real data elements and the complex SVD array with complex data elements is similar.

A measure of the computational speed of the two arrays is the time required for processors on the main diagonal to start processing new data after completing a diagonalization. These times for the different SVD arrays are tabulated in Table 6.4, where Δ_{steps} refers to the number of computation steps before the main diagonal starts processing data again.

Array	Δ_{steps}	# CORDIC Cycles (T_C)	Rel. Speed
Real SVD	3	9.75 T_C	1.0 T_{RSVD}
Complex SVD (Direct)	3	33.0 T_C	3.38 T_{RSVD}
Complex SVD (Staggered)	4	22.0 T_C	2.26 T_{RSVD}
Herm. EVD (Direct)	3	16.5 T_C	1.69 T_{RSVD}

Table 6.4: Relative Performance of Matrix Decomposition Arrays

The data in Table 6.4 needs explanation. Systolic arrays are characterized by synchronized multiprocessing. All processors active during a computation step must

synchronize at completion before initiating the next step. This implies that processors that finish early idle until the slowest active processor finishes computation. Balanced computation and I/O is thus important.

Δ	Real SVD	Com. SVD (D)	Com. SVD (S)	Herm. EVD (D)
1	$3.25 T_C$	$11.0 T_C$	$5.5 T_C$	$5.5 T_C$
2	$3.25 T_C$	$11.0 T_C$	$5.5 T_C$	$5.5 T_C$
3	$3.25 T_C$	$11.0 T_C$	$5.5 T_C$	$5.5 T_C$
4	-	-	$5.5 T_C$	-

Table 6.5: Step-wise Execution Times of the Matrix Decomposition Arrays

The fastest computation of the SVD for a real 2×2 matrix using CORDIC was shown to require $3.25 T_C$ (Table 3.3). From the discussion of the area/time complexity of the complex CORDIC SVD processor in § 5.4, we know that a Q transformation requires $5.5 T_C$. Again, both the times mentioned above include the time to generate the respective rotation parameters involved.

In all of the systolic arrays listed in Table 6.4, generation of the rotational parameters is done only along the main diagonal. For the direct scheme, the processors along any diagonal are active once every three computation steps while in the staggered schemes the same processors are active twice every four steps. Table 6.5 shows the stepwise split-up of the times shown in Table 6.4.

The last column of Table 6.4 shows the time to complete a sweep of the “parallel ordering” relative to the real SVD array. The staggering of computations in the complex SVD array allows

$$\frac{33.0}{22.0} = 1.5,$$

i.e., a 50% speedup over the direct scheme. Also, despite the involved nature of computations needed by the complex SVD scheme, with the staggering of computations, it requires less than three times the computation time for the real SVD array.

6.5 Wavefront vs. Systolic Computation

A drawback of systolic arrays is that data movement is governed by global timing. Synchronization is achieved by inserting extra delays to ensure correct timing. This problem is compounded in the case of large arrays, where the burden of synchronizing the whole array becomes infeasible.

A solution is to exploit the locality of control-flow in addition to the data-flow locality inherent in most systolic algorithms. In doing so, array processing becomes self-timed and data-driven. The central idea is to trade correct timing for correct sequencing. Systolic array processors which exhibit the data-driven property are termed *wavefront* array processors. The principal advantage of wavefront arrays over systolic arrays is speed.

In the context of the SVD arrays, wavefront type processing can improve speed up by allowing further overlap of the computation steps. Since the rotation parameters are obtained at the main diagonal processors before they are applied to the data, they may be transmitted to the immediately off-diagonal processors as soon as they become available.

The time to apply a real two-sided rotation is $2.25 T_C$ while the time to generate and/or apply the inner unitary transformation of the Q transformation is $2.25 T_C$ with additional time to perform some pre-processing (Table 5.3). The latter result is due to the fact that the arguments of the complex data elements are obtained as a by-product of the polar transformation sub-step (Table 5.2) in the application of the unitary transformation. It is therefore possible to reduce the off-diagonal processing times by up to $1.0 T_C$.

Also, by overlapping the application of the Q transformation across diagonal processors, the additional pipelining effect will improve processor utilization and reduce

idle time. Thus, there is definitely some performance advantage to be gained by transmitting the rotation parameters sooner than in the systolic scheme and adopting a wavefront type of scheduling. However, extra synchronization is necessary to ensure proper sequencing of computations.

In estimating the relative speed of the real and complex SVD arrays, it was assumed that the rate of convergence for the SVD-Jacobi scheme does not depend on the nature of the matrix data (real or complex). The next chapter discusses the computer simulation of the systolic algorithms. The simulation is then used to study the convergence behavior of the SVD-Jacobi method and to verify the correctness of the systolic algorithms.

Chapter 7

Simulation of the Complex SVD Array

In the previous chapters, a systolic scheme for the SVD of an arbitrary complex matrix was developed. The iterative nature requires simulation of the algorithm on a computer so that the convergence behavior may be studied for various matrix/array sizes. Since a parallel algorithm is to be simulated, a computer with suitable multiprocessing capability should prove to be advantageous.

7.1 The Connection Machine

The distinguishing features of systolic arrays map well onto the SIMD [31] paradigm of computation. Although there are significant differences between systolic arrays and SIMD computers [25] the architectural similarities provide excellent hardware support for the simulation of systolic arrays. The Connection Machine, a SIMD computer which supports the data parallel model of computation with good inter-processor communication capability, was chosen for the simulation of these arrays.

The Connection Machine employs the data parallel model (SIMD) of computation. Each instruction is executed by all processors in parallel. However, each processor may be selectively activated or deactivated. Variations in computations across the processor elements (PEs) is accommodated by selective activation of processors. Memory on the Connection Machine is distributed with each processor possessing local memory. The Connection Machine communication primitives allow transfer of

data from one processor's memory to another. Parallel transfer of data in regular patterns is a very useful feature of the Connection Machine hardware.

The programming environment of the Connection Machine supports parallel versions of several common high level languages. The operations on the Connection Machine hardware are specified using Paris (PARallel Instruction Set). The Connection Machine hardware essentially operates as a co-processor to a host or front-end computer. Currently VAX and Sun architectures can serve as front-ends to the Connection Machine. For Lisp programming, a Symbolics Lisp machine can serve as a front-end [76].

Most of the simulation was written in C/Paris [75], a front-end C compiler with a Paris interface to control the Connection Machine hardware. The use of C/Paris for most of the simulation improved code efficiency and performance due to the low level control of the Connection Machine hardware possible through Paris. The control code is written in C* (version 6.0) [77], a parallel C language compiler. The syntax of C* is quite powerful while preserving ease of notation.

In [19], a simulation of the Brent-Luk-VanLoan real SVD array using the RPPT (Rice Parallel Processing Testbed) is described. Each PE in the array was modeled both as a simple instruction set processor and as an MC68020 processor. Processor arrays up to matrix dimensions of 30×30 were simulated using 4-way and 8-way meshes.

7.2 Simulation Model

The logical unit of simulation is the PE. Each PE is represented in hardware by a processor on the Connection Machine (Figure 7.1). All PEs in a systolic array are identical. They perform similar computations with minor variations depending on

the location in the index set of processors. In the simulation of the complex SVD array, each processor is modeled as a set of registers. The simulation code allows the specification of the array configuration and register allocation per PE.

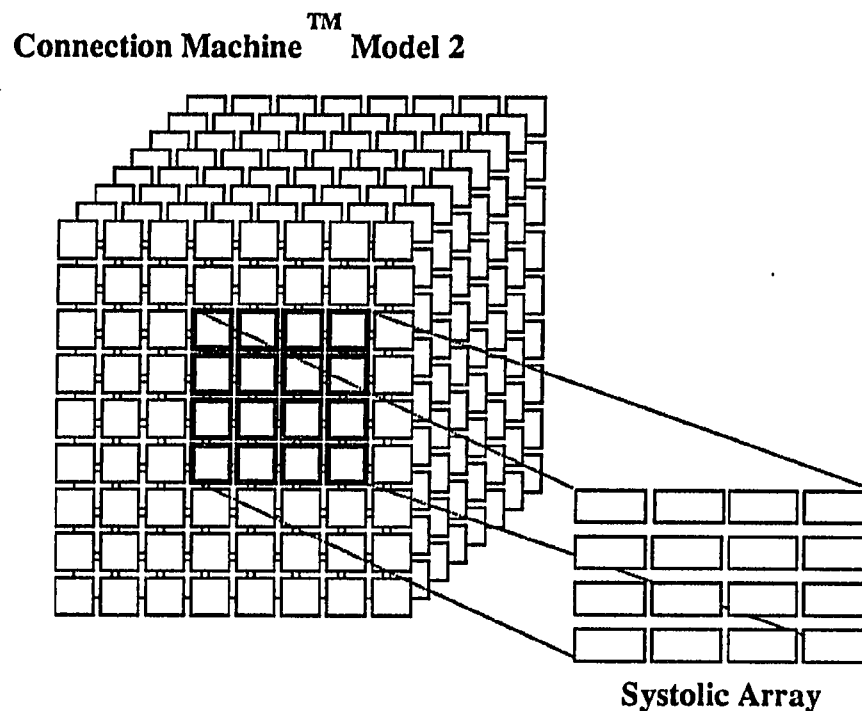


Figure 7.1: Mapping of the Systolic Array onto the CM2

Registers may store fixed or floating point data of user defined bit precision. This is possible due to the special bit-addressable memory and bit-serial math capability of the Connection Machine processor. Interaction of processors is through the exchange of data stored in these registers. As most systolic designs tend to have local interconnections, near-neighbor communication is predominant in systolic arrays. Random communication patterns are rare. However, both forms of communication are supported in the simulation model.

The state of a processor at any time during computation is characterized by the data in its registers. A *snapshot* is the cumulative state of all the processors in the array at any time during simulation. *Snapshots* capture significant details of array activity and are extremely useful in the verification of the design and the correctness of the algorithms.

Array @ t = 00			
1.000 + 2.000 i	3.000 + 4.000 i	5.000 + 6.000 i	7.000 + 8.000 i
2.000 + 4.000 i	6.000 + 8.000 i	6.000 + 4.000 i	2.000 + 4.000 i
8.000 + 7.000 i	6.000 + 5.000 i	4.000 + 3.000 i	2.000 + 1.000 i
4.000 + 2.000 i	4.000 + 6.000 i	8.000 + 6.000 i	4.000 + 2.000 i
Array @ t = 1 sweep			
2.726 + 0.000 i	0.000 + 0.000 i	1.796 + 0.272 i	-2.634 - 0.399 i
-0.000 - 0.000 i	8.036 - 0.000 i	-1.865 - 1.903 i	-1.271 - 1.298 i
-1.553 + 1.288 i	0.766 - 0.635 i	4.877 + 0.000 i	0.000 + 0.000 i
0.925 - 0.368 i	1.875 - 0.746 i	0.000 + 0.000 i	25.792 + 0.000 i
Array @ t = 2 sweeps			
1.302 - 0.000 i	-0.000 - 0.000 i	0.008 - 0.000 i	-0.367 + 0.000 i
-0.000 + 0.000 i	8.896 + 0.000 i	0.322 - 0.000 i	0.007 - 0.000 i
0.027 + 0.000 i	-0.438 - 0.000 i	5.722 + 0.000 i	0.000 + 0.000 i
-0.376 - 0.000 i	-0.023 - 0.000 i	-0.000 - 0.000 i	26.035 - 0.000 i
Array @ t = 3 sweeps			
1.307 + 0.000 i	-0.000 + 0.000 i	0.000 + 0.000 i	-0.030 + 0.000 i
0.000 + 0.000 i	8.941 + 0.000 i	0.000 + 0.000 i	0.000 + 0.000 i
0.000 + 0.000 i	-0.000 + 0.000 i	5.678 + 0.000 i	0.000 + 0.000 i
-0.027 + 0.000 i	-0.000 + 0.000 i	0.000 + 0.000 i	26.040 + 0.000 i

Table 7.1: Time-Stamped Snapshots of a 2×2 Processor Array

Table 7.1 shows the state of a complex SVD 2×2 processor array at the completion of each of three sweeps required during computation of the SVD. The contents of the

array show the diagonal elements in the processors along the main diagonal converging to the singular values while the data elements in the off-diagonal processors converge to zero. The numbers beside the title in each *snapshot* indicates the time-stamp.

7.3 Simulation Experiments and Results

The principal aims for simulating the complex SVD array are verification of the data interchange timing and observance of the convergence behavior in terms of the number of sweeps required for a given matrix (array) size. A similar study was also performed by Brent, Luk and VanLoan [6] to observe the convergence rate for the real SVD using “parallel ordering” and the two-sided rotation scheme for diagonalizing a real 2×2 matrix (2.6).

In their study, Brent, Luk and VanLoan used random $n \times n$ matrices M , whose elements were uniformly and independently distributed in $[-1, 1]$. The stopping criterion used was that $off(M)$ as computed in (2.3) was reduced to 10^{-12} times its original value. A similar experiment was performed for the two-step diagonalization scheme (§ 4.6) with the systolic scheme described in § 6.2.

Table 7.2 shows the average and maximum number of sweeps required for the convergence of the two-step diagonalization scheme. A comparison of the number of sweeps required for convergence of the real SVD scheme with similar matrix (array) sizes is shown in Figure 7.2. The singular values obtained from the simulation runs for a few sample cases per matrix size, were validated using LINPACK routines.

The convergence behavior of the SVD-Jacobi method with “parallel ordering” for real data matrices is given in [6]. Figure 7.2 includes a plot of logarithms to the base 2, to validate the conjecture that $O(\log n)$ sweeps are required for convergence.

n	Trials	Avg. # Sweeps	Max. # Sweeps
4	100	4.650833	5.250000
6	100	5.484500	5.950000
8	100	6.118214	7.035714
10	100	6.381667	7.361111
12	100	6.681818	7.454545
14	100	6.904231	7.826923
16	100	7.065500	8.033334
18	100	7.078970	7.897059
20	100	7.218553	8.263158
40	10	8.115385	8.459784
60	10	8.411017	8.786517
80	10	9.006330	9.173082
100	1	9.128788	-

Table 7.2: Complex CORDIC SVD Convergence Behavior

The number of sweeps required for the convergence of the complex SVD scheme is greater than that for the real SVD, but the convergence behavior is identical. Also, since the time required to complete a sweep in the complex SVD array is 2.26 times greater than the real SVD array (Table 6.4), the overall time for computation is less than three times that for the real SVD array.

7.4 Terminating Computations on the Complex SVD Array

In the previous chapter, the issue of terminating computations on the complex SVD array was not addressed. However, from the simulation results it is clear that for most matrix dimensions in practice, we may assume a constant number of sweeps to guarantee convergence. In doing so, the need for global communication of results is not required to ensure the satisfaction of suitable threshold criteria.

The simplest scheme for termination then, is to perform a pre-determined number of sweeps S (from Figure 7.2, $S = 10$ seems reasonable). This implies that processor

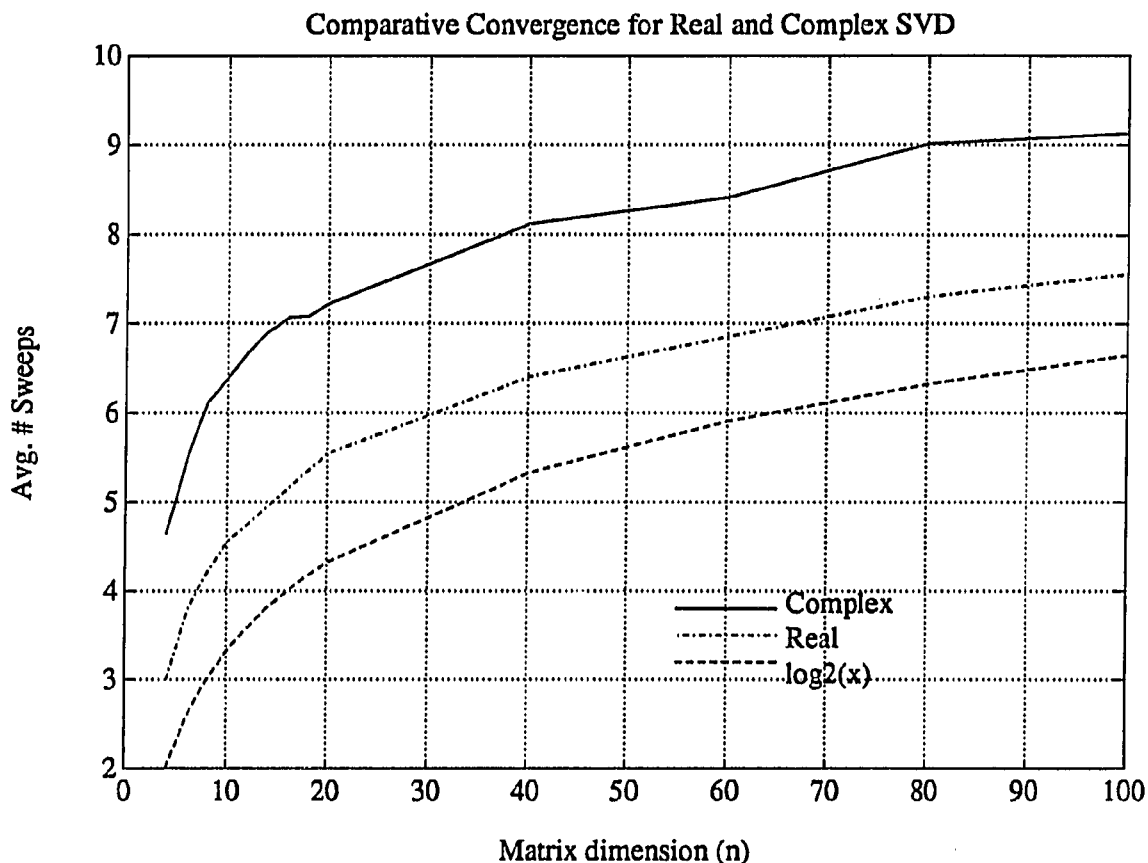


Figure 7.2: Comparative Convergence of Real and Complex SVD Schemes

P_{ij} ceases computation at time step T_{ij} where

$$T_{ij} = \Delta_{steps}S(n-1) + \Delta_{ij} + \Delta_{steps},$$

since a sweep takes $\Delta_{steps}(n-1)$ time steps. The value of Δ_{steps} is either 3 or 4 depending on the staggering of computations (Table 6.4) and $\Delta_{ij} = |i-j|$ represents the distance of the processor from the main diagonal.

A more sophisticated criterion for termination is suggested in [6] for the real SVD array. Termination is made contingent on the performance of no non-trivial rotations during the previous sweep. A similar criterion can be extended to the complex SVD

array where non-trivial Q transformations are detected. However, the scheme requires communication along the diagonal, that can be done in $n/2$ time steps.

This completes the discussion on the various aspects of the complex SVD array. In the next chapter, a summary of the thesis is presented along with directions for future work.

Chapter 8

Conclusions

The principal contribution of this thesis is the development of a systolic algorithm and, a hardware and performance efficient architecture implementable in VLSI, for computing the Singular Value Decomposition of an arbitrary complex matrix. In concluding the thesis, the discussion in the previous chapters is recapitulated in brief.

8.1 Summary

The Singular Value Decomposition is an important matrix factorization used extensively in engineering applications. It is particularly useful in the context of signal processing, image processing and robotics applications. Real-time data processing necessitates the use of special-purpose hardware to sustain the computational speed required.

Systolic array architectures with inherent parallelism and synchronized multiprocessing capabilities are well suited to meet the challenge of real-time data processing. A variety of systolic arrays have been proposed in the literature for computing the SVD of a matrix. Most of the systolic algorithms proposed in this context concern matrices with real data elements. However, complex matrices do occur in engineering practice; especially adaptive beam-forming algorithms in signal processing applications are known to require the SVD factorization.

The Jacobi-type methods, which form the basis for most of the systolic algorithms

are especially amenable to parallel processing. Several methods have been suggested in the literature to compute the SVD of an arbitrary complex matrix using Jacobi-type methods. The methods are either limited in their applicability or implementation in special-purpose hardware, or do not efficiently adapt to systolic processing.

In this thesis, a novel hardware oriented two-step diagonalization scheme for the SVD of a complex 2×2 matrix, the basic step in a Jacobi-type procedure for the computation of the SVD, was presented. Each step in the scheme was a two-sided unitary transformation (\mathcal{Q} transformation), designed to be efficiently implementable in hardware using CORDIC. A review of the CORDIC algorithms was also presented to illustrate the applicability, efficiency of hardware and the performance advantage possible.

An expandable array of 2×2 processors due to Brent, Luk and VanLoan is the best known systolic array for the SVD of real matrices. The array uses a Jacobi-type method with a "parallel ordering" and requires $O(n^2)$ processors to compute the SVD of an $n \times n$ matrix. A systolic array similar in structure to the Brent-Luk-VanLoan systolic array, with an enhanced scheduling and data exchange algorithm designed to efficiently implement the two-step diagonalization scheme was proposed for the solution of the complex SVD problem. Also, the \mathcal{Q} transformation was shown to be applicable to the symmetric eigenvalue decomposition of a Hermitian matrix using an array similar to the complex SVD array.

The behavior of a processor in the complex SVD array, illustrating the systolic computation, was detailed. An architecture for the complex 2×2 processor, exploiting the parallelism in the CORDIC implementation of the two-step diagonalization scheme, was presented. A hardware scheduling algorithm for the architecture, demonstrating this parallelism was also described.

The systolic algorithm was simulated on the Connection Machine to verify correctness and observe the convergence behavior of the algorithm. The complex SVD algorithm required more sweeps than the real SVD scheme but the convergence behavior was identical in that $O(\log n)$ sweeps are required for convergence given an $n \times n$ input matrix. Termination criteria and schemes based on the simulation results were also suggested.

In spite of the involved nature of computations in diagonalizing a complex 2×2 matrix, the time for completing a sweep in the complex CORDIC SVD array is less than three times that for a CORDIC based implementation of the real SVD array.

8.2 Future Work

There are several issues regarding the complex SVD array which were not addressed in this thesis. The accumulation of the right and left singular vectors can be easily accommodated during the idle time of the processors. The handling of undersized and oversized matrix dimensions was not discussed in the thesis. However, several schemes for the real case as presented in [6] can be extended to the complex SVD array.

VLSI implementation of the architecture proposed for the complex 2×2 processor and integration of both the real and complex SVD schemes on the same processor without additional hardware are achievable goals. Also, of particular interest are fault-tolerance issues and reconfiguration algorithms for load sharing and rerouting of data around faulty processors.

Bibliography

- [1] H. M. Ahmed. *Signal Processing Algorithms and Architectures*. PhD thesis, Dept. of Electrical Engineering, Stanford Univ., Stanford, CA, June 1982.
- [2] H. M. Ahmed, J. M. Delosme, and M. Morf. Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing. *IEEE Computer*, 15(1):65–82, January 1982.
- [3] H. C. Andrews and C. L. Patterson. Singular Value Decompositions and Digital Image Processing. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-24(1):26–53, February 1976.
- [4] A. Bojanczyk, R. P. Brent, and H. T. Kung. Numerically Stable Solution of Dense Systems of Linear Equations Using Mesh-Connected Processors. *SIAM Journal of Scientific and Statistical Computing*, 5(1):95–104, March 1984.
- [5] R. P. Brent and F. T. Luk. The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays. *SIAM Journal of Scientific and Statistical Computing*, 6(1):69–84, January 1985.
- [6] R. P. Brent, F. T. Luk, and C. F. Van Loan. Computation of the Singular Value Decomposition Using Mesh-Connected Processors. *Journal of VLSI and Computer Systems*, 1(3):242–270, 1985.

- [7] C. Bridge, P. Fisher, and R. Reynolds. Asynchronous Arithmetic Algorithms for Data-Driven Machines. *IEEE 5th Symposium on Computer Arithmetic*, pages 56–62, May 1981.
- [8] F. Briggs and K. Hwang. *Computer Architectures and Parallel Processing*. McGraw Hill, 1984.
- [9] A. Bunse-Gerstner. Singular Value Decompositions of Complex Symmetric Matrices. *J. Comp. Applic. Math.*, 21:41–54, 1988.
- [10] P. A. Businger and G. H. Golub. Algorithm 358: Singular Value Decomposition of the Complex Matrix. *Comm. ACM*, 12:564–565, 1969.
- [11] J. R. Cavallaro. *VLSI CORDIC Processor Architectures for the Singular Value Decomposition*. PhD thesis, School of Electrical Engineering, Ithaca, NY, August 1988.
- [12] J. R. Cavallaro and A. C. Elster. A CORDIC Processor Array for the SVD of a Complex Matrix. In R.J. Vaccaro, editor, *SVD and Signal Processing II (Algorithms, Analysis and Applications)*, pages 227–239. Elsevier, New York, 1991.
- [13] J. R. Cavallaro and F. T. Luk. Architectures for a CORDIC SVD Processor. *Proc. SPIE Real-Time Signal Processing*, 698(IX):45–53, August 1986.
- [14] J. R. Cavallaro and F. T. Luk. CORDIC Arithmetic for an SVD Processor. *Journal of Parallel and Distributed Computing*, 5(3):271–290, June 1988.
- [15] T. F. Chan. An Improved Algorithm for Computing the Singular Value Decomposition. *ACM Trans. Math. Soft.*, 8:72–83, 1982.

- [16] D. S. Cochran. Algorithms and Accuracy in the HP-35. *Hewlett-Packard J.*, pages 10–11, June 1972.
- [17] T. F. Coleman and C. F. Van Loan. *Handbook for Matrix Computations*. SIAM, Philadelphia, PA, 1988.
- [18] D. Daggett. Decimal-Binary Conversions in CORDIC. *IRE Transactions on Electronic Computers*, EC-8(3):335–339, Sept. 1959.
- [19] W. P. Dawkins. Efficient Simulation of Simple Instruction Set Array Processors. Master's thesis, Rice University, Department of Electrical and Computer Engineering, November 1989.
- [20] J. M. Delosme. CORDIC Algorithms: Theory and Extensions. *Proc. SPIE Advanced Algorithms and Architectures for Signal Processing*, 1152(IV):131–145, August 1989.
- [21] J. M. Delosme. VLSI Implementation of Rotations in Pseudo-Euclidean Spaces. *IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2:927–930, April 1983.
- [22] J. M. Delosme. The Matrix Exponential Approach to Elementary Operations. *Proc. SPIE Advanced Algorithms and Architectures for Signal Processing*, 696(I):188–195, August 1986.
- [23] J. M. Delosme. A Processor for Two-Dimensional Symmetric Eigenvalue and Singular Value Arrays. *IEEE 21th Asilomar Conf. on Circuits, Systems, and Computers*, pages 217–221, November 1987.
- [24] A. M. Despain. Fourier Transform Computers Using CORDIC Iterations. *IEEE Transactions on Computers*, C-23(10):993–1001, October 1974.

- [25] P. M. Dew and L. J. Manning. Comparison of Systolic and SIMD Architectures for Computer Vision Computation. *Proc. Inter. Workshop on Systolic Arrays, University of Oxford, July 1986.*
- [26] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. Chapter 11: The Singular Value Decomposition. In *Linpac Users' Guide*, pages 11.1–11.23. SIAM, Philadelphia, PA, 1979.
- [27] M. D. Ercegovac and T. Lang. Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD. *IEEE Trans. Computers*, 39(6):725–740, June 1990.
- [28] L. M. Ewerbring and F. T. Luk. Computing the Singular Value Decomposition on the Connection Machine. *IEEE Trans. on Computers*, 39(1):152–155, January 1990.
- [29] A. M. Finn. *Systolic Array Computation of the Singular Value Decomposition*. PhD thesis, School of Electrical Engineering, Cornell Univ., Ithaca, NY, May 1983.
- [30] A. M. Finn, F. T. Luk, and C. Pottle. Systolic Array Computation of the Singular Value Decomposition. *Proc. SPIE. Vol. 341. Real-Time Signal Processing V*, pages 34–43, 1982.
- [31] M. J. Flynn. Very High Speed Computing Systems. *Proceedings of the IEEE*, Vol. 54:1901–1909, 1966.
- [32] G. E. Forsythe and P. Henrici. The Cyclic Jacobi Method for Computing the Principal Values of a Complex Matrix. *Transactions of the American Mathematical Society*, 94(1):1–23, January 1960.

- [33] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler. *Matrix Eigensystem Routines - EISPACK Guide Extension*. Springer-Verlag, Berlin, 1977.
- [34] W. M. Gentleman and H. T. Kung. Matrix Triangularization by Systolic Arrays. *Proc. SPIE Real-Time Signal Processing IV*, 298:19-26, August 1981.
- [35] G. H. Golub and W. Kahan. Calculating the Singular Values and Pseudo-inverse of a Matrix. *J. SIAM Ser. B: Numer. Anal.* 2, pages 205-224, 1965.
- [36] G. H. Golub and C. F. Van Loan. *Matrix Computations, Second Edition*. Johns Hopkins Univ. Press, Baltimore, MD, 1989.
- [37] G. H. Golub and C. Reinsch. Singular Value Decomposition and Least-squares Solutions. In J. H. Wilkinson and C. Reinsch, editors, *Handbook for Automatic Computation. Vol. 2 (Linear Algebra)*, pages 134-151. Springer-Verlag, New York, 1971.
- [38] J. Greenstadt. A Method for Finding Roots of Arbitrary Matrices. *Math. Tables Aids Comput.*, 9:47-52, 1955.
- [39] E. R. Hansen. On Cyclic Jacobi Methods. *J. Soc. Indust. Appl. Math*, 11:448-459, 1963.
- [40] G. L. Haviland and A. A. Tuszynski. A CORDIC Arithmetic Processor Chip. *IEEE Trans. Computers*, C-29(2):68-79, Feb. 1980.
- [41] D. E. Heller and I. Ipsen. Systolic Networks for Orthogonal Decompositions. *SIAM J. Sci. Statist. Comput.*, 4:261-269, 1982.
- [42] N. D. Hemkumar, K. Kota, and J. R. Cavallaro. CAPE - VLSI Implementation of a Systolic Processor Array: Architecture, Design and Testing. *Proceedings of the*

Ninth Biennial University/Government/Industry Microelectronics Symposium,
June 1991 (to appear).

- [43] M. R. Hestenes. Inversion of Matrices by Biorthogonalization and Related Results. *J. Soc. Indust. Appl. Math*, 6:51–90, 1958.
- [44] S. Hitotumatu. Complex Arithmetic through CORDIC. *Kodai Math. Sem. Rep.*, 26:176–186, 1975.
- [45] I. Ipsen. Singular Value Decomposition with Systolic Arrays. *Proc. SPIE Real-Time Signal Processing*, 495(VII):13–21, August 1984.
- [46] D. H. Johnson. The Application of Spectral Estimation Methods to Bearing Estimation Problems. *Proceedings of the IEEE*, 70(9):1018–1028, September 1982.
- [47] L. Johnsson. A Computational Array for the QR-method. *Proc. 1982 Conf. on Advanced Research in VLSI*, pages 123–129, MIT, Cambridge, MA (1982).
- [48] S. L. Johnsson and V. Krishnaswamy. Floating-Point CORDIC. Technical Report YALEU/DCS/RR-473, Dept. of Computer Science, Yale Univ., New Haven, CT, April 1986.
- [49] E. G. Kogbetliantz. Solution of Linear Equations by Diagonalization of Coefficients Matrix. *Quarterly of Applied Mathematics*, 14(2):123–132, 1955.
- [50] K. Kota. Architectural, Numerical and Implementation Issues in the VLSI Design of an Integrated CORDIC SVD Processor. Master's thesis, Rice University, Department of Electrical and Computer Engineering, May 1991.
- [51] H. T. Kung. Why Systolic Architectures? *IEEE Computer*, 15(1):37–46, January 1982.

- [52] H. T. Kung and C. E. Leiserson. Systolic Arrays (for VLSI). *Sparse Matrix Symposium*, pages 256–282, SIAM, 1978.
- [53] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, 1987.
- [54] S. Y. Kung. On Supercomputing with Systolic/Wavefront Array Processors. *IEEE Proceedings*, 72(7):867–884, July 1984.
- [55] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer, and D. V. Bhaskar Rao. Wavefront Array Processor: Language, Architecture, and Applications. *IEEE Trans. on Computers*, C-31(11):1054–1066, November 1982.
- [56] M. Lotkin. Characteristic Values of Arbitrary Matrices. *Quarterly of Applied Mathematics*, 14(3):267–275, 1956.
- [57] F. T. Luk. A Triangular Processor Array for Computing Singular Values. *Journal of Linear Algebra and Its Applications*, 77:259–273, 1986.
- [58] F. T. Luk. A Rotation Method for Computing the QR-Decomposition. *SIAM Journal of Scientific and Statistical Computing*, 7(2):452–459, April 1986.
- [59] F. T. Luk. Computing the Singular Value Decomposition on the ILLIAC IV. *ACM Transactions on Mathematical Software*, 6(4):524–539, December 1980.
- [60] F. T. Luk and H. Park. A Proof of Convergence for Two Parallel Jacobi SVD Algorithms. *IEEE Trans. on Computers*, 38(6):806–811, June 1989.
- [61] J. E. Meggitt. Pseudo Division and Pseudo Multiplication Processes. *IBM J.*, pages 210–226, April 1962.

- [62] J. Moreno. Analysis of Alternatives for a Singular Value Decomposition Processor. Master's thesis, Computer Science Department, Univ. of California at Los Angeles, Los Angeles, CA, October 1985.
- [63] J. M. Muller. Methodologies de Calcul des Fonctions Elementaires. Technical report, Institute National Polytechnique de Grenoble, December 1985.
- [64] C. M. Rader. Wafer-Scale Systolic Array for Adaptive Antenna Processing. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pages 2069–2071, April 1988.
- [65] A. H. Sameh. Solving the Linear Least Squares Problem on a Linear Array of Processors. *Proc. Purdue Workshop on Algorithmically-specialized Computer Organizations*, 1982.
- [66] A. H. Sameh. On Jacobi and Jacobi-Like Algorithms for a Parallel Computer. *Mathematics of Computation*, 25(115):579–590, July 1971.
- [67] R. Schreiber. On the Systolic Arrays of Brent, Luk, and Van Loan. *Proc. SPIE Real-Time Signal Processing*, 431(VI):72–76, August 1983.
- [68] R. Schreiber. A Systolic Architecture for Singular Value Decomposition. *Proc. 1st Internat. Coll. on Vector and Parallel Computing in Scientific Applications*, Paris, France, March 1983.
- [69] Y. S. Shim and Z. H. Cho. SVD Pseudo Inversion Image Reconstruction. *IEEE Trans. Acoustics, Speech, and Signal Processing*, pages 904–909, August 1981.
- [70] L. H. Sibul. Application of Singular Value Decomposition to Adaptive Beamforming. *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 2:33.11.1–33.11.4, March 1984.

- [71] L. H. Sibul and A. L. Fogelsanger. Application of Coordinate Rotation Algorithm to Singular Value Decomposition. *IEEE Int. Symp. Circuits and Systems*, pages 821–824, 1984.
- [72] J. M. Speiser and H. J. Whitehouse. Parallel Processing Algorithms and Architectures for Real-Time Signal Processing. *Proc. SPIE Real-Time Signal Processing*, 298(IV):2–9, August 1981.
- [73] J. M. Speiser and H. J. Whitehouse. A Review of Signal Processing with Systolic Arrays. *Proc. SPIE Real-Time Signal Processing*, 431(VI):2–6, August 1983.
- [74] G. W. Stewart. A Jacobi-Like Algorithm for Computing the Schur Decomposition of a Nonhermitian Matrix. *SIAM Journal of Scientific and Statistical Computing*, 6(4):853–864, October 1985.
- [75] Thinking Machines. *Connection Machine, Introduction to Programming in C/Paris*. Thinking Machines Corporation, Cambridge MA, 1990.
- [76] Thinking Machines. *Connection Machine, Model CM-2 Technical Summary*. Thinking Machines Corporation, Cambridge MA, 1990.
- [77] Thinking Machines. *Connection Machine, Programming in C**. Thinking Machines Corporation, Cambridge MA, 1990.
- [78] J. D. Ullman. *Computational Aspects of VLSI*. Computer Science Press, Rockville, MD, 1984.
- [79] A. J. Van der Veen and E. F. Deprettere. A Parallel VLSI Direction Finding Algorithm. *Proc. SPIE Advanced Algorithms and Architectures for Signal Processing*, 975(III):289–299, August 1988.

- [80] J. Volder. The CORDIC Trigonometric Computing Technique. *IRE Trans. Electronic Computers*, EC-8(3):330–334, Sept. 1959.
- [81] J. S. Walther. A Unified Algorithm for Elementary Functions. *AFIPS Spring Joint Computer Conf.*, pages 379–385, 1971.
- [82] J. H. Wilkinson. Note on the Quadratic Convergence of the Cyclic Jacobi Process. *Numer. Math.*, 4:296–300, 1962.
- [83] B. Yang and J. F. Böhme. Reducing the Computations of the SVD Array given by Brent and Luk. *SPIE Advanced Algorithms and Architectures for Signal Processing IV*, 1152:92–102, 1989.