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.

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

Parallel Algorithms and Architectures for near-far resistant CDMA acquisition

by

Kishore Kota

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

[Signatures with names]

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

Dr. Don H. Johnson
Professor of Electrical and Computer Engineering and of Statistics

Dr. Behnaam Aazhang
Professor of Electrical and Computer Engineering

Dr. Danny C. Sorensen
Professor of Computational and Applied Mathematics

Houston, Texas
May, 1996
Parallel Algorithms and Architectures for near-far resistant CDMA acquisition

Kishore Kota

Abstract
Subspace-based algorithms are a class of algorithms for estimation problems in array signal processing and more recently near-far resistant Code Division Multiple Access (CDMA) acquisition problems. Subspace-based algorithms are based on estimating signal or noise subspaces from the received data vectors and then performing some form of optimization to estimate the desired parameters. This thesis presents two new parallel algorithms applicable in the estimation of signal or noise subspaces from the received data vectors.

The first algorithm is a pipelined SVD algorithm which allows pipelining of multiple independent singular value decomposition (SVD) problems on a single processor array. The resultant algorithm uses the flexibility provided by the Jacobi algorithm by defining a new parallel ordering to result in a simple uniform array in which all communication including the initial load and the final unload operations are pipelined.

The second algorithm described in this thesis is a sliding window SVD updating algorithm where the signal or noise subspace is updated whenever a new observation vector is received by applying a fixed-length window over the data. An important result shown in this thesis is that an important property of downdating problems, known as relational stability, extends to a hybrid fixed-point and floating-point algorithm. By performing most of the computation in fixed-point, significant gains in
implementation complexity may be realized. This thesis also extends an SVD algorithm originally developed for an exponential window to the sliding window problem through the use of carefully implemented hyperbolic rotations.

A second major topic of this thesis is demonstrating the applicability of a number of algorithms and architectures developed in the array signal processing area to the near-far resistant acquisition problem in CDMA communication systems. This thesis classifies the available literature in this area and performs a qualitative analysis of the different algorithms from the view of applicability towards the CDMA problem.

The third contribution of this thesis is a unified parallel architecture to implement the backend portions of the CDMA problem. Through the combination of a matrix vector multiplication array, an inner product computation array and an array of processors to perform second-order or third-order polynomial optimization a parallel architecture to implement the backend processing in the CDMA problem is realized.
Acknowledgments

When I look back at the six years I spent at Rice, I realize the extraordinary role played by so many different people in shaping my thoughts, my ideas and just about every aspect of me. For such a valuable gift, I will remain grateful to each and every one of them.

On the academic front, I am indebted to Dr. Cavallaro for providing me the opportunity to work in this exciting field. I am especially grateful to him for the latitude and freedom he granted me, allowing me to widen my knowledge base and learn many exciting things from diverse areas. I am also grateful to him for believing in me and encouraging me through the most trying times.

I am grateful to Dr. Aazhang, Dr. Johnson and Dr. Sorensen for agreeing to be a part of my thesis committee and for providing me the different perspectives I needed to complete this work.

Special thanks go to N.D.Hemkumar for constantly being my sounding board and asking me all the right questions. I will remain forever indebted to him for instilling in me an insatiable thirst for knowledge, and since I don’t accept anything without proof, by providing me with a living proof that this is not an impossible task. Thanks also go to Roomkey and Raghu for helping me out with this thesis and especially the defense.

Life at Rice was a lot more than academics. Maya, Ranga, Arati, Shivakumar, Vijji, Ram, Aparna, Sundeep, Brinda, Shalu, Roomkey, Vinod, Raji and Raghu all played vital roles in my own personal development. I wouldn’t be what I am nor
would my years at Rice be my most memorable without any of them. I already miss all those camping trips, those cookouts, potlucks, picnics, India-nites, late night recording sessions, DJing on KTRU, making “Magar Kyon” and the innumerable other activities I had the opportunity to be a part of.

Saving the best for the last, I thank my parents who played such a vital role all through my life by always placing my interests ahead of theirs. Thanks to my mother for being such a driving force and instilling in me a desire to always succeed. Thanks to my father for lighting the way and making it so easy for me to follow. Thanks to my brother Vijay for providing me the necessary competition to achieve my goals.

I hear the orchestra playing, so I have to cut this short. Thanks everyone for this wonderful chapter of my life.
Contents

Abstract .................................................. ii
Acknowledgments ........................................ iv
List of Tables ........................................... xii
List of Illustrations ..................................... xiii

1 Introduction ............................................ 1
  1.1 Motivation ........................................ 1
  1.2 Methodology ....................................... 4
  1.3 Overview of the thesis ............................. 8

2 Background ............................................. 10
  2.1 Techniques from numerical linear algebra ....... 10
      2.1.1 Givens rotation ............................ 11
      2.1.2 The QR decomposition ..................... 12
      2.1.3 QR updating ................................ 13
      2.1.4 Singular Value Decomposition (SVD) .... 14
      2.1.5 The Jacobi method to compute the SVD ... 15
  2.2 Signal Subspace based methods in signal processing .......... 20
      2.2.1 A mathematical model for observations ...... 20
      2.2.2 Maximum-likelihood estimation of the parameters .... 21
      2.2.3 Reducing complexity of the ML estimation formulation .... 23
PLEASE NOTE

Page(s) missing in number only; text follows.
Filmed as received.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>CDMA Communications</td>
<td>24</td>
</tr>
<tr>
<td>2.3.1</td>
<td>System model</td>
<td>24</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Transmitted Signal Model</td>
<td>25</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Channel Model</td>
<td>27</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Subspace-based algorithms</td>
<td>29</td>
</tr>
<tr>
<td>2.4</td>
<td>Systolic Algorithms and Architectures</td>
<td>38</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Matrix-vector multiplication array</td>
<td>38</td>
</tr>
<tr>
<td>2.4.2</td>
<td>QR updating array</td>
<td>40</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Brent-Luk-Van Loan SVD array</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>A Pipelined SVD Algorithm</td>
<td>48</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>48</td>
</tr>
<tr>
<td>3.2</td>
<td>Pipelining on linear arrays</td>
<td>51</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Throughput-Hardware tradeoffs</td>
<td>56</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Comparison with in-core methods</td>
<td>59</td>
</tr>
<tr>
<td>3.3</td>
<td>Convergence of the new ordering</td>
<td>61</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>Sliding Window Subspace Tracking</td>
<td>65</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>65</td>
</tr>
<tr>
<td>4.2</td>
<td>Background</td>
<td>65</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Cholesky Downdating Algorithms</td>
<td>65</td>
</tr>
<tr>
<td>4.2.2</td>
<td>The Hyperbolic Singular Value Decomposition</td>
<td>66</td>
</tr>
<tr>
<td>4.3</td>
<td>Sliding window subspace tracking algorithm</td>
<td>68</td>
</tr>
<tr>
<td>4.3.1</td>
<td>An algorithm for hyperbolic SVD updating</td>
<td>69</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Computation of hyperbolic rotations</td>
<td>72</td>
</tr>
</tbody>
</table>
5 Architectures for CDMA Acquisition - Frontend 90

5.1 Introduction ............................................ 90
5.2 Complexity of near-far resistant acquisition ............ 90
5.3 Strategies to improve eigen-analysis algorithms ........... 92
5.4 Formulation of the Frontend Problem .................... 93
5.4.1 Formation of an observation matrix .................. 95
5.5 Improved algorithms for the frontend .................... 98
5.5.1 A practical way to compute EVD of the correlation matrix 98
5.5.2 Numerically robust algorithms for the frontend .......... 99
5.5.3 A qualitative evaluation of front-end algorithms ........ 104
5.6 Quantitative analysis .................................. 114
5.6.1 Rectangular vs. Exponential window ................... 114
5.6.2 Group delay of the estimates ........................ 118
5.7 Summary ................................................. 119

6 Architectures for CDMA Acquisition - Backend 124

6.1 Introduction ............................................ 124
6.2 Deterministic MUSIC estimator .......................... 126
6.3 Maximum-likelihood MUSIC ............................. 132
6.4 Estimator for a Multipath Channel ....................... 135
6.5 Computational complexity of the backend processing .................. 138
6.6 Summary .............................................................................. 140

7 Summary and Future Work ..................................................... 142

7.1 Future Work ........................................................................... 144

7.1.1 Delay-locked Loop for acquisition ................................. 144

7.1.2 Relational stability of CORDIC-based hyperbolic downdating 145

Bibliography .............................................................................. 147
Tables

1.1 Sample parameters for a voice communication system .......................... 3

2.1 Parallel Ordering ............................................................................. 44

4.1 Results of Stewart’s test [61] performed in fixed-point/hybrid arithmetic 77

5.1 Computational complexity of near-far resistant acquisition ............... 91
5.2 Classification of frontend algorithms ............................................... 100
5.3 Comparison of a rectangular FIR transfer function and an IIR transfer function with one pole .......................................................... 118
5.4 Frontend Design Space ..................................................................... 120
5.5 Algorithm specific design options ..................................................... 121

6.1 Computational complexity of various frontend options ...................... 138
6.2 Complexity of acquisition of a single user ........................................ 139
6.3 Complexity of tracking a single user ................................................ 139
Illustrations

1.1 A CDMA cell .................................................. 3

2.1 Computing the QR decomposition using Givens rotations ............ 13
2.2 One sweep of the Jacobi algorithm to compute SVD ................. 16
2.3 A linear model of the observation vectors ............................. 20
2.4 A typical CDMA transmitter and receiver ......................... 25
2.5 CDMA RF Frontend ........................................... 26
2.6 Baseband pre-processing ....................................... 27
2.7 Example of the CDMA signal generated by one user ............... 28
2.8 Passband to baseband conversion ................................ 29
2.9 Systolic array for matrix multiplication ............................ 39
2.10 Gentleman-Kung systolic array [27] for QRD updating .......... 41
2.11 Internal cells of QRD updating array .......................... 42
2.12 The Brent-Luk-Van Loan SVD Array ............................ 44
2.13 Snapshots of Brent-Luk-Van Loan SVD Array .......................... 45

3.1 A completely unrolled pipelined SVD array ................. 52
3.2 Snapshots of the pipelined SVD array in operation ............ 53
3.3 An unrolled linear array ........................................ 57
4.1 A parallel architecture to implement sliding window SVD 70
4.2 Computing a $2 \times 2 \mathbf{H}$-orthogonal transformation 73
4.3 Computing a $2 \times 2 \mathbf{H}$-orthogonal transformation (Chamber’s algorithm) 74

5.1 Synchronization Block 93
5.2 Probability of acquisition and standard deviation of the error in the delay estimate over a range of signal to noise ratios for different frontend algorithms ($K=5$). 115
5.3 Probability of acquisition and standard deviation of the error in the delay estimate over a range of signal to noise ratios for different frontend algorithms ($K=13$). 116

6.1 Parallel architecture to implement deterministic MUSIC backend for a single user 130
6.2 Details of a parallel architecture to implement deterministic MUSIC backend for a single user 131
6.3 Parallel architecture to implement Maximum-likelihood MUSIC backend for a single user 133
6.4 Parallel architecture to implement Multipath MUSIC backend for a single user 136
What they undertook to do
They brought to pass;
All things hang like a drop of dew
Upon a blade of grass.

W. B. Yeats, "Gratitude to the unknown instructors"
To my parents
Chapter 1

Introduction

1.1 Motivation

Subspace-based signal processing is a class of signal processing algorithms based on estimation of certain invariant subspaces of the vector space spanned by the observations. This class of algorithms was originally developed for array signal processing problems. Array signal processing involves sampling propagating wave fields both in time and space using an array of antennas and processing the observations to preferentially receive signals from certain directions while attenuating the signals from other directions. The subspace-based techniques (also called eigenanalysis techniques) form a class of high resolution techniques for signal direction of arrival estimation.

A recent application of these algorithms is near-far resistant acquisition of Code Division Multiple Access (CDMA) communication signals. CDMA is a technique to allow multiple users access to a given communication channel and may be considered a generalization of time-division multiple access (TDMA) and frequency-division multiple access (FDMA). The IS-54 standard and the European standard GSM (Global System for Mobile Communications) [54] for digital cellular telephony are based on TDMA techniques, while the IS-95 [64] standard is based on a version of CDMA. The following description by Flower [24] gives a flavor of the differences between the different techniques:

Imagine eight people at a dinner table trying to have four conversations at once, with nobody seated next to the person he or she is talking to. Cacophony. You could give each person one minute to talk while everyone
else shuts up for one biker to tell the other about his new super-stroke, a minute for one father to brag to another about his 5-year-old. Call this time division.

Or you could assign each conversation a language: the bikers talk in Farsi, the fathers in Nahuatl. As long as they know the right languages, and as long as no one shouts, everyone can have their conversations at the same time. Call this code division.

CDMA or spread spectrum techniques have been in use in military communication systems for quite sometime. The main reason for the use of CDMA in such systems was the anti-jam capabilities of such systems. Recently CDMA has found use in commercial systems due to other properties such as multi-path resistance, coexistence with existing narrow band systems and higher bandwidth utilization.

The restriction "... as long as no one shouts..." in the above description of CDMA is called power control. This requires a central base station to monitor the error rate of each user and send back information to either raise or lower the user power to result in a feedback system. An alternative implementation of CDMA allows interference cancellation, with no restriction on the power transmitted by the different users. This class of algorithms for CDMA communications are termed near-far resistant. Such systems would be applicable whenever power control is not feasible.

The CDMA acquisition problem consists of estimating the time delays of different users when they simultaneously access the communication channel. The users are assumed to be asynchronous and the base station is assumed to receive a linear combination of the different users’ signals. From these observations it is necessary to estimate the time instant when the frames for the different users’ packets start, so the data in the frames may be recovered. More traditional techniques such as early-late gate techniques use a delay-locked loop to acquire the timing of a given user. However, in a CDMA context, these algorithms are not near-far resistant. The subspace-based
Figure 1.1: A geographical region is divided into cells, each serviced by a base station. All mobile users within a cell communicate with the base station for that cell. The cells themselves are interconnected typically using an existing land-based interconnect.

methods on the other hand are techniques applicable in the presence of the near-far effect.

In cellular systems, such as the IS-95 standard, a given geographical area is divided into cells, each serviced by a base station (see Fig. 1.1). All the users within a cell communicate with the base station through a wireless handset. The base stations themselves are interconnected typically through existing terrestrial links such as the T1 lines common in wireline telephone communications. Typical parameters of a CDMA cellular system for toll-quality voice communications are shown in Table 1.1.

<table>
<thead>
<tr>
<th>Table 1.1: Sample parameters for a voice communication system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Bandwidth</td>
</tr>
<tr>
<td>Desired Bit rate</td>
</tr>
<tr>
<td>Error probability</td>
</tr>
<tr>
<td>Average power limitation per user</td>
</tr>
<tr>
<td>Desired capacity</td>
</tr>
</tbody>
</table>
A recent trend in digital cellular systems is to use antenna array processing techniques to result in a form of space division multiple access. Such techniques would allow multiple users to reuse the spectrum within a cell and hence improve spectral utilization. Currently, this is done in the IS-95 standard by using three directional antennas to divide a cell into three sectors covering 120 degrees each.

Subspace-based techniques find application in such systems when the directional antennas are replaced by arrays of omnidirectional antennas and through the use of additional signal processing, typically beamforming techniques. The high resolution direction of arrival techniques may in addition be combined with CDMA acquisition to result in yet another application for subspace based techniques to jointly estimate user time delays and the directions of arrivals.

Another use of the subspace-based techniques is in system identification, where the state-space representation of the system generating the observations needs to be identified. A tutorial introduction to the connection between these state space identification techniques in control theory and the subspace-based techniques may be found in Vanderveen, Deprettere and Swindlehurst [66].

1.2 Methodology

Within the last decade a large body of work has focussed on developing parallel and numerically stable algorithms for subspace-based techniques applied to direction-finding problems in array signal processing. The goals of this thesis are three-fold. One goal is to investigate many of these previous algorithms simultaneously from communications, signal processing, numerical, architectural and implementation viewpoints. Another goal is to use the insight gained from investigating all these views to develop newer algorithms for the same problems. A third goal is to investigate
the applicability of these algorithms to CDMA acquisition problems and study some practical implementation issues which arise in this context. The results of this thesis would form a basis for implementation of the CDMA acquisition problem.

A key aspect of this thesis is the investigation of these algorithms from various viewpoints. Each viewpoint reveals a number of different approaches to the problem and imposes certain restrictions on the resulting algorithms/architectures. In the CDMA acquisition problem, the communications viewpoint essentially determines the performance requirements of the final system. The data rates and bandwidths set the throughput requirements of the system. For instance, the GSM system uses a bandwidth of 25MHz which is divided into 8 time slots and 124 frequency slots to result in a symbol period in the order of $3.7 \mu sec$. In the IS-95 standard, the user data rate is 9600 bits/sec, while the bandwidth is 1.2MHz, which results in a chip interval of $1 \mu sec$. Hence samples arrive in the IS-95 system at the rate of 1.2MHz, which essentially determines the throughput requirements. Other application dependent considerations determine the latency requirements. For instance, in two-way speech communications there would be a limit on the maximum tolerable latency. Typically data communications such as file transfers have less stringent latency requirements, but higher throughput requirements. In implementations involving decision-feedback, stability of the algorithms would dictate the maximum latency. In particular it would be necessary to restrict the tracking error of the decisions being fed back to be less than the time variation of the desired parameters. The tracking error depends directly on the latency of the estimation algorithms. The acceptable application dependent bit error rates determine the accuracy of the desired estimates and hence indirectly the computation requirements.

The signal processing issues enter the picture when one needs to deal with the stochastic nature of the signals and when the same problem could be reformulated
in several ways to account for issues such as non-stationarity of signals. In a communications system, such as digital cellular phones, the received signals are affected by movement of the mobile users to result in fading. The presence of obstructions and reflecting surfaces such as buildings results in multi-path. The signal processing viewpoint determines how all these variations may be accounted for to result in accurate estimates of the desired quantities. The signal processing also determines the feasibility of satisfying the desired communications requirements. The communications and signal processing issues together determine the dimensions of the matrices and the quantities to be computed. For instance, a bandwidth of 1.2MHz and a data rate of 9600 baud as in the IS-95 standard results in a spreading gain of 128, which in turn determines one of the dimensions of the matrices involved. The signal processing considerations determine whether a non-recursive method or a recursive method is required to handle requirements. This determines the other dimension of the matrix whose SVD needs to be computed at each time instant. We will discuss such issues in greater detail in the rest of the text.

The numerical viewpoint investigates how the desired quantities could be approximated and computed in a numerically stable manner. The numerical stability of the resulting algorithms plays a key role in practical applicability of the algorithms. The goal is to reduce the required computations, while simultaneously preserving their stability. For instance, if only an orthonormal basis for the noise subspace is desired, algorithms have been developed which require an order of magnitude less computation than an algorithm which computes a complete 2-sided decomposition. Different algorithms which tradeoff computation for accuracy are possible for the same calculation. This allows one to choose the right amount of computation for the desired accuracy. For instance, the subspace tracking algorithm by Moonen, Van Dooren and
Vandewalle [49] allows one to trade off accuracy for the number of Jacobi steps that need to be performed at each update.

The architectural viewpoint investigates algorithms at both the array and the processor level. At the array level, the algorithms are developed to have key properties such as systolic computation or distributed memory computation. The emphasis in this thesis is the development of algorithms with these desired properties. The systolic arrays shown in this thesis should be interpreted more as systolic algorithms and a logical architecture for the implementation of the algorithm. The physical architecture of the arrays would look very different. The arrays would have to be mapped onto physical processors to result in the final architecture. We do not investigate this process of mapping the logical arrays onto physical array. There is an entire field of study devoted to this problem and a number of automatic techniques and tools have been developed and are being developed. The mapping would depend on application requirements and the current technology. The requirement of systolic computation restricts communication to occur only between adjacent processors of an array. The systolic paradigm results in extensive parallelism and pipelining and a reduction in the host to array communication requirements. Distributed computation is a desirable property since it avoids problems of sharing memory and the associated coherency and consistency problems.

At the processor level we investigate primarily the fixed-point versus floating-point arithmetic tradeoff. Since floating-point is significantly more complex than fixed-point, it is desirable to design all algorithms to operate in fixed-point. This in turn affects the numerical aspects of algorithms and hence is primarily investigated at that level. Other aspects of the processor viewpoint is the use of special-purpose arithmetic techniques such as CORDIC (Co-ordinate Rotation Digital Computer). Although this thesis does not cover it, the processor level also requires study of the
architecture of the processor, the processor resource requirements such as registers, busses and controllers, and scheduling requirements.

1.3 Overview of the thesis

The thesis is divided into four chapters. Chapters 3 and 4 describe two ways to perform what we call frontend subspace tracking. Chapter 3 describes the pipelined SVD, an algorithm applicable when a non-recursive formulation of the subspace-tracking problem is used. This algorithm is an offshoot of investigating subspace tracking at the numerical and architectural levels. Most of the previous algorithms applicable for non-recursive (or block) formulations of the subspace-based methods, assume an "in-core" computation. This implies frequent breaks in the pipeline whenever a new block of data needs to be processed, since the host I/O communications has not been considered part of the algorithm. The pipelined SVD uses a different ordering for the Jacobi method resulting in an algorithm that seamlessly integrates the host I/O with the rest of the SVD computation. The resulting algorithm is extremely simple, allows SVD computation from adjacent data blocks to proceed without any pipeline breaks and allows a wide variety of implementations. The resulting ordering is shown to be equivalent to other orderings known to converge.

Chapter 4 describes how the SVD of a block of observations may be updated when a new row is added and an old row deleted. We call the resulting formulation, the sliding window SVD. We show how the hyperbolic QR downdating technique [14] and the exponential downdating algorithm by Moonen [49] may be combined to result in an array that implements the sliding window SVD updating. We will extend the relational stability analysis of Stewart [61] to this new algorithm implemented
using mostly fixed-point arithmetic and show that stability properties similar to those obtained by other algorithms may be obtained.

Chapter 5 describes the subspace-based method for CDMA acquisition and describes various formulations of the problem that result from investigation from different viewpoints. We show how the implementation may be divided into frontend and backend algorithms. We then analyze previous algorithms using a variety of criteria and discuss some advantages and caveats of using the algorithms in the CDMA system.

Chapter 6 describes parallel algorithms for backend algorithms for CDMA in greater detail. This chapter presents a new architecture which maps well to a variety of backend problem formulations.
Chapter 2

Background

The algorithms in this thesis build on a significant amount of research done in many different areas, such as numerical linear algebra, CDMA communications, array signal processing, parallel algorithms, parallel computer architectures and systolic algorithms. This chapter will present a brief introduction to some of the ideas from these areas used extensively in this thesis.

2.1 Techniques from numerical linear algebra

Numerical linear algebra techniques play an important role in the practical implementation of many signal processing algorithms. In this thesis, one-sided and two-sided orthogonal decompositions of data matrices are used extensively to estimate signal and noise subspaces. Surprises from finite-precision effects are avoided through the use of carefully constructed algorithms from linear algebra research. Algorithms based on linear algebra concepts are in general very regular and allow significant pipelining and parallelism to be extracted from the computation. In this thesis, the Jacobi method plays a crucial role in the implementation because of the amount of flexibility that it allows in ordering of the computations. Such flexibility is a key property required of algorithms in order to effectively parallelize them. We will define a number of important decompositions and then describe algorithms to compute them.
2.1.1 Givens rotation

A Givens rotation, $\Phi^{pq}(\theta) \in \mathbb{R}^{n \times n}$, parameterized by a pivot pair $(p, q)$ and an angle $\theta$, is an orthogonal transformation defined as:

\[
\Phi^{pq}(\theta) = \begin{pmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & \cdots & \cos \theta & \cdots & \sin \theta & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots & \ddots & \vdots \\
0 & \cdots & -\sin \theta & \cdots & \cos \theta & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1
\end{pmatrix}
\]

(2.1)

which is the identity matrix except for the $2 \times 2$ submatrix formed by the intersection of the $p^{th}$ and $q^{th}$ rows and columns of the matrix. The Givens rotation is used to zero selective elements in a matrix. It may be considered a vector rotation transformation in the $(p, q)$ plane. Due to its special structure, pre-multiplication by a Givens rotation affects only the $p^{th}$ and $q^{th}$ rows of a matrix. Similarly, post-multiplication by a Givens rotation affects only the corresponding two columns of a matrix.

The parameter $\theta$ of a Givens rotation is typically computed to transform a 2-vector \(
\begin{pmatrix}
\alpha \\
\beta
\end{pmatrix}
\) into a vector of the form \(
\begin{pmatrix}
\sqrt{\alpha^2 + \beta^2} \\
0
\end{pmatrix}
\). When fixed-point arithmetic is used to compute the desired quantities, care needs to be exercised on how this transformation is computed. An algorithm to compute this rotation with small absolute error as described by Wilkinson [71] is shown in Algorithm 2.1. The Givens rotation is stored as two fixed-point numbers ($c, s$) corresponding to $(\cos \theta, \sin \theta)$. Pre-multiplication of a vector \(
\begin{pmatrix}
\alpha \\
\beta
\end{pmatrix}
\) by a $2 \times 2$ Givens rotation computed in the previous step, is shown in Algorithm 2.2.
Algorithm 2.1 \( \{c, s\} \leftarrow \text{Givens.compute } \{\alpha, \beta\} \)

if \( \alpha = 0 \)
  \( c \leftarrow 0 \)
  \( s \leftarrow 1 \)
elseif \( |\alpha| < |\beta| \)
  \( m \leftarrow \alpha / \beta \)
  \( c \leftarrow \frac{1}{2}m / \sqrt{\frac{1}{4}m^2 + \frac{1}{4}} \)
  \( s \leftarrow \frac{1}{2} / \sqrt{\frac{1}{4}m^2 + \frac{1}{4}} \)
else
  \( n \leftarrow \beta / \alpha \)
  \( c \leftarrow \frac{1}{2} / \sqrt{\frac{1}{4}n^2 + \frac{1}{4}} \)
  \( s \leftarrow \frac{1}{2}n / \sqrt{\frac{1}{4}n^2 + \frac{1}{4}} \)
end

2.1.2 The QR decomposition

The QR decomposition of a matrix \( A \in \mathcal{R}^{m \times n} \) is an orthogonal matrix \( Q \in \mathcal{R}^{m \times m} \) and an upper right trapezoidal matrix \( R \in \mathcal{R}^{r \times n} \) related as follows:

\[ A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \]  

(2.2)

where \( r \) is the rank of the matrix \( A \). If the matrix \( A \) is full rank, then \( R \) is upper right triangular. Columns of the matrix \( Q \) form a basis for the column space of the matrix \( A \).

The QR decomposition may be computed using Givens rotations. If we use \( \Phi_k^{pq} \) to denote the Givens rotation that zeros the \((p, q)^{th}\) element at the \( k^{th} \) step, then the algorithm may be described graphically as in Fig. 2.1, where an \( X \) denotes some
Algorithm 2.2 \( \{\gamma, \delta\} \leftarrow \text{Givens.apply} \{\alpha, \beta, c, s\} \)

\[
\gamma \leftarrow \alpha * c + \beta * s \\
\delta \leftarrow -\alpha * s + \beta * c
\]

end

non-zero quantity, \(\epsilon\) denotes a small number close to the machine precision and \(X\) is used to denote the elements which changed at the previous step. The orthogonal matrix is obtained as the product of the Givens rotations computed at each step. Hence in the example, the matrix \(Q'\) is obtained as \(\Phi_{6}^{34} \Phi_{5}^{33} \Phi_{4}^{32} \Phi_{3}^{12} \Phi_{2}^{13} \Phi_{1}^{14}\).

2.1.3 QR updating

Often in signal processing problems, as new observations become available, the data matrix is appended with a new row and the QR decomposition of this updated matrix is desired. In such situations, it is economical to update the previously available decomposition instead of recomputing the decomposition from scratch. Suppose \(A_{k}\) denotes the data matrix at time step \(k\). Then at time step \((k + 1)\), the data matrix

\[
A = \begin{pmatrix}
X & X & X \\
X & X & X \\
X & X & X \\
X & X & X \\
X & X & X \\
X & X & X \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
X & X & X \\
X & X & X \\
X & X & X \\
X & X & X \\
X & X & X \\
X & X & X \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\epsilon & X & X \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
\epsilon & \epsilon & \epsilon \\
\epsilon & \epsilon & \epsilon \\
\epsilon & \epsilon & \epsilon \\
\epsilon & \epsilon & \epsilon \\
\epsilon & \epsilon & \epsilon \\
\epsilon & \epsilon & \epsilon \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
\Phi_{1}^{14} & \Phi_{2}^{13} & \Phi_{3}^{12} & \Phi_{4}^{11} & \Phi_{5}^{10} & \Phi_{6}^{9} & \Phi_{7}^{8} & \Phi_{8}^{7} & \Phi_{9}^{6} & \Phi_{10}^{5} & \Phi_{11}^{4} & \Phi_{12}^{3} & \Phi_{13}^{2} & \Phi_{14}^{1} & \Phi_{15}^{0}
\end{pmatrix}
\]

Figure 2.1: Computing the QR decomposition using Givens rotations
\( A_{k+1} \) can be recursively defined as:

\[
A_{k+1} = \begin{pmatrix} A_k \\ a'_{k+1} \end{pmatrix}.
\] (2.3)

Hence, if the QR decomposition at time step \( k \) is available, then the decomposition at time step \((k+1)\) may be obtained as:

\[
A_{k+1} = \begin{pmatrix} Q_k R_k \\ a'_{k+1} \end{pmatrix} = \begin{pmatrix} Q_k & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_k \\ a'_{k+1} \end{pmatrix}.
\] (2.4)

Hence, through a sequence of Givens rotations this may be updated as follows:

\[
A_{k+1} = \begin{pmatrix} Q_k & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \phi_1^{n+1} \\ \phi_2^{n+1} \\ \cdots \\ \phi_n^{n+1} \end{pmatrix} \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix}.
\] (2.5)

\( Q_{k+1} \)

An important point to observe is that the orthogonal factor \( Q_k \) is not explicitly required to update the factor \( R_k \). Note that in the QR updating problem, the dimensions of the matrix \( Q_k \) increase by one at each update. The upper right triangular factor \( R_k \), however, is a fixed-size matrix. Applications which require updating of this form normally only need the \( R_k \) factor and hence the continually growing memory requirements of \( Q_k \) would not be a concern.

2.1.4 Singular Value Decomposition (SVD)

Given a \( m \times n \) matrix \( A \), the SVD of \( A \) is a two-sided decomposition of the form

\[
A = U \Sigma V',
\] (2.6)

where \( U \) and \( V \) are \( m \times m \) and \( n \times n \) orthogonal matrices and \( \Sigma \) is a \( m \times n \) diagonal matrix of singular values. The columns of \( U \) and \( V \) are called the left and right singular vectors, respectively. A left singular vector, a right singular vector and a singular value form a triplet. The columns of \( U \) which correspond to non-zero
singular values form a basis for the column space of $A$, while the columns of $V$ which
correspond to non-zero singular values form a basis for the row space of $A$. Hence,
one of the uses of the SVD is to expose orthonormal bases for the column and row
spaces of a matrix.

2.1.5 The Jacobi method to compute the SVD

The Jacobi method was originally developed for the symmetric eigenvalue prob-
lem. The algorithm was later extended to compute the singular value decomposition
by Kogbetliantz [40]. The cyclic Jacobi method was used successfully in practice for
a long time before the convergence of the algorithm was proved by Forsythe and
Henrici [25].

The Jacobi algorithm to compute the SVD consists of computing orthogonal ma-
trices $U_i$ and $V_i$ such that the sequence of matrices $A_i$ converges to a diagonal matrix,
where the matrices $A_i$ are defined as

$$A_0 = A$$
$$A_{i+1} = U_i^t A_i V_i.$$  \hspace{1cm} (2.7)

The Jacobi method consists of choosing a pivot $(p_i, q_i)$ at an iteration $i$ such that
$1 \leq p_i < q_i \leq n$ and computing Givens rotations $U_i$ and $V_i$ to annihilate the $(p_i, q_i)$
and the $(q_i, p_i)$ elements of $A_i$. The two Givens rotations, $U_i$ and $V_i$, are determined
by the $2 \times 2$ submatrix formed by the intersection of the $p_i^{th}$ and $q_i^{th}$ rows and columns:

$$
\begin{pmatrix}
0 & c_1 & s_1 \\
-s_1 & c_1 & 0
\end{pmatrix}
\begin{pmatrix}
a_{p_i,p_i} & a_{p_i,q_i} \\
a_{q_i,p_i} & a_{q_i,q_i}
\end{pmatrix}
\begin{pmatrix}
c_2 & s_2 \\
-s_2 & c_2
\end{pmatrix}.
$$  \hspace{1cm} (2.8)

Fig. 2.2 illustrates the matrix elements affected at each Jacobi step. Notice that each
step destroys the zeros created at the previous step. However, at the end of the sweep
we are guaranteed that the off-diagonal norm of the matrix is reduced. Several such
sweeps will drive the matrix towards a diagonal matrix.
Figure 2.2: One sweep of the Jacobi algorithm to compute SVD

The following algorithm which is a fixed-point variant of the algorithm described by Brent, Luk and van Loan [11] shows a way to compute the two Givens rotations at each Jacobi step. The rotations are stored as fixed-point numbers \( (c_1, s_1, c_2, s_2) \). The procedure described below ensures that the absolute error in the computed rotations is small.

**Algorithm 2.3** \( \{c_1, s_1, c_2, s_2\} \leftarrow \text{Jacobi.compute}\left(\begin{pmatrix} w & x \\ y & z \end{pmatrix}\right) \)

\[
\mu_1 \leftarrow z - w \\
\mu_2 \leftarrow y + x \\
\text{if } (\mu_2 = 0) \\
\quad \chi_1 \leftarrow 1 \\
\quad \sigma_1 \leftarrow 0 \\
\text{elsif } (|\mu_2| \geq |\mu_1|) \\
\quad \rho_1 \leftarrow \frac{\mu_1}{\mu_2} \\
\quad \tau_1 \leftarrow \frac{\text{sign}(\rho_1)}{|\rho_1| + \sqrt{1 + \rho_1^2}} \\
\quad \chi_1 \leftarrow \frac{\chi}{\sqrt{1 + \tau_1^2}} \\
\quad \sigma_1 \leftarrow 2\chi_1 \tau_1 \\
\text{else}
\]
\[ \begin{align*}
\rho_1 & \leftarrow \frac{\mu_2}{2\mu_1} \\
\tau_1 & \leftarrow \frac{\frac{1}{2}p_1}{\frac{1}{2} + \sqrt{\frac{1}{4} + p_1^2}} \\
\chi_1 & \leftarrow \frac{\frac{1}{2}}{\sqrt{\frac{1}{4} + \tau_1^2}} \\
\sigma_1 & \leftarrow 2\chi_1\tau_1
\end{align*} \]

end

\[ \begin{align*}
\mu_1 & \leftarrow z + w \\
\mu_2 & \leftarrow y - x
\end{align*} \]

if (\(\mu_2 = 0\))

\[ \begin{align*}
\chi_2 & \leftarrow 1 \\
\sigma_2 & \leftarrow 0
\end{align*} \]

elsif (\(|\mu_2| \geq |\mu_1|\))

\[ \begin{align*}
\rho_2 & \leftarrow \frac{\mu_1}{2\mu_2} \\
\tau_2 & \leftarrow \frac{\text{sign}(\sigma_2)}{|\sigma_2| + \sqrt{\frac{1}{4} + \rho_2^2}} \\
\chi_2 & \leftarrow \frac{\frac{1}{2}}{\sqrt{\frac{1}{4} + \tau_2^2}} \\
\sigma_2 & \leftarrow 2\chi_2\tau_2
\end{align*} \]

else

\[ \begin{align*}
\rho_2 & \leftarrow \frac{\mu_2}{2\mu_1} \\
\tau_2 & \leftarrow \frac{\frac{1}{2}p_2}{\frac{1}{2} + \sqrt{\frac{1}{4} + p_2^2}} \\
\chi_2 & \leftarrow \frac{\frac{1}{2}}{\sqrt{\frac{1}{4} + \tau_2^2}} \\
\sigma_2 & \leftarrow 2\chi_2\tau_2
\end{align*} \]

end

\[ \begin{align*}
c_1 & \leftarrow \chi_1 \ast \chi_2 + \sigma_1 \ast \sigma_2 \\
s_1 & \leftarrow \sigma_1 \ast \chi_2 - \chi_1 \ast \sigma_2 \\
c_2 & \leftarrow \chi_1 \ast \chi_2 - \sigma_1 \ast \sigma_2 \\
s_2 & \leftarrow \sigma_1 \ast \chi_2 + \chi_1 \ast \sigma_2
\end{align*} \]

end

In the original Jacobi method \((p_i, q_i)\) is chosen at each iteration to be the indices for which the off-diagonal element is the largest. The cyclic Jacobi method is obtained by performing several sweeps of \(n(n-1)/2\) iterations, where the pivots within a sweep
are chosen a priori based on an ordering to successively annihilate each off-diagonal pair of elements i.e. the choice of pivots is independent of the matrix. Forsythe and Henrici [25] proved the convergence of this algorithm for the cyclic-by-rows ordering. The cyclic-by-rows ordering corresponds to the following sequence of pivots in each sweep:

\[(1,2)(1,3)(1,4)\cdots(1,n)\]
\[(2,3)(2,4)\cdots(2,n)\]
\[(3,4)\cdots(3,n)\]
\[\vdots\]
\[(n-1,n)\]

The parallel cyclic Jacobi method involves choosing a cyclic ordering which allows the Givens rotations of several iterations to be computed in parallel. The algorithm described above will be referred to as the two-sided Jacobi method. The one-sided Hestenes method [37] consists of computing a single Givens rotation \(V_i\) at each iteration to force the sequence of matrices \(A_i\) to converge to a matrix with orthogonal columns, where \(A_i\) are defined as

\[A_0 = A\]
\[A_{i+1} = A_iV_i.\]  \hspace{1cm} (2.9)

The one-sided method may be regarded as an implicit two-sided method applied to the symmetric matrix \(A^TA\). A different one-sided method proposed by Eberlein and Park [22] computes two rotations as in the two-sided method. However, only local column updates are performed, postponing the update of other matrix elements to when needed at a later iteration. The computational complexity of this method is
comparable to the Hestenes method. For our purposes it suffices to group this method along with the Hestenes method under the unified title of one-sided Jacobi methods.

Noting the structure of Givens rotations, several parallel orderings have been proposed which allow the rotations of $k \approx n/2$ successive iterations to be computed independent of the intervening iterations. The one-sided Jacobi methods, in addition, allow these rotations to be applied independently to columns of $A_i$, to update $A_i$ directly to $A_{i+k}$. Consequently parallel implementations of the one-sided Jacobi methods exploit $O(n)$ parallelism by performing approximately $n/2$ iterations in parallel, resulting in linear arrays to compute the SVD. In two-sided Jacobi methods, on the other hand, updates to $2 \times 2$ submatrices of $A_i$ are independent of each other, allowing update of these submatrices directly to the corresponding $2 \times 2$ submatrices in $A_{i+k}$. The application of these plane rotations can be pipelined to result in effectively a constant time update, thus achieving another $O(n)$ parallelism. This is the essence of the square arrays for two-sided Jacobi methods. Hence, the computation now consists of several sweeps where each sweep can be executed in $O(n)$ time. It is conjectured that a predetermined number of sweeps $O(\log n)$ suffice in practice, resulting in an algorithm that computes the SVD in $O(n \log n)$ time on $O(n^2)$ processors or $O(n^2 \log n)$ time on $O(n)$ processors. Most of the previous algorithms thus result in a throughput of either $O(1/n \log n)$ or $O(1/n^2 \log n)$. Note that if higher throughput is required, it is always possible to achieve it by using multiple arrays in parallel and multiplexing the inputs to and outputs from these arrays. Though such a scheme increases the throughput by an arbitrary amount (at least in theory), the latency is still limited to $O(n \log n)$ for a square array and $O(n^2 \log n)$ for a linear array. Hence a scheme that just improves the throughput without any other benefits would not be interesting in itself.
2.2 Signal Subspace based methods in signal processing

2.2.1 A mathematical model for observations

The starting point of signal subspace methods is a linear model for the observation vectors. These methods require a series of observation vectors which may be considered the result of a linear filter corrupted by noise, as shown in Fig. 2.3. The input to the filter, \( x_i \in \mathbb{C}^{2K} \), is a stochastic process. This vector is filtered by a memoryless filter, \( A(\tau) \in \mathbb{C}^{N \times 2K} \). The resulting vector is corrupted by additive white Gaussian noise to result in the observation vector, \( y_i \):

\[
y_i = A(\tau)x_i + n_i.
\] (2.10)

The filter response \( A(\tau) \) is some known function of unknown parameters \( \tau \). The goal of the algorithm is to estimate these possibly slowly changing parameters from the received observations with no knowledge of the input data vector \( x_i \) except possibly for second order statistics.

In the direction of arrival problems, it is desired to estimate the direction of arrival of certain narrowband signals using data obtained from an antenna array. Each observation vector, \( y_i \), in this problem is formed by simultaneously sampling the output of the array of antennas and stacking these observations to form a vector. A series of such observation vectors is obtained by sampling the antenna outputs periodically. The resulting observations have the above linear structure, where the matrix \( A(\tau) \), called the array manifold, depends on the geometry of the array, the impulse response of the sensors and the direction of arrival of the signals. Hence, in
practice, since the geometry of the array and the response of the sensors are design parameters, the structure of the array manifold is known as a function of the unknown directions of arrival of the signals. The input vector $x_t$ depends on the actual signal transmitted and does not affect the array manifold. The input vector is obtained by sampling all the transmitted signals at a time instant, and stacking all these values to form a vector. The array manifold typically consists of one column for each signal impinging on the array and is dependent on the direction of arrival of that signal but independent of the directions of arrival of all the other signals. This structure is a direct result of the linearity of the wave equations which determine the physics of propagating waves. Hence given a series of observations and provided that the signal sources move relatively slowly, the directions of arrival of all the signals may be estimated using signal subspace methods. Later in this chapter we discuss how a similar structure is obtained for the CDMA acquisition problem.

2.2.2 Maximum-likelihood estimation of the parameters

Given the above mathematical model for the observations, it is desired to estimate the unknown parameters and track them. When an a priori distribution on the unknown parameters is unavailable, a maximum-likelihood estimation of the problem is appropriate. Maximum-likelihood formulation [39] of the above estimation problem may be stated as follows:

$$\hat{(\tau_{ML}, X_{ML})} = \arg \max_{\tau, X} f_{Y|\tau, X}(Y|\tau, X),$$

(2.11)

in other words, the maximum-likelihood estimates are the parameters which maximize the probability density function of the observation matrix $Y$, which is formed from $L$ columns each of which is an observation vector $y_t$ defined as before. The input data matrix $X$ is formed in a similar manner.
For a given set of parameters $\tau$, the probability distribution of $L$ observation vectors assuming independent, zero mean, circular Gaussian noise vectors is:

$$f_{Y|\tau,X}(Y|\tau, X) = \prod_{i=0}^{L-1} \frac{1}{\sqrt{\det[\pi K_N]}} \exp \left\{ -\frac{1}{2} (y_i - A(\tau)x_i)'K_N^{-1}(y_i - A(\tau)x_i) \right\}$$

(2.12)

where $K_N$ is the covariance matrix of the noise vector. If the covariance matrix is given by $\sigma^2 I$, then the maximum-likelihood estimates are given by:

$$(\hat{\tau}_{ML}, X_{ML}) = \arg\min_{\tau, X} \sum_{i=0}^{L-1} (y_i - A(\tau)x_i)'(y_i - A(\tau)x_i).$$

(2.13)

In a matrix notation, the same equation may be rewritten as:

$$(\hat{\tau}_{ML}, X_{ML}) = \arg\min_{\tau, X} \|Y - A(\tau)X\|_F^2,$$

(2.14)

where $\|.\|_F$ denotes the Frobenius norm. It is well-known that this problem may be separated into two simpler optimization problems [66]. The trick is to realize that for a given $\hat{\tau}_{ML}$, the solution for $X_{ML}$ is simply a projection of the observation matrix into the column space of $A(\tau)$. If $A(\tau)^+$ is the pseudo-inverse of $A(\tau)$, then the maximum-likelihood solution for the data matrix may be written as:

$$X_{ML} = A(\tau)^+ Y.$$  

(2.15)

Hence the maximum-likelihood solution for the parameter vector $\tau$ is given by the following optimization problem:

$$\hat{\tau}_{ML} = \arg\min_{\tau} \| (I - A(\tau)A(\tau)^+)Y \|_F^2,$$

(2.16)

If $S_A(\tau)^\perp$ denotes an orthonormal basis into the null space of $A(\tau)$, then the above equation may be rewritten as:

$$\hat{\tau}_{ML} = \arg\min_{\tau} \| S_A(\tau)^\perp Y \|_F^2.$$  

(2.17)
Notice that this formulation breaks down when the dimension of the data vectors, $x_i$, exceeds the dimension of the observation vectors, $y_i$, since the null space basis reduces to the zero matrix. In the CDMA problem, this translates into a limit on the number of users who can simultaneously use the system.

2.2.3 Reducing complexity of the ML estimation formulation

At this point in the problem, there are two approaches for the implementation of the estimation procedure. One of these is derived directly from the above maximum-likelihood formulation for the parameters. The first thing to note about the above formulation is that the basis for the null space of the matrix $A(\tilde{r})$ is a function of the parameters $\tilde{r}$. In most of the problems of interest such as array signal processing and the CDMA acquisition problem, this null space basis cannot be obtained in a closed form. This implies that the objective function for the above optimization problem has to be computed numerically and cannot be simplified any further. Hence, one of the implementation alternatives at this point is to use some optimization procedure (which will usually be based on some heuristic since the objective function is multimodal), and extract any computational efficiencies that may obtained in the numerical computation of the objective function. The rationale behind this approach is that the parameter vector $\tilde{r}$ is expected to change very slowly and hence the null space basis has to computed once and merely updated to track the parameters. However, the disadvantage of such an approach is that the matrices involved can get very large with the dimensions in the order of thousands for the CDMA problem. Such large dimensions are likely to pose implementation problems when processing at the data rates typically encountered in signal processing and communications problems.

The other approach tries to simplify the above maximum-likelihood formulation further by approximating it. This is the approach which underlies the implementa-
tions in this thesis. The simplification is obtained by observing that asymptotically as \( L \to \infty \), the observation matrix \( \mathbf{Y} \) may be replaced by a low rank approximation of the matrix, since most of the energy of the observations lies in a lower dimensional subspace. Hence if \( \hat{\mathbf{Y}} \) is a low rank approximation of the observation matrix \( \mathbf{Y} \), then the maximum-likelihood criterion may be approximated by the following expression:

\[
\tilde{\tau}_{\text{ML}} = \arg \min_{\hat{\mathbf{r}}} \left\| \mathbf{S}_{\mathbf{A}}(\hat{\mathbf{r}})^{\dagger} \hat{\mathbf{Y}} \right\|_F^2.
\]

(2.18)

2.3 CDMA Communications

Fig. 2.4 shows the components of a CDMA transmitter and receiver. The receiver consists of a RF frontend/demodulator (see Fig 2.5) which demodulates the composite of several user's signals down to baseband. The baseband signal consists of an in-phase and quadrature component, which form the real and imaginary parts of the observations received at the input of the baseband preprocessing unit. The subspace-based techniques which are used in this thesis assume the baseband processing shown in Fig 2.6.

2.3.1 System model

The CDMA system consists of \( K \) users communicating with a base station over a multiple access channel. The signal, \( s_k(t) \) transmitted by the \( k \)th user passes through a possibly time-varying linear channel with impulse response \( h_k(t) \). The received signal \( r(t) \) at the base station is a linear superposition of all the signals \( s_k(t) \ast h_k(t) \), where \( \ast \) denotes the convolution operator, corrupted by additive noise \( n(t) \).

\[
r(t) = \sum_{k=0}^{K-1} s_k(t) \ast h_k(t) + n(t)
\]

(2.19)
2.3.2 Transmitted Signal Model

The baseband signal or the equivalent baseband signal in a passband communication system, \( \tilde{s}_k(t) \), transmitted by user \( k \) consists of a pulse \( a_k(t) \) of duration \( T \) modulated in the \( i^{th} \) symbol interval by \( b_k^i \). The pulse \( a_k(t) \) itself consists of a transmit chip waveform \( g_k(t) \) of duration \( T_c \), called the chip period, modulated at the \( l^{th} \) chip interval by \( a_k^0 \).

\[
\tilde{s}_k(t) = \sum_i b_k^i a_k(t - iT) \\
= \sum_i b_k^i \sum_{l=0}^{N-1} a_k^l g_k(t - iT - lT_c) 
\]

(2.20) (2.21)

In a passband system, this waveform modulates a carrier of frequency \(\omega_c \) and results in the transmitted signal, \( s_k(t) \):

\[
s_k(t) = \sqrt{2} \text{Re}[\tilde{s}_k(t)e^{j\omega_c t}] 
\]

(2.22)

The spreading codes for the different users are chosen such that they have good autocorrelation and cross-correlation properties, i.e. the sequences should be almost
orthogonal to any shifts of themselves or the spreading codes of other users. Typically noise-like codes such as M-sequences or Gold sequences are used in these systems.

Figure 2.7 shows an example of the signal generated by a given user. The example shows a system with a spreading gain $N = 7$. The user has been assigned a 7-bit spreading code $a_k = (-1, -1, 1, -1, 1, 1, 1)$. The data transmitted in the three successive bit intervals shown are $b_k^{-1} = -1$, $b_k^i = -1$ and $b_k^{i+1} = 1$. The baseband chip waveform, $g_k(t)$, is a rectangular pulse of duration $T_c$, the chip interval. The resulting baseband CDMA signal $\tilde{s}_k(t)$ is used to modulate the carrier to result in the passband transmitted signal.
2.3.3 Channel Model

The signal of the $k^{th}$ user is received by the receiver through a channel $h_k(t)$ which introduces an arbitrary attenuation, and an arbitrary delay.

$$h_k(t) = \beta_k \delta(t - \tau_k) \quad (2.23)$$

A multipath channel may be modeled as generating $r$ dominant paths, each of which introduces an arbitrary attenuation and delay:

$$h_k(t) = \sum_{i=0}^{r-1} \beta_k^{(i)} \delta(t - \tau_k^{(i)}). \quad (2.24)$$

In this thesis we assume that the multipath spread is restricted to be less than half the period of the spreading code. We refer the reader to Bensley [2] for the reasoning behind this assumption.
Figure 2.7: Example of the CDMA signal generated by one user
2.3.4 Subspace-based algorithms

Formation of observation vectors

The received passband signal $r(t)$ is initially demodulated to the corresponding complex baseband signal $\tilde{r}(t)$. There are several formulations and implementations of this step [44]. For simplicity, we will assume that the frontend of the receiver consists of an analytic receive filter with impulse response $\sqrt{2}f(t)e^{j\omega t}$, followed by a multiplication with the signal $e^{-j\omega t}$ to produce the equivalent baseband signal $\tilde{r}(t)$ (Fig. 2.8). We will assume that the carrier frequency is known. Hence the received complex baseband signal may be written as:

$$\tilde{r}(t) = \sum_k \sqrt{2} P_k e^{j\phi_k} \sum_i \hat{b}_k a_k(t - iT - \tau_k) + \tilde{n}(t),$$

(2.25)

where, $\phi_k$ is the phase of the $k^{th}$ user's carrier with respect to the local oscillator and $P_k$ is the received signal power for the $k^{th}$ user.

The equivalent baseband signal $\tilde{r}(t)$ is passed through a chip matched filter, $g(-t)$. The output of this filter is sampled at the chip rate and constitute the observations $y_{i,t}$. An observation vector is formed from $N$ successive samples. The timing reference to form the observations is arbitrary. In general, the observation vector, $y_i$, will correspond to two adjacent symbols of each user.

$$y_i = \begin{pmatrix} y_{i,0} \\ y_{i,1} \\ \vdots \\ y_{i,N-1} \end{pmatrix} \in \mathbb{C}^{N \times 1}$$

(2.26)
where,
\[ y_{i,t} = \int_{iT+iT_c}^{iT+(t+1)T_c} \tilde{r}(t)g(t - iT - iT_c)dt. \] (2.27)

**Statistical properties of the observation vectors**

If the time delays of the users do not change very fast, then over small periods of time, the observation vectors may be considered the result of a stationary random process. Hence the approach we take in the development of our acquisition algorithms is to design subspace-based algorithms assuming a stationary random process and then extend these algorithms to the non-stationary case.

Under the assumption of stationary conditions, the observation vectors may be modeled as the result of a linear filter corrupted by noise, as shown in Fig. 2.3. The input, \( x_i \in \mathbb{C}^{2K} \), to the filter is a random process. This vector is filtered by a linear filter, \( A(\tilde{r}) \in \mathbb{C}^{N \times 2K} \). The result of the filter is corrupted by additive white Gaussian noise to result in the observation vector, \( y_i \):

\[ y_i = A(\tilde{r})x_i + n_i. \] (2.28)

The structure of the linear filter \( A(\tilde{r}) \) is assumed to be completely known. By this we mean, for any given parameter vector, \( \tilde{r} \), the matrix, \( A(\tilde{r}) \) can be readily constructed.

We now present an interesting property of the correlation matrix of the received vectors. This property is a key to all the subspace-based algorithms presented in this thesis. If we assume that the input process is uncorrelated with the noise process, the correlation matrix of the received vectors may be written as:

\[ R = \mathcal{E}[y_i y_i'] \]
\[ = A(\tilde{r})\mathcal{E}[x_i x_i']A(\tilde{r})' + \mathcal{E}[n_i n_i'] \] (2.29)
\[ = A(\tilde{r})SA(\tilde{r})' + \sigma^2 I, \]
where, $S \in \mathcal{C}^{2K \times 2K}$ is the correlation matrix of the input random process. From the above equation it can be shown that given sufficient signal to noise ratio, the eigenvalue decomposition of the correlation matrix will reveal a structure where the smallest $N - 2K$ eigenvalues and the corresponding eigenvectors correspond to the noise subspace, while the largest $2K$ eigenvalues correspond to the signal subspace. If the underlying random process is stationary (a condition that translates in this system to time delays which do not vary with time), an estimate of the correlation matrix may be obtained by forming the sample correlation matrix, $\hat{R}$:

$$\hat{R} = \frac{1}{L} \sum_{i=m}^{m+L-1} y_i y_i'.$$  \hspace{1cm} (2.30)

The eigenvalue decomposition of the sample correlation matrix can thus be used to estimate the signal and the noise subspaces:

$$\hat{R} = \hat{U} \Sigma \hat{U}' = \begin{pmatrix} U_s & U_n \end{pmatrix} \begin{pmatrix} \Sigma_s & 0 \\ 0 & \Sigma_n \end{pmatrix} \begin{pmatrix} U_s' \\ U_n' \end{pmatrix}$$  \hspace{1cm} (2.31)

where the columns of $U_n$ are the eigenvectors corresponding to the $N - 2K$ smallest eigenvalues and hence form an estimate for the basis of the noise subspace. It follows that the projection of any column of $A(\bar{r})$ into this estimated subspace will be non-zero with zero probability. It has been shown that this projection results in a vector that has a Gaussian distribution.

Structure of the $A(\bar{r})$ matrix

In this section we will show how the observations defined by Eq. 2.25, Eq. 2.26 and Eq. 2.27 may be rewritten in the form of Eq. 2.10. This derivation requires that all the users transmit signals using the same baseband chip waveform, $g(t)$.

An observation $y_{i,l}$ may be written from Eq. 2.21, Eq. 2.25 and Eq. 2.27 as

$$y_{i,l} = \int_{iT+Tc}^{iT+(l+1)Tc} \sum_{k=0}^{K-1} \sqrt{2P_k} e^{j\phi_k} \sum_i b_i^l \sum_{p=0}^{N-1} a_p^k g(t - iT - pTc - \tau_k)g(t - iT - lTc) dt$$
\[
\begin{align*}
&+ \int_{iT+T_c}^{iT+(l+1)T_c} \hat{n}(t)g(t - iT - lT_c)dt.
\end{align*}
\]

If the chip waveform \( g(t) \) is time limited to one chip interval, \([0, T_c]\), and if in addition without any loss of generality the delays of all the users are restricted to one symbol interval, \( \tau_k \in [0, NT_c] \), then the above equation reduces to

\[
y_{i,l} = \sum_{k=0}^{K-1} \sqrt{2P_k}e^{j\phi_k} \left\{ b_{k}^{i-1}a_{k}^{(N-|\tau_k|+l-1)} \phi_{r,k} \chi_{l\leq|\tau_k|} + b_{k}^{i-1}a_{k}^{(N-|\tau_k|+l)} \phi_{l,k} \chi_{l<|\tau_k|} \right. \\
&\left. + b_{k}^{i}a_{k}^{l-|\tau_k|-1} \phi_{r,k} \chi_{l>|\tau_k|} + b_{k}^{i}a_{k}^{l-|\tau_k|} \phi_{l,k} \chi_{l\geq|\tau_k|} \right\} + n_{i,l},
\]

where,

\[
\begin{align*}
\phi_{l,k} &= \int_{\tau_k-[\tau_k]}^{T_c} g(t)g(t - \tau_k + [\tau_k])dt \\
\phi_{r,k} &= \int_{0}^{iT+(l+1)T_c} g(t)g(t + T_c - \tau_k + [\tau_k])dt \\
n_{i,l} &= \int_{iT+T_c}^{iT+(l+1)T_c} \hat{n}(t)g(t - iT - lT_c)dt,
\end{align*}
\]

and \( \chi_A \) is the indicator function for set \( A \). The quantities \( \phi_{l,k} \) and \( \phi_{r,k} \) depend only on the chip waveform and the fractional part of \( \tau_k/T_c \) and correspond to the left and right portions of a chip waveform when filtered with a matched filter that is offset in time by this fractional part. Note that the restriction that the received chip waveform be limited to a chip interval is not practical in bandlimited systems and hence will invariably result in inter-chip interference. Also note that the use of pulses which satisfy the Nyquist criterion is not sufficient to eliminate this interference because the received-chip match filter is not synchronized to the incoming chip waveforms. Hence, in practice, it is necessary to shape the pulse to have very little energy outside the chip interval such that the inter-chip interference is negligible.
Rewriting the above equation for $y_{i,j}$ using a matrix notation, we obtain

$$y_{i,j} = \left( \begin{array}{c} a_0^{(N-[\tau_0]+i)} \phi_{r,0} \chi \leq [\tau_0] + a_0^{(N-[\tau_0]+i)} \phi_{l,0} \chi \geq [\tau_0] \\ a_0^{(l-[\tau_0]+l-1)} \phi_{r,0} \chi > [\tau_0] + a_0^{(l-[\tau_0]+l-1)} \phi_{l,0} \chi \geq [\tau_0] \\ \vdots \\ a_k^{(N-[\tau_k]+l-1)} \phi_{r,k} \chi \leq [\tau_k] + a_k^{(N-[\tau_k]+l-1)} \phi_{l,k} \chi < [\tau_k] \\ a_k^{(l-[\tau_k]+1-l)} \phi_{r,k} \chi > [\tau_k] + a_k^{(l-[\tau_k]+1-l)} \phi_{l,k} \chi \geq [\tau_k] \\ \vdots \\ a_{K-1}^{(N-[\tau_{K-1}]-1)} \phi_{r,K-1} \chi \leq [\tau_{K-1}] + a_{K-1}^{(N-[\tau_{K-1}]-1)} \phi_{l,K-1} \chi < [\tau_{K-1}] \\ a_{K-1}^{(l-[\tau_{K-1}]-1)} \phi_{r,K-1} \chi > [\tau_{K-1}] + a_{K-1}^{(l-[\tau_{K-1}]-1)} \phi_{l,K-1} \chi \geq [\tau_{K-1}] \\ \end{array} \right) \times \sqrt{2P_0 e^{i\phi_0} b_0^{-1}} \begin{array}{c} \sqrt{2P_0 e^{i\phi_0} b_0^{-1}} \\ \vdots \\ \sqrt{2P_k e^{i\phi_k} b_k^{-1}} \\ \sqrt{2P_k e^{i\phi_k} b_k^{-1}} \\ \vdots \\ \sqrt{2P_{K-1} e^{i\phi_{K-1}} b_{K-1}^{-1}} \\ \sqrt{2P_{K-1} e^{i\phi_{K-1}} b_{K-1}^{-1}} \\ \end{array} + (n_{i,j}).$$

The observation vector $y_i$ may now be written in a matrix notation as:

$$y_i = \left( \begin{array}{c} a_k^{(N-[\tau_k]-1)} \phi_{r,k} + a_k^{(N-[\tau_k])} \phi_{l,k} \\ a_k^{(N-[\tau_k])} \phi_{r,k} + a_k^{(N-[\tau_k]+1)} \phi_{l,k} \\ \vdots \\ \vdots \\ a_k^{(N-1)} \phi_{r,k} \\ 0 \\ 0 \\ \vdots \\ \vdots \\ a_k^{(N-[\tau_k]-2)} \phi_{r,k} + a_k^{(N-[\tau_k]-1)} \phi_{l,k} \\ \end{array} \right) \begin{array}{c} \left( \begin{array}{c} a_0^0 \phi_{r,k} \\ a_0^0 \phi_{l,k} \\ a_0^0 \phi_{r,k} \\ a_0^0 \phi_{l,k} \\ a_0^0 \phi_{r,k} \\ a_0^0 \phi_{l,k} \\ \vdots \\ \vdots \\ a_0^0 \phi_{r,k} \\ a_0^0 \phi_{l,k} \\ \end{array} \right) \end{array}.$$
\[
\begin{align*}
\begin{pmatrix}
\vdots \\
\sqrt{2P_k e^{j\phi_k}} b_k^{i-1} \\
\sqrt{2P_k e^{j\phi_k}} b_k^i \\
\vdots
\end{pmatrix} + 
\begin{pmatrix}
n_{i,0} \\
n_{i,1} \\
\vdots \\
n_{i,N-1}
\end{pmatrix}
= A(\bar{\tau})x_i + n_i.
\end{align*}
\] (2.36) (2.37)

Hence, under some assumptions, the observation vectors obtained in a CDMA system have the desired property that the energy of the vector is concentrated in a subspace of the vector space spanned by the observation vectors. In addition, if we assume that the mapping from \(\bar{\tau}\) to \(A(\bar{\tau})\) is invertible, the time delays of the users can be estimated by solving non-linear optimization problems described in the following sections. Since the estimation of the signal and noise subspaces from the correlation matrix of the observation vectors is only mildly dependent on the relative power levels of the users, the resulting algorithms are near-far resistant to a large degree. Simulations of the algorithm lead to the same conclusion. For additional details of these simulations we refer the reader to Bensley and Aazhang [3].

**Deterministic MUSIC algorithm**

Once a basis for the noise subspace is estimated from the observation vectors, the estimation of the unknown delays is posed as a non-linear optimization problem. The key idea used in formulating this problem is that since the noise subspace is by definition the best estimate of the null space of the matrix, \(A(\bar{\tau})\), the columns of this matrix would have very small components in the noise subspace, as opposed to any other vector. The MUSIC algorithm essentially exploits this property and poses the estimation problem for the time delay for a given user \(k\) as:

\[
\hat{\tau}_k = \arg \min_{\tau_k \in [0, N)} \| a_k(\tau_k)^{\dagger} S_{\perp} \|^2_2,
\] (2.38)
where, the columns of $a_k(\tau_k)$ are the columns of $A(\tau)$ corresponding to user $k$, and $\tau_k$ is the time delay of that user. The formulation essentially estimates the time delay of user $k$ as the value (from the valid range of time delays) that results in the vector with smallest components when projected into the estimated null space.


Maximum Likelihood MUSIC algorithm

The above formulation of the estimation problem ignores the statistical distribution of the null vectors. Under certain assumptions, it has been shown that the projection of any column of the matrix $A(\tau)$ onto the estimated null space vectors results in a random variable that has a Gaussian distribution. This knowledge of the distribution of the vectors allows the use of a maximum-likelihood formulation to estimate the time delay. If the probability distribution that was assumed is indeed the correct distribution, this should result in a better estimate of the user time delay.

Hence, after determining the probability distribution function of the projected vectors, the maximum-likelihood estimate may be formed as follows:

$$\Lambda(a_k()) = -a(N-2K)\ln(a_k()^\dagger W a_k()) - L a_k()^\dagger S a_k() / a_k()^\dagger W a_k()$$

where,

$$\eta = \frac{1}{N-2K} \sum_{k=2K+1}^{N} \sigma_k^2$$

$$W = \eta \left[ \sum_{k=1}^{2K} \frac{\sigma_k^2}{(\eta - \sigma_k^2)^2} v_k v_k^\dagger \right]$$

$$= S \text{ diag} \left( \frac{\eta \sigma_1^2}{(\eta - \sigma_1^2)^2}, \frac{\eta \sigma_2^2}{(\eta - \sigma_2^2)^2}, \cdots, \frac{\eta \sigma_{2K}^2}{(\eta - \sigma_{2K}^2)^2} \right) S^\dagger.$$
Acquisition in multipath channels

In multipath channels, the algorithm remains the same up to the formulation of the estimation problem. Due to the additional complexity of multiple paths, the estimation of the time delays of the $r$ strongest paths is performed in two steps. First, all the possible columns which would form the columns of $A(\tau)$ if the delay of user $k$ were an integer multiple of a chip period, are used to form two matrices $A_k^r$ and $A_k^l$ defined as:

$$A_k^r = \begin{bmatrix}
0 & a_{N-1}^k & a_{N-2}^k & \cdots & a_1^k \\
0 & 0 & a_{N-1}^k & \cdots & a_2^k \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & \vdots \\
\end{bmatrix}$$

$$A_k^l = \begin{bmatrix}
a_0^k & 0 & 0 & \cdots & 0 \\
a_1^k & a_0^k & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
a_{N-1}^k & a_{N-2}^k & \cdots & a_0^k \\
\end{bmatrix}$$

(2.43)  (2.44)

It should be noted that $a_k(\tau_k)$ in the presence of multipath would be a matrix of the form

$$a_k(\tau_k) = (A_k^r w_k \quad A_k^l w_k)$$

(2.45)

where, $w_k$ is a vector of the form:

$$w_k = \sum_{i=0}^{r-1} \alpha_i c(\nu_i^k)$$

(2.46)
and \( \alpha_i \) is the relative strength of the signal along the \( i \)th path and \( c(\nu) \) has the special structure:

\[
c(\nu) = \begin{pmatrix}
0 \\
0 \\
\vdots \\
[\nu] - \nu \\
\nu - [\nu] \\
0 \\
\vdots \\
0
\end{pmatrix}
\]  

(2.47)

In other words, the columns of \( \mathbf{A}(\vec{\tau}) \) corresponding to user \( k \) will be certain linear combinations of the columns of \( \mathbf{A}_k^r \) and \( \mathbf{A}_k^i \).

The first step of the estimation process is to ignore the restriction on the allowed linear combinations and simply determine the linear combination which minimizes energy in the null space. This involves the projection of the matrices \( \mathbf{A}_k^r \) and \( \mathbf{A}_k^i \) onto the null space and then estimation of a vector \( \mathbf{w}_k \in \mathbb{C}^N \) which minimizes the quantity

\[
\| (\mathbf{w}_k^r \mathbf{A}_k^r \mathbf{S}_\perp \quad \mathbf{w}_k^i \mathbf{A}_k^i \mathbf{S}_\perp) \|_F.
\]

Note that the estimated \( \mathbf{w}_k \) obtained in this manner will not in general be of the form described in Equation 2.46. This portion of the estimation requires the computation of the eigenvector corresponding to the smallest eigenvalue. The second step of the estimation procedure is to refine this estimate of \( \mathbf{w}_k \) to obtain a vector in the desired parametric form. This step requires a series of least squares fits. We will describe the implementation details of these algorithms later in this thesis.

Please refer to Bensley and Aazhang [3] for additional details of the algorithms.
2.4 Systolic Algorithms and Architectures

In this thesis we are interested in systolic algorithms and architectures since they scale well and exhibit a high degree of parallelism. Systolic algorithms are characterized by communication which is localized in time and space [43]. This allows design of arrays which communicate externally only at the boundaries. Hence an order of magnitude reduction in communication requirements is achieved. This is a very important property in real-time signal processing systems. In this section we describe the matrix-vector multiplication array which is used as a subarray in more complicated arrays such as the subspace tracking array [50]. In this thesis it is used in a sliding window subspace tracking array and in several architectures which implement the CDMA backend computation. The QR array similarly is used as a subarray in the subspace tracking array, triangular SVD array [46] and others. The Brent, Luk, van Loan SVD array [11] forms the basis for most of the later implementations of Jacobi SVD algorithms. We will discuss each of these algorithms in this section.

2.4.1 Matrix-vector multiplication array

Figure 2.9 shows a matrix-vector multiplication array, where the matrix is statically distributed in the array, while the input vectors are piped into and the result vectors piped out of the array. Multiple vectors may be pipelined on the array without any breaks. Hence, the same array is also a matrix-matrix multiplication array, where one of the matrices is static, while the columns of the other matrix are piped into the array and the columns of the resulting matrix piped out. The structure of this matrix-vector multiplication array is useful whenever one of the matrices is constant (or it is updated in place in a distributed manner), while several different input vectors can be pipelined into the array. The array achieves a throughput of
Figure 2.9: Systolic array for matrix multiplication
$O(1)$ independent of the size of the problem, provided the host to array I/O is scaled linearly with the dimensions of the array.

Fig. 2.9 (a) shows the dataflow for the matrix multiplication $C = B'A$, where

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix},$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}.$$  

Fig. 2.9 (b) details the operation of one of the cells in the matrix multiplication array. The element $a$ is held in the memory of the cell, while $b_{in}$ and $c_{in}$ are inputs to the array in a given cycle. The cell computes the result $c_{out}$ and $b_{out}$ and is input to the neighboring cells in the next cycle.

### 2.4.2 QR updating array

An array to pipeline multiple updates of QR updating is shown in Fig. 2.10 and Fig. 2.11. The array consists of a triangular structure of cells each of which stores one element of the current $R$ factor. The new vectors enter the array from the top skewed one cycle for each position towards the right to satisfy timing dependencies. The diagonal cells compute a Givens rotation to rotate the incoming element $a_{ij}^{(i-1)}$ into the element $r_{ii}$ stored in that cell, and store the updated $r_{ii}$. The Givens rotation is passed on to the processor on the right. The off-diagonal cells use the Givens rotation received from the left to rotate the 2-vector formed by the stored element $a_{ij}$ and the element $a_{ij}^{(i-1)}$ received from the top. The updated $r_{ij}$ element is saved in the cell for
Figure 2.10: *Gentleman-Kung systolic array* [27] for QRD updating
Figure 2.11: Description of the internal cells of the Gentleman-Kung systolic array for QRD updating
the next update while the updated $a^{(i)}_{ij}$ is passed on to the bottom neighbor. The Givens rotation received from the left is passed on unaltered to the processor on the right.

2.4.3 Brent-Luk-Van Loan SVD array

The Brent, Luk and Van Loan systolic array [11] consists of a $p/2 \times p/2$ array of processors (Figure 2.12) to compute the SVD of a $p \times p$ matrix. The matrix dimension, $p$, is assumed to be even. Each processor stores a $2 \times 2$ submatrix and has the ability to compute the SVD of a $2 \times 2$ matrix. The special structure of the Jacobi rotation matrix allows computing the angles for a $p \times p$ two-sided rotation in terms of a basic $2 \times 2$ rotation. The application of the transformation on a $p \times p$ matrix can also be expressed as a set of $2 \times 2$ transformations. Thus the $2 \times 2$ SVD forms the basic step for the $p \times p$ SVD.

A $2 \times 2$ SVD can be described as

$$ R(\theta_l)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix}, \tag{2.48} $$

where $\theta_l$ and $\theta_r$ are the left and right rotation angles, respectively. The rotation matrix is

$$ R(\theta) = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}, \tag{2.49} $$

and the input matrix is

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

The angles $\theta_l$ and $\theta_r$ necessary to diagonalize the matrix $M$ can be shown to be the inverse tangents of the data elements of $M$:

$$ \theta_r + \theta_l = \tan^{-1} \left[ \frac{c + b}{d - a} \right], $$

$$ \theta_r - \theta_l = \tan^{-1} \left[ \frac{c - b}{d + a} \right]. \tag{2.50} $$

The left and right rotation angles in equation 2.7 can be computed using the equations 2.50 on a $2 \times 2$ matrix formed by $\begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix}$. It may be noted that the $p/2$ diagonal
Figure 2.12: The Brent-Luk-Van Loan SVD Array for Computing the SVD of an 8 $\times$ 8 Matrix

Table 2.1: Parallel Ordering data exchange with 8 elements. This is the ordering used when eight chess players have to play a round-robin tournament, each player playing one game a day.

\[
\begin{align*}
(1,2) & \quad (3,4) & \quad (5,6) & \quad (7,8) \\
(1,4) & \quad (2,6) & \quad (3,8) & \quad (5,7) \\
(1,6) & \quad (4,8) & \quad (2,7) & \quad (3,5) \\
(1,8) & \quad (6,7) & \quad (4,5) & \quad (2,3) \\
(1,7) & \quad (8,5) & \quad (6,3) & \quad (4,2) \\
(1,5) & \quad (7,3) & \quad (8,2) & \quad (6,4) \\
(1,3) & \quad (5,2) & \quad (7,4) & \quad (8,6)
\end{align*}
\]
processors store precisely the matrices required for \( p/2 \) successive iterations, corresponding to a single row of parallel ordering in Table 2.1. All the transformations that annihilate the off-diagonal elements in the diagonal processors are independent and hence can be computed in parallel. An update of the entire array requires the left angles to be propagated from a diagonal processor to all the processors on the same row, and the right angle to be propagated to all the processors on the same column. This propagation is systolic. The update of the matrix is followed by a permutation of the rows and the columns of the matrix according to the parallel ordering. An 8-connected mesh is required to perform this data exchange. The permutation results in the next set of sub-matrices, required to compute the new set of angles, to be formed at the diagonal processors. Since the diagonal processors require data only from their diagonal neighbors, new computation can start as soon as these neighbors complete their portion of the computation. Thus, the updates of the matrix, corresponding to separate iterations in equation 2.7 are pipelined. This leads to
diagonal waves of activity originating at the main diagonal and propagating outwards. This is shown in Figure 2.13. The various steps in the operation of this array are listed below:

- The \( p/2 \) diagonal processors compute the required transformation angles using the \( 2 \times 2 \) sub-matrix stored in them.

- Each diagonal processor propagates the left angle, which it computed, to its east and west neighbor. Similarly the right rotation angles are propagated north and south.

- The non-diagonal processors receive the left and right angles and apply the 2-sided transformation to their sub-matrices. After the computation, they forward the left and right rotation angles to the processor next in the row or column.

- Processor P22 in Figure 2.12 receives its next set of data elements from processors P11, P33, P13 and P31. Hence, it has to wait till processors P31 and P13 finish their computations. Similarly, all the other diagonal processors need to wait for their diagonal neighbors to complete the transformations before exchanging data. Exchange of data in different parts of the array is staggered in time.

- After the data exchange, a new set of angles is computed by the diagonal processors and the same sequence of operations is repeated.

Each diagonal processor requires \( (p-1) \) iterations to complete a sweep. The number of sweeps required for convergence is conjectured to be \( O(\log p) \). Such a relationship has been observed in simulations. Since each iteration for a diagonal processor is completed in constant time, the total SVD requires \( O(p \log p) \).

Many systolic algorithms have since been developed for the SVD problem, but they still retain some of the salient features of the array described above. A later refinement of the array using a different ordering, odd-even ordering, results in a triangular array [46]. The SVD updating array by Moonen, van Door-n and Vandewalle [50] integrates the QR decomposition array and the triangular SVD array along other novel ideas. Both these
arrays incorporate the same diagonal data movement described in the Brenk-Luk-van Loan array.

There are a number of other algorithms which are discussed in this thesis. However, a detailed description of all these algorithms is beyond the scope of this thesis. Some of the fundamental ideas behind many of these algorithms and references to the original algorithms will be presented as the need arises throughout the rest of this thesis.
Chapter 3

A Pipelined SVD Algorithm

3.1 Introduction

This chapter describes one of the new systolic algorithms which was developed as part of this thesis. The architecture described in this chapter, the pipelined SVD array, is a solution to an architectural problem arrived at by fundamentally modifying the underlying numerical algorithm. As such this development straddles the architectural and numerical linear algebra areas to arrive at an elegant solution. This array is one of the alternative architectures available to the designer of a CDMA acquisition system as will be described later in Chapter 5.

The new idea used to arrive at this array is a new ordering for the choice of pivots in the cyclic Jacobi method for the parallel computation of the singular value decomposition (SVD) or equivalently the Hermitian eigenvalue decomposition problem. This new ordering possesses some interesting properties which none of the orderings proposed previously in the literature possess. These properties make this ordering preferable over the others in some implementation scenarios. In particular, this new ordering allows multiple SVD problems to be pipelined on a given array because the communication required to load the new problem and unload the results of the previous problems are integrated into the rest of the computation. Many of the earlier systolic algorithms such as the matrix vector multiplication and more complicated arrays such as the Gentleman and Kung QR array allow such seamless integration of I/O with the rest of the computation. However, most of the previous SVD arrays do not allow such integration. The array described in this chapter
solves this problem resulting in an array more in tune with the earlier concept of systolic arrays.

Numerous arrays have been proposed in the past decade for parallel computation of the SVD. The basis of most of these arrays is the cyclic Jacobi method coupled with the systolic paradigm. This results in a powerful algorithm where the entire computation can be performed in a uniform manner and hence is particularly amenable to parallelization either in custom arrays or general purpose parallel machines. In this chapter we assume the reader to be familiar with the sequential cyclic Jacobi algorithm [25] and any of the parallel variants [11]. Since the literature on this problem is extensive we will refer only to those papers which are relevant to the problem at hand and refer the reader to the many papers on this problem for a tutorial introduction.

Some of the earliest algorithms developed for parallel SVD perform the computation in-core, for instance the Brent and Luk (BL) array [10]. By “in-core” we mean that these algorithms assume the input matrix is already available distributed among the processors and leave the resulting matrices distributed. These algorithms are also the ones typically implemented on general purpose parallel machines, either the one-sided variant [5] or the two-sided variants [48]. The cyclic Jacobi method allows a lot of flexibility in the order in which pivot pairs, viz. the next pair of off-diagonal elements to be annihilated, may be chosen. All the variants of the Jacobi method we find in the literature are a direct result of this flexibility. Some of the reasons to look for alternative orderings are to reduce communication, increase the number of computations that can be performed in parallel, and provide uniformity and ease of implementation of the orderings. Given the in-core assumptions, Gao and Thomas [26] showed the lower bounds that can be achieved on the first two parameters, namely the number of parallel cycles of computation and the number of messages required for communication in any parallel version of the cyclic Jacobi method. They also proposed an ordering which achieves these lower bounds on a hypercube interconnection. Other orderings have been proposed to achieve these minimum bounds for the ring [5, 22] and
torus [18] topologies using bi-directional communication links between adjacent processors. When these algorithms are implemented as special-purpose VLSI processors, special cases in the orderings tend to add many additional states to the controllers, resulting in larger and slower controllers [41]. From that point of view it appears that these new orderings would require more complex controllers than the BL ordering. A variant of the ordering we propose in this chapter can achieve the lower bounds and result in simpler controllers. However, in this chapter we are interested in a slightly different problem, namely the implications of relaxing the in-core assumption on the cyclic Jacobi algorithm. In particular, when a large number of matrices need to be continuously decomposed, the time to load and unload the matrices may become significant.

There do exist previous SVD algorithms which do not make the in-core assumption for the input matrices but leave the results in-core, e.g. the triangular array by Luk [46], the SVD updating array of Moonen, van Dooren and Vandewalle [49]. These arrays relax the in-core assumption by performing an initial QR followed by the Jacobi SVD. There are other arrays such as the ones by Finn [23] and Schimmel [55] which do not make the in-core assumption and let the results flow out of the array systolically. Finn showed that his array exhibits empirically observed linear convergence. Schimmel's array implements the one-sided Jacobi method but does not exploit all the parallelism available. None of the aforementioned methods utilize the flexibility afforded by the cyclic Jacobi method in the choice of the ordering to relax the in-core assumption. In this chapter we present a new ordering which allows the implementation of the cyclic Jacobi method without the in-core assumption. Hence matrices are loaded as computation proceeds and the results flow out of the array in a seamless manner resulting in an array closer to the traditional concept of a systolic array. We develop our result for a one-sided Jacobi method and show that this results in a family of architectures with varying throughput-hardware tradeoffs. In particular the various members of the family correspond to varying degrees of unrolling.
the sweeps. The resulting array is also surprisingly uniform and simple, requiring only unidirectional communication links. We also describe a few applications of our method.

3.2 Pipelining on linear arrays

In order to focus on the ordering aspects of linear arrays, we abstract the arrays previously described in the literature as follows. These arrays consist of \( n/2 \) processors, typically interconnected in a ring with bi-directional links. Each processor holds the data of two columns of the matrix. Computation proceeds in timesteps, which we call subsweeps. Each subsweep begins with processors exchanging data using some distributed algorithm which implements the desired ordering. Following this the processors compute a Givens rotation to orthogonalize the two columns and thus orthogonalize the columns (see Sec. 2.1.5). We omit the details of this update since they have been described numerous times previously [10, 22] and such detail is not necessary to understand the problem at hand. We will assume the Hestenes method is being implemented, since replacing it with the other one-sided method by Eberlein and Park does not affect any of our conclusions. There are a number of possible scenarios for host to array I/O. Since all of them result in the same conclusions we assume the host to array I/O is through the leftmost processor and the array to host I/O is through the rightmost processor. In this scenario we assume that the array operates in a special host I/O mode to load new matrices or unload the results of the previous computation. In the I/O mode the processors receive data from their left neighbor and either save it or pass the data to the right neighbor if that data is not intended for them. The array operates synchronously in either the SVD mode, or the host I/O mode.

The array which we propose in this chapter is based on an ordering which is a variant of the odd-even ordering first described by Stewart [62]. In fact, our ordering may be considered as the odd-even ordering coupled with additional data movement to allow a simple implementation on a distributed memory machine. Typically the odd-even ordering
Figure 3.1: A completely unrolled linear array has $nS/2$ processors, where $n$ is the number of columns in the matrix and $S$ is the number of sweeps, both of which are restricted to be even. The resulting array achieves the highest throughput for this family of architectures.
Figure 3.2: Snapshots of a completely unrolled linear array in operation for \( n = 6 \). We show only two of the sweeps being performed since the figure repeats for the rest of the sweeps. We define a subsweep as the time to compute an angle, update the columns stored in the array and communicate with the neighboring processors. A new problem is loaded and a new solution is obtained every 6 subsweeps. A new column is loaded and unloaded every subsweep, once the pipeline is full. Exceptions to normal operation, due to the presence of an on token associated with the first column in the processor, is indicated in this figure by oval boxes for the processors. Presence of an on token is indicated in boldface. The numbers in the boxes indicate the numbers associated with the data in the columns when they were first introduced into the array. The figure shows that all the pairings required to complete a sweep indeed occur.
in previous work is described using the notion of a combined distributed-shared memory configuration [46, 49, 62]. We prefer a fully distributed memory implementation.

We define our processor array interconnected as shown in Fig. 3.1. The array consists of \(nS/2\) processors, where \(S\) is the number of sweeps which need to be performed for convergence and is constrained to be an even number. The processors are connected left to right using uni-directional links. The host to array I/O takes place through the left communication link of the leftmost processor and the array to host I/O takes place from the right communication link of the rightmost processor. On such an array it is possible to pipeline \(S\) independent problems at any time.

We find it easier to describe the operation of the array with an example shown in Fig. 3.2. The example is for a matrix with 6 columns. The figure illustrates the operation of the array for only 2 sweeps, but certainly more sweeps would be required for convergence. Each of the boxes in the figure represents a processor, with the numbers in each box \([c_1, c_2]\) representing the two columns being processed in the processor. The number for any column represents the original column number when it was introduced in the array. The vertical axis represents time and shows the snapshots of the array after each timestep starting with timestep 0. At each timestep a new column is introduced into the array from the left end of the array. The columns are introduced in the order 6, 5, 4, 3, 2, 1 in this example. However, the columns could have been introduced in any order and this does not affect the rest of the discussion. Along with the data, a single bit of information, which we call a token, is associated with each column. The tokens are denoted as \(t_1\) and \(t_2\) in Fig. 3.1. The token could be either on or off, to differentiate between columns of different matrices. When we say a column has moved to some processor we mean that the data and the token associated with that column have moved. We initially associate an on token with column 6 and an off token with all other columns. The token is used as a control mechanism.

The functionality of the processors is very simple. Each timestep consists of an external data exchange, computation and an internal data exchange in that order. The external
data exchange essentially consists of shifting the columns (both the data and the token) one position to the right. Hence external data exchange for a processor involves the following steps:

1. Passing the second column $c_2$ to the processor on the right,

2. Shifting the first column $c_1$ to become the second column and

3. Receiving the column from the processor on the left to form the new first column.

The next two steps, i.e. the computation and the internal data exchange are conditional on the token associated with the first column in the processor after the external data exchange. If the token associated with the first column is off, a Givens rotation is computed to orthogonalize the two columns. Note that there are two possible rotations that orthogonalize the columns, namely the inner rotation and the outer rotation [25]. It is well-known that the outer rotation may be regarded as an inner rotation followed by a permutation [62]. If the processor uses inner rotations to transform the two columns, this should be followed by an explicit exchange of the data in the two columns to complete the timestep. This internal data exchange exchanges only data between columns, not the tokens. If the outer rotation is used, then no explicit internal data exchange is required.

On the other hand, if the token associated with the first column is on, then no computation or internal data exchange is required for that timestep. As can be verified from Fig. 3.2, this rather simple sequence of operations performed synchronously by every processor results in an ordering in which every pivot pair occurs and hence if the number of processors is large enough, a sufficient number of sweeps are performed to result in a matrix with orthogonal columns. In addition if the number of processors is some even multiple, $S$, of $n/2$, then exactly $S$ sweeps would be performed on the input matrix $A$, and the columns streaming out of the rightmost processor would be columns of the resultant matrix $A_{nS/2}$ as defined in Eq. 2.9. At timestep 6 in this example, a new problem could be introduced into the array, thus resulting in pipelining multiple SVDs on the array. If for some reason
a new problem is not introduced, it is still necessary to introduce an on token in timestep 6 to indicate the end of the current matrix. Thus if $S$ sweeps are required for convergence, then the use of $nS/2$ processors results in an array in which $S$ problems may be pipelined, resulting in a throughput corresponding to a new problem every $n$ timesteps.

It is easy to verify that the resulting ordering is equivalent to the cyclic-by-rows ordering, where equivalence is the same notion used by Hansen [36] and later by Luk and Park [47], Shroff and Schreiber [57]. For example, from Fig. 3.2 the sequence of rotations could be represented as follows:

$$V = V_{56}V_{46}V_{36}V_{26}V_{16}V_{16}V_{15}V_{14}V_{13}V_{12}V_{65}V_{64}V_{63}V_{62} \cdots$$

$$\equiv V_{56}V_{46}V_{36}V_{26}V_{16}V_{15}V_{14}V_{13}V_{12}V_{65}V_{64}V_{63}V_{62} \cdots$$

$$\equiv V_{56}V_{46}V_{36}V_{26}V_{16}V_{15}V_{14}V_{13}V_{12}V_{65}V_{64}V_{63}V_{62} \cdots$$

$$\equiv (V_{56}V_{46}V_{36}V_{26}V_{16}V_{15}V_{14}V_{13}V_{12})(V_{65}V_{64}V_{63}V_{62} \cdots$$

The Jacobi method using cyclic-by-rows and cyclic-by-columns orderings has been proven to converge in a seminal paper by Forsythe and Henrici [25] and hence by equivalence, our algorithm is guaranteed to converge. We have included a sketch of a formal proof of the above statements in Sec. 3.3.

### 3.2.1 Throughput-Hardware tradeoffs

The array described in the previous section is the completely unrolled array, where all the sweeps have been unrolled. This allows new problems to be introduced into the array, one every $n$ subsweeps. Hence an $S$-fold improvement in throughput is obtained over an array with $n/2$ processors. However if the throughput requirements are lower, it is possible to lower the hardware cost. This is done by mapping the logical array onto a physical array with fewer processors.
Figure 3.3: A linear array where all the sweeps have not been unrolled. The idea is to reuse portions of the unrolled array at the expense of the ability to load new problems. Depending on the size of the array it may be possible to compute several problems in a pipelined fashion.
There are two classical approaches to such mapping, namely the **globally serial locally parallel (GSLP)** approach and the **globally parallel locally serial (GPLS)** approach. Suppose we want to map the logical array onto a physical array of \( q \) processors. In the GSLP approach, one divides the logical array into blocks with \( q \) logical processors in each block. The logical array is then mapped onto a physical array of \( q \) processors by mapping each of these blocks onto the physical array. Successive blocks get mapped onto the same hardware. In the GPLS approach one divides the logical array into \( q \) blocks and each block gets mapped onto a single processor of the physical array.

### Globally serial - locally parallel mapping

A GSLP mapping of the pipelined SVD array is possible by reusing processors with a feedback connection that allows the matrix to recirculate in the array. In particular, smaller arrays with \(nz/2\) processors are possible, where \(z\) is a design parameter and can be chosen to be any integer factor of \( S \), the number of sweeps required for convergence. In a linear array with \(nz/2\) processors, it is possible to pipeline \( z \) problems at a time. New sets of \( z \) problems get introduced into the array once every \( nS \) subsweeps. Similarly, solutions to \( z \) problems are obtained every \( nS \) subsweeps. However, the price paid for this is additional control complexity. Figure 3.3 shows an array, where switches are necessary at the boundaries to allow either the feedback connection or the connection to the host. The input switch allows connection to the external host for the first \( nz \) subsweeps followed by the feedback connection for the remaining subsweeps. The output switch feeds back the data and the token for the first \( n(S-z) \) subsweeps and enables the connection to the external host for the last \( nz \) subsweeps. The special case of an array with \( z = 1 \) is always possible. We now have a family of processor arrays starting with \( n/2 \) processors all the way to \( nS/2 \) processors for higher throughputs. Of course, it is possible to view this mapping itself as a logical array and then map it onto a smaller array using either the GSLP or the GPLS
approach, though such an exercise would be driven by actual technology constraints and
the merits of which are not obvious in a technology-independent sense.

Globally parallel - locally serial mapping

The GPLS approach is simpler to implement and requires several successive processors of
the logical array to be mapped onto a single physical processor. The resulting array will
be look like the unrolled pipelined SVD array, i.e. a linear array of processors with uni-
directional links, with new columns entering the array from the left and the old results
piped out of the array from the right. The only difference is that the number of proces-
sor is determined by practical technology constraints and the amount of memory within
the processor is determined from such considerations. In the GPLS approach all the com-
munication that takes place within a block would be localized to lie within the processor
and hence this approach has the advantage that the overall communication requirements
are reduced. Except when $NS/2q$ is an integer the last processor in the system would be
underutilized since it would have fewer columns to process. This need not necessarily be a
disadvantage, since it is conceivable that a practical system will have to handle exception
conditions and the additional time in the under-utilized processor can be used to perform
the associated control tasks. We will discuss the merits and demerits of the two approaches
later in Section 5.5.3.

3.2.2 Comparison with in-core methods

So what did we gain from this new ordering? At first glance it appears to take more
subsweeps (i.e. parallel timesteps) for each SVD and hence more time to compute each
SVD than other methods previously proposed. However, there is an implicit assumption
made in such a comparison: it ignores the time taken to load the array from and unload
the array to the host processor. Thus if a large number of SVDs are to be performed, this
time needs to be accounted.
In order to compare our array with previous arrays, let us assume that the time required to transfer one column from one processor to the next is \( n\alpha \), and the time to compute an outer angle and apply it to two columns is \( n\beta \). In a linear array there is no need to scale the communication ability of the host, quantified by \( \alpha \), with the problem size, unless the computational speed \( \beta \) is also scaled. This is because the time to load the array would be \( O(n^2) \), while computation of the SVD takes approximately \( O(n^2\log n) \). The same is not true of the square arrays. In square arrays the communication capacity of the host has to scale with problem size, even when the processors' computation speed is kept a constant.

Let us first compute the time required for an SVD using a typical algorithm that makes in-core assumptions. Assuming an optimal ordering as defined by Gao and Thomas [26], the number of subsweeps required for each sweep would be \( n(n - 1)/2 \times 1/[n/2] \). Each subsweep involves \( n\beta \) time for computation and \( n\alpha \) for communication. In addition we need \( n^2\alpha \) to load the array and \( n^2\alpha \) to unload the array. However, if we assume the existence of a forward link and a reverse link from the host to the array, then the unloading of one problem can be overlapped with the loading of another problem. Hence the amortized time for each SVD is:

\[
T_{\text{in-core}} = n^2\alpha + \frac{n(n - 1)}{2} \times \frac{1}{[n/2]} n(\alpha + \beta) \times S
\]

\[
= \begin{cases} 
  n^2\alpha + (n^2\alpha + n^2\beta) \times S & \text{if } n \text{ odd} \\
  n^2\alpha + (n - 1)(n\alpha + n\beta) \times S & \text{if } n \text{ even.}
\end{cases} 
\tag{3.1}
\]

Using the same numbers let us now compute the time required for the SVD using the array we propose in this chapter. To perform \( S \) sweeps it takes \( n(S + 1) \) subsweeps. However, the last \( n \) subsweeps overlap with those for the next SVD. Since each subsweep takes \( n\alpha + n\beta \) time, the amortized time for an SVD is:

\[
T_{\text{pipe-svd}} = (n^2\alpha + n^2\beta) \times S. 
\tag{3.2}
\]

In a compute limited system, if \( n \) is even, the in-core assumption could result in a faster implementation. However, the chief benefit of the new array is when the system is I/O
limited. In this case, since $S$ is $O(\log n)$ and in practice always less than $n$, the proposed array actually computes the SVD faster. In addition, it does not require the links between processors to be bi-directional. The processor descriptions are all the same and hence the proposed array should result in cheaper hardware. Quantitatively, the condition under which the proposed array would be faster than one that makes in-core assumptions is

$$\frac{\alpha}{\beta} > \begin{cases} \frac{S}{n-S} & \text{if } n \text{ even} \\ 0 & \text{if } n \text{ odd.} \end{cases}$$

(3.3)

Since $S/(n - S)$ tends to zero as $n \rightarrow \infty$, for any given set of technology constraints parameterized as $\alpha/\beta$, there exists $n_0$ such that for any $n > n_0$, the pipelined array will be faster.

### 3.3 Convergence of the new ordering

In this section, we sketch some of the important steps in formally proving the properties of the ordering we introduced in this chapter. Before we start, here are a few definitions. At any given time index, the position, $\mathcal{P}(c)$, of a column $c$ denotes its position in the array at that time instant. The position is an integer such that $0 \leq \mathcal{P}(c) < nS$. Thus column 1 in the leftmost processor would be numbered 0, while column 2 would be numbered 1, column 1 in the processor to the right of the leftmost processor is 2 and so on. As described earlier, the subsweeps are numbered starting from 0 and increasing in time.

For a column $c$, we define the function $\Psi(c)$ to be the time index of the subsweep at which it was introduced into the array. The function $\Omega(c)$ is the index of the latest subsweep, including $\Psi(c)$, at which an on token was introduced into the array, i.e. the time index at which a column from the current matrix was first introduced into the array. We make this distinction since the data from many matrices can simultaneously reside in the systolic array at any given time. For any column $c$, we also define the notion of a relative subsweep. A subsweep $t$ is defined to be the relative subsweep $t - \Omega(c)$ for the column $c$.

The following facts may be easily verified:
PROPOSITION 3.1 A column associated with an off token when present in an even position in the array will always move two positions to the right in one subsweep. Hence that column will continue to remain in an even position as long as the condition required for this proposition is satisfied.

PROPOSITION 3.2 A column $c$ in an odd position in the array at some subsweep will remain in the same position for the next subsweep if the token associated with the column in position $\mathcal{P}(c)-1$ is off. Thus the column remains stuck in that position as long as the condition required for this proposition is satisfied.

PROPOSITION 3.3 An on token always moves right by one position in one subsweep under any conditions.

The following theorem gives an exact relation for the position of a column $c$ at any subsweep.

PROPOSITION 3.4 The position of a column $c$ at some relative subsweep $k$ is given by

$$
\mathcal{P}_k(c) = \begin{cases} 
2(k - p) - 2jn & \text{if } \max(2p + (2j - 1)n, p - 1) < k \leq 2p + 2jn \\
2p + 1 + 2jn & \text{if } 2p + 2jn < k \leq 2p + (2j + 1)n,
\end{cases}
$$

(3.4)

where $p \equiv \Psi(c) - \Omega(c)$ is the relative subsweep at which the column $c$ was introduced into the array. The quantity $j$ is an integer that satisfies either the constraints

$$
\max(2p + (2j - 1)n, p - 1) < k \leq 2p + 2jn
$$

or the constraints

$$
2p + 2jn < k \leq 2p + (2j + 1)n.
$$
Proof This proposition may be verified by induction using the Propositions 3.1, 3.2, 3.3.

From Proposition 3.4, the following corollaries can be derived.

Corollary 3.1 Every pair of columns $a$ and $b$ which are introduced into the array such that $\Omega(a) = \Omega(b)$ will get paired at every subsweep $t$ that satisfies

$$t - [\Psi(a) + \Psi(b) - \Omega(a)] \equiv 0 \mod n.$$ 

To show equivalence of this ordering with the cyclic-by-rows ordering, we restate a result shown by Shroff and Schreiber [57] about cyclic wavefront orderings. In the same paper they also show that cyclic wavefront orderings are equivalent to cyclic by rows ordering.

PROPOSITION 3.5 In a cyclic ordering $O$ of the pairs $\{(g, h), 1 \leq g < h < n\}$, let $I(g, h)$ be the index at which pair $(g, h)$ occurs. If

$$I(g, h - 1) < I(g, h) < I(g + 1, h)$$

for all $1 \leq g < h \leq n$ then $O$ is called a cyclic wavefront ordering.

PROPOSITION 3.6 The ordering described by Eq. 3.4 is equivalent to a cyclic wavefront ordering, and is hence equivalent to the cyclic-by-rows ordering.

Proof Choose two arbitrary columns $a, b$ such that $\Omega(a) = \Omega(b)$ and $\Omega(a) \leq \Psi(a) < \Psi(b) \leq \Omega(a) + n$. Let $b_{-1}$ be the column introduced into the array immediately before $b$ and $a_{+1}$ be the column introduced into the array immediately following $a$. Then Proposition 3.5, can be verified using corollary 3.1, i.e. it can be verified that the pairs $(a, b_{-1})$, $(a, b)$ and $(a_{+1}, b)$ occur in that order in a given sweep. It can also be verified that $(a, b_{-1})$ from the next sweep will not occur until $(a, b)$ from the current sweep is completed. Similar precedence constraints are satisfied by other pairs. These two conditions are sufficient to guarantee equivalence of the ordering to cyclic wavefront orderings.
3.4 Summary

The linear pipelined array we have proposed in this chapter is developed at a sufficiently high level that other ideas for low level pipelined implementations, such as the various methods for bit-level pipelining [20, 52], can be easily incorporated. The pipelining idea results in a very simple array which should considerably ease practical implementations. Theoretically, we found that for large problems the pipelined array never performs worse than arrays which make in-core assumptions. The host I/O requirements are simple and they match the way data is obtained in subspace tracking problems. The array should result in a cost-effective solution in subspace tracking applications which do not have very stringent latency requirements.
Chapter 4

Sliding Window Subspace Tracking

4.1 Introduction

We now describe another new algorithm applicable in general in subspace-based signal processing systems. The focus of this chapter is a sliding window subspace tracking algorithm, an architecture to implement it, and an analysis of its numerical properties. Sliding window subspace tracking involves the update of orthonormal bases for the signal and/or noise subspaces of the received observations using the L most recent observations. The goal of this chapter is to derive a numerically stable and computationally efficient algorithm to update/downdate the previous estimate of the basis when a new observation is received. In this chapter we extend methods used for the Cholesky downdating problem to this SVD problem, resulting in a sliding window subspace tracking algorithm. The resulting architecture is very similar to the exponential downdating algorithm for subspace tracking by Moonen, Van Dooren and Vandewalle [49]. An important result derived in this chapter is that a hybrid system using floating-point for some critical values and fixed-point for all the other computations enjoys many of the stability properties enjoyed by downdating algorithms implemented completely in floating-point.

4.2 Background

4.2.1 Cholesky Downdating Algorithms

An intermediate step of the sliding window SVD algorithm derived in this chapter is Cholesky (or QR) downdating. Hence we will present some background on the stability
properties of the algorithms which implement Cholesky downdating. We are primarily interested in downdating algorithms which have a structure similar to updating, are numerically stable and are parallelizable.

We will consider three algorithms for Cholesky downdating. We call them the hyperbolic downdating technique, the Linpack algorithm [21] and the Chamber's algorithm [14]. Of these algorithms, the hyperbolic rotations algorithm (which is a naive implementation of hyperbolic rotations) exhibits undesirable numerical properties. A variation of this algorithm, the Chamber's algorithm, does not suffer from the same drawbacks. The Linpack algorithm exhibits numerical properties similar to the Chamber's algorithm [61]. The Linpack algorithm, however, is difficult to pipeline due to a backward dependence which requires frequent breaks in the pipeline and hence is not a good candidate for a parallel implementation. The hyperbolic downdating and the Chamber's algorithms on the other hand may be parallelized to result in algorithms very similar to the QR-decomposition systolic algorithm by Gentleman and Kung [27]. It has been shown by Stewart [61] that Chamber's algorithm possesses the desirable relational stability property of the Linpack algorithm. In addition, since this algorithm exhibits the structure found in the QR decomposition algorithms, the exponential downdating algorithm of Moonen [49] can be readily extended to a sliding window subspace tracking algorithm with Chamber's algorithm used as an intermediate step.

4.2.2 The Hyperbolic Singular Value Decomposition

The idea of hyperbolic rotations is based on the notion of *pseudo-orthogonal rotations*. The other term used in literature to describe this notion is *H*-orthogonal [28]. A *H*-orthogonal matrix is a consequence of the definition of an indefinite scalar product. Given a Hermitian matrix *H*, the indefinite scalar product is defined as *z'* *H*y, where *x* and *y* are vectors. This can be seen to be an extension of the definite scalar (or inner) product *z'* *y*. An important distinction is that unlike the usual scalar product, the quantity *z'* *H*x is not
constrained to be greater than or equal to zero, and hence the term “indefinite” in the indefinite scalar product. Extending the usual notion of orthogonality, two vectors are considered $H$-orthogonal when their indefinite scalar product equals zero. $H$-orthogonal matrices may be defined in a similar manner: a matrix $U$ is $H$-orthogonal when $U'HU$ equals $H$. It may be easily seen that when $H$ equals the identity matrix, all these definitions reduce to the usual definitions. A general theory has been developed based on the notion of indefinite scalar products [28].

In this thesis we will be interested in the special case when the Hermitian matrix $H$ takes on a special structure, a diagonal matrix with entries on the diagonal drawn from the set \{1, -1\}. The usual hyperbolic counterpart of the Givens rotation in $\mathbb{R}^{2 \times 2}$, defined as

\[
\begin{pmatrix}
\cosh x & -\sinh x \\
-\sinh x & \cosh x
\end{pmatrix}
\]

for any $x \in \mathbb{R}$ may be easily verified to be $H$-orthogonal where

\[
H = \begin{pmatrix}
1 & 0 \\
0 & -1
\end{pmatrix}.
\]

Based on these ideas a counterpart to the usual Singular Value Decomposition (SVD) has been developed, called the Hyperbolic Singular Value Decomposition [53]. The hyperbolic SVD of a matrix $A$, defined as:

\[
A = W\Sigma V'
\]

exists under certain conditions. In this decomposition $W$ is $H$-orthogonal with respect to a signature matrix $H$, $\Sigma$ is a diagonal matrix of singular values and $V$ is an orthogonal matrix.

Another useful result we will use in the derivation is that a matrix which consists of a $2 \times 2$ hyperbolic rotation embedded in an identity matrix is also pseudo orthogonal. For
instance, if $U(m, n, x)$ is defined as

$$
\{U(m, n, x)\}_{ij} = \begin{cases} 
\cosh x & \text{if } i = j = m \text{ or } i = j = n \\
-\sinh x & \text{elsif } i = m, j = n \text{ or } i = n, j = m \\
1 & \text{elsif } i = j \\
0 & \text{otherwise}
\end{cases},
$$

where $m < n$, then the matrix $U(m, n, x)$ is $H$-orthogonal where $H$ is

$$
\{h_{ij}\} = \begin{cases} 
-1 & \text{if } i = j = n \\
1 & \text{if } i = j \neq n \\
0 & \text{otherwise}
\end{cases}.
$$

### 4.3 Sliding window subspace tracking algorithm

The sliding window subspace tracking problem is to update the SVD of $Y_{p-1}$ to the SVD of $Y_p$, where $Y_p$ is defined recursively as:

$$
\begin{pmatrix} 
Y'_{p-L} \\
Y_p
\end{pmatrix} = \begin{pmatrix} 
Y_{p-1} \\
Y'_p
\end{pmatrix},
$$

where $L$ is the window length and $y_p$ is the observation vector received at the $p^{th}$ time step. Rearranging this equation, we get

$$
Y'_pY_p = Y'_{p-1}Y_{p-1} + y_py'_p - y_{p-L}y'_{p-L}.
$$

(4.2)

Since

$$
Y'_pY_p = y_{p-L+1}y'_{p-L+1} + y_{p-L+2}y'_{p-L+2} + \cdots + y_py'_p,
$$

$Y'_pY_p$ is positive definite for all $p$. This implies that the matrix $\hat{Y}'_pH\hat{Y}_p$ is positive definite, where we define:

$$
\hat{Y}_p = \begin{pmatrix} 
Y_{p-1} \\
Y_p \\
Y'_{p-L}
\end{pmatrix},
$$

(4.3)

and

$$
H = \begin{pmatrix} 
I_{L \times L} & 0 \\
0 & -1
\end{pmatrix}.
$$

(4.4)
Hence, there exists a hyperbolic singular value decomposition of the matrix $\hat{Y}_p$ with a signature matrix $H$. From the above relations it follows that

$$Y_p^* Y_p = \hat{Y}_p^* H \hat{Y}_p.$$  

(4.5)

If the SVD of $Y_p$ is defined as $U_p \Sigma_p V_p'$ and the hyperbolic SVD of the corresponding $\hat{Y}_p$ is given by $\hat{W}_p \hat{\Sigma}_p \hat{V}_p'$, where $\hat{W}_p$ is $H$-orthogonal with $H$ defined as in Eq. 4.4, then it follows that

$$V_p \Sigma_p^2 V_p' = \hat{V}_p \hat{\Sigma}_p^2 \hat{V}_p'.$$

(4.6)

From the uniqueness of the SVD it follows that the basis for the row space of $Y_p$ may be obtained from the hyperbolic SVD of $\hat{Y}_p$.

### 4.3.1 An algorithm for hyperbolic SVD updating

The development of the hyperbolic SVD updating in this section follows the corresponding development of the SVD updating technique by Moonen, Van Dooren and Vandewalle [49]. The algorithm updates only an upper triangular matrix $\tilde{R}_i$ and an orthogonal matrix $\tilde{V}_i$, since these are the only quantities required in this problem. $\tilde{R}_i$ is an upper triangular approximation of $\tilde{\Sigma}_i$ in a sense defined by Moonen, van Dooren and Vandewalle [49].

The updating algorithm may be derived as follows:

$$\tilde{Y}_i = \begin{pmatrix} y'_{i-1} \\ y'_i \\ y'_{i-L} \\ U_{i-1} R_{i-1} V'_{i-1} \end{pmatrix} = \begin{pmatrix} y'_i \\ y'_{i-L} \\ U_{i-1} \begin{pmatrix} 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{i-1} \\ y'_i V_{i-1} \\ y'_{i-L} V_{i-1} \end{pmatrix} \right).$$

(4.7)

(4.8)

(4.9)
Figure 4.1: A parallel architecture to implement sliding window SVD

\[
W_{i-1} \begin{pmatrix}
R_{i-1} \\
y'_i V_{i-1} \\
y'_{i-L} V_{i-1}
\end{pmatrix} = V'_{i-1}.
\]

(4.10)

It is easily verified that \( W_{i-1} \) is orthogonal and in addition, \( H \)-orthogonal. Since the product of \( H \)-orthogonal matrices is again \( H \)-orthogonal, it follows that the hyperbolic SVD updating problem now reduces to computing the hyperbolic SVD of \( [R'_{i-1} \ V'y_{i+1} \ V'y_{i-L}] ' \). It should be noted that in the above derivation the matrices \( U_{i-1} \) and \( W_{i-1} \) are implicit and are not required explicitly to update the decomposition.

The matrix, \( [R'_{i-1} \ V'y_{i+1} \ V'y_{i-L}] ' \), has the structure of an upper triangular matrix with two additional rows at the bottom. The first step is to reduce this matrix to an upper triangular matrix through a \( H \)-orthogonal transformation on the left. This is very similar to the QR updating problem. The updating may be performed through either \( 2 \times 2 \) Givens rotations followed by \( 2 \times 2 \) hyperbolic rotations, or through \( 3 \times 3 \) hyperbolic Householder transformations. If the required \( H \)-orthogonal transformation to annihilate the bottom two non-zero rows is called \( H_i \), then the product \( W_i H_i \) is still \( H \)-orthogonal. This step is followed
by a constant number, say r, steps of the Jacobi method using the odd-even ordering, where 
r is a design parameter used to control how close $R_i$ is to the diagonal matrix $\Sigma_i$. The choice 
of odd-even ordering preserves the upper triangular property of the matrix, while driving it towards a diagonal matrix. All these steps may be conveniently pipelined on an array without any backward dependences, resulting in an $O(n^2)$ update algorithm which can be easily parallelized to use $O(n^2)$ processors to result in $O(1)$ throughput. Algorithm 4.1 describes the algorithm used to perform the required updates. Fig. 4.1 shows a parallel architecture to pipeline the various steps of the above computations. It is obtained through a few straightforward modifications to the exponential window SVD array described by Vanpouck, Moonen and Deprettere [68].

Algorithm 4.1 SlidingWindowUpdate

function $[ss, vv] = hsvdud(s, v, a, b, r)$

$[m, n] = \text{size}(s)$;
$a = v' * a$;
$b = v' * b$;

% Use Givens rotations to update row "b"
for i=1:n
    $g = \text{givens}(s(i,:), b(i))'$;
    $\text{tmp} = g*[s(i,:); b']$;
    $s(i,:) = \text{tmp}(1,:)$;
    $b = \text{tmp}(2,:)'$;
end

% Use hyperbolic rotations to downdate row "a"
for i=1:n
    $h = \text{dgivens}(s(i,:), a(i))'$;
    $\text{tmp} = h*[s(i,:); a']$;
    $s(i,:) = \text{tmp}(1,:)$;
    $a = \text{tmp}(2,:)'$;
end

% Perform "r" Jacobi steps using odd-even ordering

for i=1:r
for j=1:2:n-1
    p = [j:j+1];
    [row,col] = jacobi(s(p,p),1);
    s(p,:) = row'* s(p,:);  
    s(:,p) = s(:,p) * col;
    v(:,p) = v(:,p) * col;
end

for j=2:2:n-1
    p = [j:j+1];
    [row,col] = jacobi(s(p,p),1);
    s(p,:) = row'* s(p,:);  
    s(:,p) = s(:,p) * col;
    v(:,p) = v(:,p) * col;
end

ss = s;
vv = v;

end

4.3.2 Computation of hyperbolic rotations

The manner in which the hyperbolic rotations (or the hyperbolic Householder transformations) are computed is extremely important for the numerical stability of the above algorithm. Hence we will concentrate on this computation in great detail. The limitations of the downdating problem and the algorithms to implement downdating the Cholesky decomposition have been studied in great detail by Stewart [61], Bojanczyk, Brent, Van Dooren and De Hoog [9] and Alexander, Pan and Plemons [1]. The general conclusion is that the stability of the downdating problem is inherently limited due to the sensitivity of the problem to small perturbations when the downdated matrix has a large condition number. This is quantified by the following theorem [61]:

**Theorem 4.1** Let $A$ be positive definite, and let $\tilde{A} = A + H$, where $H$ is symmetric. Then for all sufficiently small $H$, $\tilde{A}$ is positive definite. If $R$ is the
Computation of a hyperbolic rotation, $\Psi_2^{12}$, such that \[
\begin{pmatrix}
\alpha \\
\beta
\end{pmatrix} = \Psi_2^{12} \begin{pmatrix}
\cdot \\
0
\end{pmatrix}
\]

\[
c = \sqrt{(\alpha - \beta)(\alpha + \beta)/a}
\]

\[
s = \beta/a
\]

\[
\Psi_2^{12} = \frac{1}{c} \begin{pmatrix}
1 & -s \\
-s & 1
\end{pmatrix}
\]

Application of $\Psi_2^{12}$ to compute the vector \[
\begin{pmatrix}
\gamma \\
\delta
\end{pmatrix} = \Psi_2^{12} \begin{pmatrix}
\alpha \\
\beta
\end{pmatrix}
\]

\[
\gamma = (\alpha - s\beta)/c
\]

\[
\delta = (\beta - s\alpha)/c
\]

---

**Figure 4.2: Computing a $2 \times 2$ H-orthogonal transformation**

Cholesky factor of $A$ and $\tilde{A}$ is the Cholesky factor of $\tilde{A}$, then

\[
\frac{||\tilde{R} - R||}{||R||} < \frac{||R^{-1}||^2}{\sqrt{2}} ||H||
\]

(4.11)

The problem occurs when a matrix is to be downdated to a matrix of reduced rank. Perturbation analysis reveals that this computation is inherently unstable and no algorithm can stabilize this. However, certain downdating algorithms possess the property of *relational stability* (it was called *weak forward stability* by Alexander, Pan and Plemmons [1]). This property ensures that when a sequence of updates and downdates are performed, the numerical errors introduced at a downdating step will not affect the accuracy of the result of a later update, provided the final result is mathematically well conditioned. This implies large intermediate errors get "corrected" \(^1\) by the later iterations. It has been shown that

\(^1\)The term “corrected” is misleading in this context. The relationally stable algorithms possess the property that the result matrices are the solution to a nearby problem. However, since the problem is highly sensitive to perturbations, these errors get magnified to cause large apparent errors. At a later update when the problem becomes well-conditioned, these errors “disappear” since the algorithm has always been computing the result of a nearby problem, thereby causing the illusion that the algorithm corrects its own errors.
Computation of $\Psi_2^{1|2}$ such that \( \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \Psi_2^{1|2} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} 
\)
\[
c = \sqrt{(\alpha - \beta)(\alpha + \beta)/\alpha}
\]
\[
s = \beta/\alpha
\]
\[
\Psi_2^{1|2} = \frac{1}{c} \begin{pmatrix} 1 & -s \\ -s & 1 \end{pmatrix}
\]

Application of $\Psi_2^{1|2}$ to compute the vector $\begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \Psi_2^{1|2} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$:
\[
\gamma = (\alpha - s\beta)/c
\]
\[
\delta = c\beta - s\gamma
\]

Figure 4.3: Computing a $2 \times 2$ H-orthogonal transformation (Chamber's algorithm)

the hyperbolic rotation computed in the naive way as in Fig 4.2 is not relationally stable, while the same transformation computed using the Chamber’s algorithm [14](see Fig. 4.3), which just reorders some of the computations is relationally stable.

4.3.3 Previous stability analyses of downdating algorithms

Here, we will review the analyses of the Linpack downdating, hyperbolic downdating and the Chamber’s algorithms. All these previous analyses assume a floating point implementation. In this thesis, we will investigate if the same results hold in fixed-point arithmetic. The Cholesky downdating problem consists of computing $R$, where $S'S - xx' = R'R$. It is assumed that the matrix $S'S - xx'$ is positive definite and hence the Cholesky decomposition exists.

It can be shown that mathematically there exists an orthogonal $Q$ such that
\[
\begin{pmatrix} S \\ 0 \end{pmatrix} = Q \begin{pmatrix} R \\ x' \end{pmatrix}.
\]

(4.12)
However, if $R$ is the result of either the Linpack algorithm or the Chamber's algorithm, computed in floating-point arithmetic, then it has been shown that there exists an orthogonal matrix $Q$ (which is not computed) and an error matrix $E$ satisfying

$$||E|| < k ||S|| \epsilon_M,$$

such that

$$\begin{pmatrix} S \\ 0 \end{pmatrix} = Q \begin{pmatrix} R' \\ x' \end{pmatrix} + E.$$

(4.13)

(4.14)

This result is not backward stability, since it is not possible to concentrate the entire error in the matrix $S$ and the vector $x$. Stewart calls this property relational stability because the defining mathematical relation between the true quantities continues to be satisfied up to a small error by the computed quantities. The hyperbolic downdating algorithm is neither backward nor relationally stable and hence exhibits poor stability properties compared to the other two methods.

In fixed-point, the three algorithms will be neither backward nor relationally stable. However, they still perform about the same relative to each other. i.e. the test described by Stewart [61] still shows that the hyperbolic downdating algorithm has poor properties compared to the Chambers and the Linpack algorithms.

Table 4.1 shows this behavior. The test starts by generating random upper triangular matrices $R_0$, $R_1$ and $R_2$ and vectors $x$, $y$ such that $R_1$ is the result of downdating $R_0$ by the vector $x$ and $R_2$ is the result of updating matrix $R_1$ by the vector $y$. The matrices are actually computed by generating an ill-conditioned matrix $R_1$ and updating it with random vectors $x$ and $y$ to result in well-conditioned $R_0$ and $R_2$ respectively.

The second part of the test is to use the various downdating algorithms and downdate $R_0$ with the vector $x$ to result in $R_{h1}$, $R_{l1}$ and $R_{c1}$ using the hyperbolic, Linpack and Chamber's algorithms respectively. These matrices are then updated with the vector $y$ to result in $R_{h2}$, $R_{l2}$ and $R_{c2}$ respectively. The relative error in the computed matrix $R_{p2}$ would be small even if the relative error in the computed $R_{p1}$ is large, provided the
downdating algorithm is relationally stable. In this table, fixed-point paths of width 29 bits
were used. This corresponds to a machine precision, \( \epsilon_M \) of approximately \( 7 \times 10^{-9} \). The
table shows the 2-norm condition numbers \( R_1 \) and \( R_2 \) for several tries of the experiment and
the corresponding relative accuracies of the computed \( R_{p1} \) and \( R_{p2} \) for various downdating
algorithms (the updating algorithm is always the same and is unconditionally stable).

Similarly the matrices \( R_{hm1} \) and \( R_{hm2} \) are obtained using a mixed system where the
hyperbolic rotation is computed in floating point and truncated to fixed-point precision
following which the rest of the computation proceeds in fixed-point. A similar procedure
is followed for the Chambers algorithm. In the table some of the experiments in which the
downdating failed are indicated by a * to alert the reader that the algorithm is not always
guaranteed to succeed.

### 4.3.4 Relational stability

In this section we will present a summary of the important results which are used in showing
that the Chamber's algorithm is relationally stable. We first present a set of conditions
which describe the stability of each of the individual steps:

1. Suppose \( \Psi \) is a \( 2 \times 2 \) hyperbolic rotation computed to reduce the \((2,1)\) element of the
vector \( \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \) to zero and thus transform it into the vector \( \begin{pmatrix} \gamma \\ 0 \end{pmatrix} \), then there exists an
error vector \( \begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix} \) and an exact hyperbolic rotation \( \Phi \), such that

\[
\Phi \begin{pmatrix} \alpha + \Delta \alpha \\ \beta + \Delta \beta \end{pmatrix} = \begin{pmatrix} \gamma \\ 0 \end{pmatrix},
\]

and the error vector satisfies the condition

\[
\left\| \begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix} \right\|_2 < K_1 \epsilon \left\| \begin{pmatrix} \gamma \\ \beta \end{pmatrix} \right\|_2,
\]

where \( K_1 \) is a constant and \( \epsilon \) is the machine precision.
Table 4.1: Results of Stewart's test [61] performed in fixed-point/hybrid arithmetic

\[ \kappa(R) = \|R\|_2 \|R^{-1}\|_2 \quad \rho_{pi} = \frac{\|R_{pi} - R\|_2}{\|R\|_2} \]

<table>
<thead>
<tr>
<th>(\kappa(R_1))</th>
<th>(\kappa(R_2))</th>
<th>(\rho_{h1})</th>
<th>(\rho_{h2})</th>
<th>(\rho_{l1})</th>
<th>(\rho_{l2})</th>
<th>(\rho_{c1})</th>
<th>(\rho_{c2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.21e+02</td>
<td>1.45e+01</td>
<td>3.04e-03</td>
<td>2.91e-05</td>
<td>7.59e-02</td>
<td>2.56e-05</td>
<td>1.23e-03</td>
<td>8.28e-07</td>
</tr>
<tr>
<td>8.41e+02</td>
<td>4.03e+01</td>
<td>4.96e-03</td>
<td>5.89e-05</td>
<td>2.55e-01</td>
<td>2.56e-04</td>
<td>5.02e-03</td>
<td>2.34e-06</td>
</tr>
<tr>
<td>2.90e+03</td>
<td>2.73e+02</td>
<td>1.50e-02</td>
<td>1.26e-04</td>
<td>8.33e-02</td>
<td>2.94e-04</td>
<td>5.73e-02</td>
<td>2.41e-04</td>
</tr>
<tr>
<td>5.84e+02</td>
<td>1.21e+01</td>
<td>3.87e-03</td>
<td>9.97e-05</td>
<td>3.17e-01</td>
<td>3.08e-05</td>
<td>3.99e-03</td>
<td>1.78e-06</td>
</tr>
<tr>
<td>1.59e+03</td>
<td>8.82e+01</td>
<td>2.22e-03</td>
<td>2.89e-05</td>
<td>4.05e-02</td>
<td>4.26e-05</td>
<td>2.25e-03</td>
<td>5.52e-06</td>
</tr>
<tr>
<td>3.66e+04</td>
<td>2.09e+01</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>2.64e+03</td>
<td>2.06e+01</td>
<td>8.69e-03</td>
<td>1.69e-05</td>
<td>3.37e-02</td>
<td>1.90e-06</td>
<td>9.32e-03</td>
<td>1.94e-06</td>
</tr>
<tr>
<td>1.38e+03</td>
<td>3.95e+01</td>
<td>2.88e-03</td>
<td>9.91e-05</td>
<td>2.49e-02</td>
<td>3.52e-05</td>
<td>2.94e-03</td>
<td>5.65e-06</td>
</tr>
<tr>
<td>7.07e+03</td>
<td>1.43e+02</td>
<td>*</td>
<td>*</td>
<td>5.95e-02</td>
<td>1.50e-05</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>3.86e+04</td>
<td>8.59e+01</td>
<td>*</td>
<td>*</td>
<td>4.29e-03</td>
<td>1.72e-06</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>8.28e+03</td>
<td>1.76e+02</td>
<td>*</td>
<td>*</td>
<td>3.65e-02</td>
<td>5.17e-05</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\rho^m_{h1})</th>
<th>(\rho^m_{h2})</th>
<th>(\rho^m_{c1})</th>
<th>(\rho^m_{c2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.59e-04</td>
<td>2.75e-06</td>
<td>1.57e-04</td>
<td>8.57e-08</td>
</tr>
<tr>
<td>3.34e-04</td>
<td>1.98e-06</td>
<td>2.09e-04</td>
<td>4.25e-07</td>
</tr>
<tr>
<td>2.09e-03</td>
<td>1.69e-05</td>
<td>8.67e-04</td>
<td>7.17e-06</td>
</tr>
<tr>
<td>6.06e-05</td>
<td>1.63e-06</td>
<td>5.90e-05</td>
<td>5.19e-08</td>
</tr>
<tr>
<td>8.98e-05</td>
<td>1.21e-06</td>
<td>8.96e-05</td>
<td>1.38e-07</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1.98e-04</td>
<td>3.29e-06</td>
<td>1.05e-03</td>
<td>1.97e-07</td>
</tr>
<tr>
<td>4.42e-05</td>
<td>2.87e-07</td>
<td>5.34e-05</td>
<td>1.23e-07</td>
</tr>
<tr>
<td>1.72e-03</td>
<td>1.41e-06</td>
<td>1.18e-03</td>
<td>8.09e-07</td>
</tr>
<tr>
<td>7.58e-03</td>
<td>2.33e-06</td>
<td>3.63e-03</td>
<td>1.09e-07</td>
</tr>
<tr>
<td>1.72e-03</td>
<td>5.36e-07</td>
<td>5.12e-04</td>
<td>1.33e-07</td>
</tr>
</tbody>
</table>
2. Suppose \( \Psi \) is a \( 2 \times 2 \) hyperbolic rotation used to transform a vector \( \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \) into the vector \( \begin{pmatrix} \gamma \\ \delta \end{pmatrix} \) then there exists an error vector \( \begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix} \) and an exact hyperbolic rotation \( \Phi \), such that \[ \Phi \begin{pmatrix} \alpha + \Delta \alpha \\ \beta + \Delta \beta \end{pmatrix} = \begin{pmatrix} \gamma \\ \delta \end{pmatrix}, \] (4.17) and the error vector satisfies the condition \[ \left\| \begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix} \right\|_2 < K_1 \epsilon \left\| \begin{pmatrix} \gamma \\ \delta \end{pmatrix} \right\|_2 \] (4.18)

3. The computation of a hyperbolic transformation \( \mathbf{H} \) to transform the matrix \( \begin{pmatrix} S \\ x' \end{pmatrix} \) into the matrix \( \begin{pmatrix} R \\ 0 \end{pmatrix} \), where \( S \) and \( R \) are both upper triangular and \( x \) is a vector, and the matrix \( (S' S - xx') \) is positive definite, satisfies the condition that there exists an error matrix \( E \) and an exact orthogonal matrix \( Q \) such that \[ Q \begin{pmatrix} S \\ 0 \end{pmatrix} = \begin{pmatrix} R \\ z' \end{pmatrix} + E, \] (4.19) and the error matrix satisfies the condition \[ \left\| E \right\| < K_2 \epsilon \left\| \mathbf{S} \right\| \] (4.20)

4. Suppose \( R_n \) is an upper triangular matrix which is the computed result of a series of \( n \) updates and downdates to an upper triangular matrix \( R_0 \), the algorithm ensures that there exists an error matrix \( E \) and an exact orthogonal matrix \( Q \) such that \[ Q \begin{pmatrix} R_0 \\ 0 \\ X_u' \\ X_d' \end{pmatrix} = \begin{pmatrix} R_n \\ X_d' \\ X_u' \\ 0 \end{pmatrix} + E \] (4.21) and the error matrix satisfies the condition \[ \left\| E \right\| < nK_4 \rho \epsilon. \] (4.22)
Here $X_d$ is defined as the matrix whose columns comprise all the vectors used in the
downdates, while $X_u$ is a matrix defined from all the vectors used in the updates. In
addition, $\rho$ is defined as

$$\rho = \max\{||R_i|| : i = 1, \ldots, n\}$$  \hspace{1cm} (4.23)

It is assumed that the matrix $R'_0 R_0 + X_u X'_u - X_d X'_d$ is positive definite.

5. If $S_n$ is a Cholesky factor of the matrix $R'_0 R_0 + X_u X'_u - X_d X'_d$ and $R_n$ is the computed
Cholesky factor, then

$$\frac{||S_n - R_n||}{||R_n||} < K_5 \hat{\rho}^2 \epsilon \left\| R_n^{-1} \right\|^2.$$  \hspace{1cm} (4.24)

The quantity $\hat{\rho}$ is defined as:

$$\hat{\rho} = \max\left\{ \rho, \left\| \left( \begin{array}{c} R_0 \\ X'_u \end{array} \right) \right\| \right\}$$  \hspace{1cm} (4.25)

Conditions 1 and 2 are related to the computation of a single $2 \times 2$ hyperbolic decom-
position. In particular, condition 1 deals with the computation of the hyperbolic rotation
to align a vector with one of the axes, while condition 2 deals with the transformation
of a vector which is unrelated to the vector which was used to compute the hyperbolic
transformation.

Condition 3 deals with the algorithm used for a rank-one downdate of a Cholesky de-
composition. This is neither backward nor forward stability, but a mixed-stability criterion.

Condition 4 deals with a series of updates and downdates. Any algorithm which satisfies
this criterion essentially computes a Cholesky factor whose accuracy is determined only by
the condition of the final Cholesky factor and the norms of all the intermediate factors.

Condition 5 is the condition which illustrates the relational stability behavior of an
algorithm used to perform a series of updates and downdates. The accuracy of the computed
factor depends only on the condition of the final matrix and the norms of the intermediate
factors.

The proof that Chamber's algorithm using floating-point arithmetic is relationally sta-
ibility is derived from the following lemmas.
Lemma 4.1 If a $2 \times 2$ transformation is computed and applied using the Chamber's algorithm then it satisfies Conditions 1 and 2.

Proof The proof may be found in the paper by Bojanczyk, Brent, van Dooren and De Hoog [9].

Lemma 4.2 If a rank-one Cholesky downdate is performed using a series of $2 \times 2$ hyperbolic rotations which satisfy conditions 1 and 2, then the resulting downdate algorithm satisfies condition 3.

Proof The proof may be found in the paper by Bojanczyk, Brent, van Dooren and De Hoog [9].

Lemma 4.3 If a series of rank-one Cholesky updates and downdates are performed using an algorithm which satisfies condition 3 for each individual update/downdate, then the resulting algorithm satisfies condition 4.

Proof The proof may be found in a paper by Stewart [61].

Lemma 4.4 If the algorithm for a series of downdates and updates satisfies condition 4 then the resulting computed Cholesky factor satisfies the relational stability condition 5.

Proof The proof may be found in a paper by Stewart [61].

4.3.5 Effect of fixed-point computation on relational stability

We now seek to relax the conditions defined above such that the conditions can be met using fixed-point computation. The key difference in the following definitions is that the resulting error vectors/matrices satisfy an absolute accuracy criterion rather than a relative accuracy
condition. Since the same computation using fixed-point and floating-point usually satisfy this kind of a relationship, our hope is that this would extend to the Cholesky downdating problem. In effect, we seek to define a weaker version of the relational stability criterion defined by Stewart. This weak definition should suffice in practice.

1. Suppose $\Psi$ is a $2 \times 2$ hyperbolic rotation computed to reduce the $(2,1)$ element of the vector $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ to zero and thus transform it into the vector $\begin{pmatrix} \gamma \\ 0 \end{pmatrix}$, then there exists an error vector $\begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix}$ and an exact hyperbolic rotation $\Phi$, such that

$$
\Phi \begin{pmatrix} \alpha + \Delta \alpha \\ \beta + \Delta \beta \end{pmatrix} = \begin{pmatrix} \gamma \\ 0 \end{pmatrix},
$$

and the error vector satisfies the condition

$$
\left\| \begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix} \right\|_2 < K_1 \epsilon
$$

(4.26)

2. Suppose $\Psi$ is a $2 \times 2$ hyperbolic rotation used to transform a vector $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ into the vector $\begin{pmatrix} \gamma \\ \delta \end{pmatrix}$ then there exists an error vector $\begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix}$ and an exact hyperbolic rotation $\Phi$, such that

$$
\Phi \begin{pmatrix} \alpha + \Delta \alpha \\ \beta + \Delta \beta \end{pmatrix} = \begin{pmatrix} \gamma \\ \delta \end{pmatrix},
$$

(4.28)

and the error vector satisfies the condition

$$
\left\| \begin{pmatrix} \Delta \alpha \\ \Delta \beta \end{pmatrix} \right\|_2 < K_1 \epsilon
$$

(4.29)

3. The computation of a hyperbolic transformation $H$ to transform the matrix $\begin{pmatrix} S \\ z' \end{pmatrix}$ into the matrix $\begin{pmatrix} R \\ 0 \end{pmatrix}$, where $S$ and $R$ are both upper triangular and $z$ is a vector,
and the matrix \((S' S - xx')\) is positive definite, satisfies the condition that there exists an error matrix \(E\) and an exact orthogonal matrix \(Q\) such that
\[
Q \begin{pmatrix} S \\ 0 \end{pmatrix} = \begin{pmatrix} R \\ x' \end{pmatrix} + E,
\]
and the error matrix satisfies the condition
\[
\|E\| < K_2 \epsilon
\]

4. Suppose \(R_n\) is an upper triangular matrix which is the computed result of a series of \(n\) updates and downdates to an upper triangular matrix \(R_0\), the algorithm ensures that there exists an error matrix \(E\) and an exact orthogonal matrix \(Q\) such that
\[
Q \begin{pmatrix} R_0 \\ 0 \\ X_d' \end{pmatrix} = \begin{pmatrix} R_n \\ X_u' \\ 0 \end{pmatrix} + E
\]
and the error matrix satisfies the condition
\[
\|E\| < nK_4 \epsilon.
\]
Here \(X_d\) is defined as the matrix whose columns comprise all the vectors used in the downdates, while \(X_u\) is a matrix defined from all the vectors used in the updates. It is assumed that the matrix \(R_0 X_u X_u' - X_d X_d'\) is positive definite.

5. If \(S_n\) is a Cholesky factor of the matrix \(R_0 R_0 + X_u X_u' - X_d X_d'\) and \(R_n\) is the computed Cholesky factor, then
\[
\frac{\|S_n - R_n\|}{\|R_n\|} < K_5 \beta \epsilon \|R_n^{-1}\|^2.
\]
The quantity \(\beta\) is defined as:
\[
\beta = \left\| \begin{pmatrix} R_0 \\ X_u' \end{pmatrix} \right\|.
\]

Given this new set of conditions, we show that lemma 4.1, 4.2, 4.3 and 4.4 hold with the new conditions replacing the corresponding relative accuracy conditions.
4.3.6 Proofs for fixed-point relational stability

In the following statements and proofs the new set of conditions described in Section 4.3.5 are referred.

**Lemma 4.5** If the algorithm for a series of downdates and updates satisfies condition 4 then the resulting computed Cholesky factor satisfies the "weak" relational stability condition 5.

**Proof** We are given that \( S_n \) is the Cholesky factor of \( R_0^2 R_0 + X_u X_u' - X_d X_d' \equiv \tilde{A} \). Since condition 4 implies

\[
Q \left\{ \begin{pmatrix} R_0 \\ 0 \\ X_u' \end{pmatrix} + \hat{E} \right\} = \begin{pmatrix} R_n \\ X_d' \\ 0 \end{pmatrix},
\]

(4.36)

we get the relation,

\[
R_0' R_0 + X_u X_u' + \begin{pmatrix} R_0 \\ 0 \\ X_u' \end{pmatrix} + \begin{pmatrix} R_0 \\ 0 \\ X_u' \end{pmatrix}' = R_n' R_n + X_d X_d'
\]

\[
(R_0' R_0 + X_u X_u' - X_d X_d') + H = R_n' R_n
\]

From the perturbation result for Cholesky decomposition as described in theorem 4.1, we get

\[
\frac{\| S_n - R_n \|}{\| R_n \|} < \frac{\| R_n^{-1} \|}{\sqrt{2}} \| H \|.
\]

(4.37)

Given that the error matrix \( E \), and hence \( \hat{E} \), is bounded by \( nK_4 \epsilon \), we get

\[
\| H \| < 2nK_4 \epsilon \tilde{p}.
\]

(4.38)

Thus the relative error in the computed Cholesky factor is:

\[
\frac{\| S_n - R_n \|}{\| R_n \|} < \sqrt{2nK_4 \epsilon \tilde{p}} \| R_n^{-1} \|^2.
\]

(4.39)
The quantity $\hat{\rho}$ is the norm of the matrix obtained if all the updates but none of the downdates are performed. It is important to note that it grows with the number of updates. Hence, it is possible for the error to grow unboundedly with the number of updates.

If outliers occur which increase $\hat{\rho}$ larger than the usual, the bound tells us to expect a persisting inaccuracy. This is a phenomenon observed which has been experimentally observed [8].

Other than $\hat{\rho}$ the bound depends only on the norm of the final Cholesky factor. Hence, as long as the final result is well-conditioned, we can expect an accurate result irrespective of the accuracy of any of the intermediate factors. This is the important relational stability criterion.

**Lemma 4.6** If a series of rank-one Cholesky updates and downdates are performed using an algorithm which satisfies condition 3 for each individual update/downdate, then the resulting algorithm satisfies condition 4.

**Proof** We will show this result holds for two successive downdates/updates. The result for a sequence of $n$ updates and downdates will follow by induction. Suppose we downdate $R_0$ with a vector $x$ to compute the Cholesky factor $R_1$, which is then updated with a vector $y$ to result in the computed factor $R_2$. For each of these individual steps it is assumed that the following relations hold:

\[
Q_0 \begin{pmatrix} R_0 \\ 0 \\ y' \end{pmatrix} = \begin{pmatrix} R_1 \\ z' \end{pmatrix} + E_1
\]

\[
Q_1 \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} z' \\ 0' \end{pmatrix} + E_2,
\]

\[
Q_2 \begin{pmatrix} R_2 \\ z' \end{pmatrix} = \begin{pmatrix} R_3 \\ 0' \end{pmatrix} + E_3
\]

\[
Q_3 \begin{pmatrix} z' \\ 0' \end{pmatrix} = \begin{pmatrix} z'' \end{pmatrix} + E_4
\]
where $Q_1$ and $Q_2$ are exact orthogonal matrices and $E_1$ and $E_2$ are error matrices whose norms are bounded by $K_2\epsilon$. If we set $E = Q_1' E_1 + E_2$ and $Q' = Q_1 Q_0$, then

\[
Q' \begin{pmatrix} R_0 \\ 0 \\ y' \end{pmatrix} = \begin{pmatrix} R_2 \\ x' \\ 0 \end{pmatrix} + E. \tag{4.40}
\]

It readily follows that

\[
\|E\| < \|E_1\| + \|E_2\| = 2K_2\epsilon \tag{4.41}
\]

In general, for $n$ updates/downdates, the resulting error matrix will grow as $nK_2\epsilon$. \hfill \Box

**Lemma 4.7** If a rank-one Cholesky downdate is performed using a series of $2 \times 2$ hyperbolic rotations which satisfy conditions 1 and 2, then the resulting downdate algorithm satisfies condition 3.

**Proof** Suppose the matrix \( \begin{pmatrix} R^{(0)} \\ z' \end{pmatrix} \) is transformed into \( \begin{pmatrix} R^{(N)} \\ 0 \end{pmatrix} \) through a series of hyperbolic rotations \( H_i H_2 \cdots H_N \), where \( N \) is the dimension of the matrix \( R^{(0)} \), and the computed intermediate matrix after \( i \) steps have been performed is \( \begin{pmatrix} R^{(i)} \\ z'_i \end{pmatrix} \). Here \( H_i \) is defined as the hyperbolic transformation which reduces the \( i \)th element of the vector \( z_{i-1} \) to zero.

After the first step the following condition will hold if we assume that the \( 2 \times 2 \) transformation satisfies condition 1 when computed from the pivot elements and condition 2 when later applied to the remaining elements of the pivot rows:

\[
H_1 \begin{pmatrix} R^{(0)} + \Delta R_0 \\ z' + \Delta z' \end{pmatrix} = \begin{pmatrix} R^{(1)} \\ z'_1 \end{pmatrix}, \tag{4.42}
\]

and the norm of the error matrix, satisfies the bound:

\[
\left\| \begin{pmatrix} \Delta R_0 \\ \Delta z' \end{pmatrix} \right\| < N K_1 \epsilon. \tag{4.43}
\]

The structure of \( \Delta R_0 \) is such that it has non-zero elements only in the first row. Since \( H_1 \) is an exact hyperbolic rotation this equation can be re-written in terms of an exact Jacobi
rotation, $Q_1$, as follows:

$$Q_1 \begin{pmatrix} R^{(1)} \\ x' + \Delta x' \end{pmatrix} = \begin{pmatrix} R^{(0)} + \Delta R_0 \\ x'_1 \end{pmatrix}$$

$$Q_1 \begin{pmatrix} R^{(1)} + \Delta R_1 \\ z' + \Delta z'_1 \end{pmatrix} = \begin{pmatrix} R^{(0)} \\ x'_1 \end{pmatrix},$$

where the new error matrix satisfies the error bound:

$$\left\| \begin{pmatrix} \Delta R_1 \\ \Delta z'_1 \end{pmatrix} \right\| < 2N K_1 \epsilon. \quad (4.44)$$

A similar bound is obtained at the second step. Because of the structure of the Jacobi rotations $Q_1$ and $Q_2$, it is possible to map the errors from the second step back to the first step to obtain:

$$Q_2 Q_1 \begin{pmatrix} R^{(2)} + \Delta R_0 + \Delta R_1 \\ x' + \Delta x'_1 + \Delta x'_2 \end{pmatrix} = \begin{pmatrix} R^{(2)} \\ x'_2 \end{pmatrix}. \quad (4.45)$$

Extending this analysis to $N$ iterations one finds that the computed quantities do indeed satisfy condition 3. 

\[ \square \]

**Lemma 4.8** Suppose a $2 \times 2$ hyperbolic transformation is defined as $\frac{1}{c} \begin{pmatrix} 1 & -s \\ -s & 1 \end{pmatrix}$ where $c^2 + s^2 = 1$. Suppose we have computed an approximate hyperbolic rotation defined in terms of finite precision quantities which approximate the exact parameters $c$ and $s$ with errors $\Delta c$ and $\Delta s$ bounded by the machine precision $\epsilon$. The application of this rotation to an arbitrary vector $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ using Chamber’s algorithm in fixed-point will satisfy condition 2.

**Proof** Suppose $\begin{pmatrix} \gamma \\ \delta \end{pmatrix}$ is the vector computed using Chamber’s algorithm described in Figure 4.3, where all the computations are performed in fixed-point. Then the following relations will hold:

$$\gamma = \frac{\alpha - ((s + \Delta s)\beta + \epsilon_1)}{c + \Delta c} + \epsilon_2$$

$$\delta = ((c + \Delta c)\beta + \epsilon_3) - ((s + \Delta s)\gamma + \epsilon_4).$$
Here we use the well-known relation that the computed result of a fixed-point multiplication or division will approximate the exact result to within machine precision. Re-writing the above relations we get:

\[
\begin{align*}
\gamma + s\beta &= \alpha + (\epsilon_2 - \epsilon_1 - \beta \Delta s - \gamma \Delta c + \epsilon_2 \Delta c) \\
-s\gamma + c\beta &= \delta + (\epsilon_4 - \epsilon_3 + \gamma \Delta s - \beta \Delta c).
\end{align*}
\]

The Givens rotation allows us to map the entire error onto \(\alpha\) and \(\beta\) while still keeping the resultant errors bounded. Hence we can rewrite the above relation in terms of an exact Givens rotation and an error vector such that:

\[
\begin{pmatrix}
\null & c \\
-s & \null
\end{pmatrix}
\begin{pmatrix}
\gamma \\
\beta + \Delta \beta
\end{pmatrix} = 
\begin{pmatrix}
\alpha + \Delta \alpha \\
\delta
\end{pmatrix},
\]

which can be re-written as:

\[
\frac{1}{c} \begin{pmatrix}
1 & -s \\
-s & 1
\end{pmatrix}
\begin{pmatrix}
\alpha + \Delta \alpha \\
\beta + \Delta \beta
\end{pmatrix} = 
\begin{pmatrix}
\gamma \\
\delta
\end{pmatrix},
\]

where the error is bounded as follows:

\[
\left\| \begin{pmatrix}\Delta \alpha \\ \Delta \beta\end{pmatrix} \right\|_2 < 4(1 + \sqrt{2})\epsilon.
\]

Now the only step which remains is showing that the computation of a single \(2 \times 2\) hyperbolic rotation is accurate. If we can compute the parameters \(c\) and \(s\) of the rotation to machine precision, then we have the result we seek. However, without any form of normalization the forward error in square root computations in fixed-point can be large when the operand is small [71].

A simple way to compute the hyperbolic rotation to machine precision is to compute it in floating point and then truncating the result to fixed-point precision.

**Lemma 4.9** If a \(2 \times 2\) hyperbolic rotation parameterized by \(c\) and \(s\) is computed as in Figure 4.3 using floating-point arithmetic, and the computed parameters are truncated to fixed-point precision, the parameters will approximate the true parameters to within machine precision.
Proof From the paper by Bojanczyk, Brent, van Dooren and De Hoog [9] we have a result that the error in the computed parameters \( c \) and \( s \) will be bounded by \( cc \). It follows that the truncated parameters will satisfy an absolute error bound.

\[ \Box \]

4.4 Summary

Downdating algorithms based on hyperbolic rotations are generally considered unstable. One of the goals of this chapter was to demonstrate that with some care it is possible to arrive at downdating algorithms which are usable in practice.

The sliding window algorithm described in this chapter is a combination of the hyperbolic QR downdating idea and the exponential window SVD updating array described by Moonen, Van Dooren and Vandewalle [49]. The exponential window SVD updating algorithm developed by Moonen uses the notion that the inherent tolerance of signal processing algorithms to occasional inaccuracies allows one to trade off accuracy for computation. This idea allows an order of magnitude reduction in computational complexity. We showed in this chapter how the idea extends to sliding window SVD algorithm designed using hyperbolic rotations. The reduction in computation over a non-updating implementation of sliding window subspace tracking is achieved by performing a Jacobi sweep over several updates. The convergence properties of the Jacobi method ensure that the computed subspaces will not diverge from the true subspace provided the signal to noise ratio is sufficiently high. The basic algorithm can be considered a sequence of three steps at each sample time: a QR updating, QR downdating and a Jacobi update step. These can all be seamlessly interleaved on a square array of processors for a constant, datasure-size-independent throughput.

The key step here which determines the numerical properties of this algorithm is the hyperbolic downdating step. While all the other portions can be implemented safely in fixed-point, it is not clear if the hyperbolic downdating can be coded in fixed-point. The analysis in this chapter shows that provided the Chamber's method is used for this key computation, all the operations including application of the hyperbolic rotations can be
coded in fixed-point, except for the generation of the hyperbolic rotation. Hence one way to implement this algorithm is to compute the parameters which define the hyperbolic rotation in floating-point, truncate them to fixed-point and then proceed with the rest of the computation in fixed-point. Such a hybrid algorithm enjoys a weak form of relational stability allowing the use of this algorithm in signal processing applications as will be shown in the next chapter.
Chapter 5

Architectures for CDMA Acquisition - Frontend

5.1 Introduction

The focus of this thesis now shifts from algorithms applicable in general to subspace-based techniques, to the specific problem of near-far resistant acquisition of CDMA communication signals. The rest of the thesis describes parallel algorithms and architectures for acquisition and tracking in near-far resistant CDMA systems. The emphasis is on reducing the computational complexity of these algorithms to facilitate digital implementation. The eigenanalysis based algorithm developed by Bensley [2] for the CDMA synchronization problem when coupled with other parallel algorithms results in a family of near-far resistant algorithms for the acquisition problem. We extend these algorithms further to be numerically robust and to allow parallel implementation necessary to satisfy latency and throughput constraints.

The algorithms of interest in this work are systolic in nature and hence are amenable to parallel implementation. In addition, these algorithms are built using techniques developed in numerical linear algebra and as a result are numerically robust. The resulting algorithms may be implemented in either custom VLSI or general-purpose DSP processors or a mixture of both.

5.2 Complexity of near-far resistant acquisition

Let us first examine the computational complexity of the eigenanalysis-based approach to CDMA synchronization, using an example. Suppose we consider a system with the parameters specified in Table 1.1, which represent the requirements of a modest communication
system for transmitting toll quality voice. The parameters of bandwidth and bit rate dictate a spreading gain no greater than 128. Let us choose a spreading gain of 31.

To achieve a probability of acquisition of 0.8 (where "acquisition" is defined as estimating the time origin of the transmitted symbol to within a single chip period of the actual delay) and a standard deviation of 0.005 chip period, simulations under ideal conditions indicate that one feasible solution is to estimate the sample correlation matrix from \( L \geq 200 \) observation vectors. The amount of computation required to implement the single-path MUSIC algorithm to estimate the delay is approximately 40 million arithmetic operations per second (see Table 5.1). This computational complexity grows as a square of the spreading gain and linearly with the desired bit rate. It is evident that a more demanding communication system would require much higher computational power of the DSP system. For instance, in order to achieve Ethernet bit rates of 10Mbits/sec, and say about 50 users, we would need on the order of 1 Tera ops/sec.

If we regard the current state of art as 170 MIPS and extrapolate using Moore’s law, which states processors double in speed every 18 months, it appears that the Ethernet speeds would have to wait for at least 20 years to be implemented on a single general purpose processor.

<table>
<thead>
<tr>
<th>Table 5.1: Computational complexity of near-far resistant acquisition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Rate</td>
</tr>
<tr>
<td>Number of users</td>
</tr>
<tr>
<td>Spreading code length</td>
</tr>
<tr>
<td>Number of observations/observation matrix</td>
</tr>
<tr>
<td>Operations to compute SVD</td>
</tr>
<tr>
<td>Operations to compute MUSIC backend (per user)</td>
</tr>
<tr>
<td>Maximum number of users</td>
</tr>
<tr>
<td>Number of users for maximum computation</td>
</tr>
<tr>
<td>Maximum computation</td>
</tr>
</tbody>
</table>
5.3 Strategies to improve eigen-analysis algorithms

Given the computational complexity of these algorithms, there are but few alternatives to tackle the implementation problem. The first alternative is to examine the problem carefully and eliminate all extraneous computation. The second avenue open to us is the use of another "law" of DSP processors, that the price per MIP falls at approximately 25% every year, allowing parallel implementations to be a viable alternative.

Subspace-based algorithms stated in the form described in Sec. 2.3.4 are not very suited to pipelined parallel implementations. However, parallel architectures have previously been developed for similar algorithms in direction-finding problems. We will extend these results to the CDMA synchronization problem.

In order to make the problem more manageable from the viewpoint of parallelization, it is convenient to separate the algorithms into smaller parts. An intuitive and convenient way to handle these algorithms is to split them as a frontend subspace decomposition and a backend non-linear optimization problem (Fig. 5.1). Other factors which guide such a treatment is that the subspace tracking problem is a well-defined linear algebra problem and has been studied extensively in literature. The problem is characterized by regular computation ideally suited for systolic arrays. The frontend computation is almost completely independent of the algorithms used in the backend and often independent of the overall problem.

The backend computation on the other hand is marked by irregular computation, where the execution times may not be predictable. Often a closed form solution is not feasible due to the multimodal nature of the objective functions, and an extensive search may be needed to solve the problem. The backend problem formulation is much more dependent on the overall problem. The above separation also allows development of several alternative designs for the frontend and backend, which can then be mixed and matched at will with very few restrictions.
Figure 5.1: Synchronization Block

The rest of the current chapter is devoted to the frontend problem, relegating the study of the backend algorithms to Chap. 6.

5.4 Formulation of the Frontend Problem

The requirement of the frontend system can be succinctly stated as the computation of an eigenvalue decomposition. However, as will be evident as we develop this chapter, this rather simplistic description of the functionality belies the large number of possible solutions.
If we begin with the algorithm described by Bensley [2], we define the frontend system as performing the following functions:

- Form a correlation matrix from several observation vectors.
- Compute eigenvalue decomposition of this sample correlation matrix
- Sort the eigenvalues and the corresponding eigenvectors into two sets (or subspaces) with the signal subspace consisting of the largest singular values and the noise subspace constituted of the rest.

However, in practice it is neither necessary nor desirable to perform all the computations in the above specified order. Let us elaborate on this point. Certain computational reductions can be achieved if the backend does not require the eigenvalues. For instance, if the deterministic MUSIC algorithm is the target architecture, then it is not necessary to compute the eigenvalues. In fact, it is not even necessary to compute the signal subspace. In all the backend architectures considered in this thesis, it is never necessary to completely sort the eigenvectors. It is sufficient to separate the signal and noise eigenvectors. Additional computational reductions can be achieved if only a few of the signal or noise eigenvectors are required.

The formulation of the subspace-based methods as described here requires the computation of the sample correlation matrix followed by an eigenvalue decomposition. Such a computation cannot be readily pipelined because in general it is necessary to compute the correlation matrix completely before the eigenvalue decomposition can be computed. This is a fundamental bottleneck of the above sequence of computations. Fortunately, there exists an equivalent formulation which eliminates this dependency and at the same time results in superior numerical accuracy. This alternative formulation transforms the eigenvalue decomposition of the correlation matrix into the singular value decomposition of an observation matrix. We will delve into this in much greater depth later in this chapter.
The algorithms developed for the frontend also depend on the way the subspace-based algorithms derived for stationary conditions, are extended to non-stationary conditions. In this context, we will separate the frontend algorithms into the classes of recursive and non-recursive (or block) formulations. The recursive formulations we will consider in this report are exponential downdating and the sliding rectangular window. It is most convenient to describe these formulations by defining an observation matrix.

5.4.1 Formation of an observation matrix

Recursive and non-recursive formulations essentially determine how an observation matrix is formed from the observation vectors defined in section 2.3.4.

The non-recursive (or the block) formulation defines a fixed size window of length $L$ over the input vectors. The sample correlation matrix for the $p^{th}$ block is then defined as:

$$
\hat{R}_p = \frac{1}{L} \sum_{i=p}^{(p+1)L-1} y_i y'_i = \frac{1}{L} Y_p' Y_p,
$$

(5.1)

where, $p = 0, 1, 2, \cdots$, and $Y_p$ is defined as

$$
Y_p = \begin{pmatrix}
Y'_pL \\
Y'_{pL+1} \\
\cdots \\
Y'_{(p+1)L-1}
\end{pmatrix}.
$$

(5.2)

The exponential downdating formulation defines the sample correlation matrix recursively as

$$
\hat{R}_p = \beta^2 \hat{R}_{p-1} + y_p y'_p = Y'_p Y_p,
$$

(5.3)

\footnote{The scaling by $1/L$ is not strictly necessary since it disappears later in the computation.}
where, \( p = 0, 1, \ldots \) and \( 0 < \beta < 1 \) is the forgetting factor. The observation matrix \( Y_p \) is itself recursively defined as

\[
Y_p = \begin{pmatrix} \beta^{p-1}y_0' \\ \vdots \\ \beta y_{p-1}' \\ y_p' \end{pmatrix} = \begin{pmatrix} \beta Y_{p-1} \\ Y_p' \end{pmatrix}.
\]

(5.4)

Another recursive formulation is the sliding window formulation which defines a rectangular window of length \( L \) over the observation vectors:

\[
\hat{R}_p = \hat{R}_{p-1} + y_p y_p' - y_{p-L+1} y_{p-L+1}' = Y_p' Y_p,
\]

(5.5)

where, \( p = 0, 1, \ldots \) and the observation matrix \( Y_p \) is defined as

\[
Y_p = \begin{pmatrix} y_{p-L+1}' \\ \vdots \\ y_{p-1}' \\ y_p' \end{pmatrix}.
\]

(5.6)

Hence the observation matrix may be recursively defined as

\[
\begin{pmatrix} y_{p-L}' \\ Y_p \end{pmatrix} = \begin{pmatrix} Y_{p-1} \\ Y_p' \end{pmatrix}.
\]

(5.7)

Although the two recursive formulations look very different, it turns out that there exist algorithms for the rectangular sliding window which have a structure very similar to those for exponential downdating.

More insight into the above formulations may be obtained by employing a filtering viewpoint. If we define a filter with impulse response \( h[n] \) operating on the input observation vectors, then the output of the filter is a vector obtained as \( y[n] * h[n] \), where each component of the vector is the corresponding input element convolved with the impulse response of the filter. The sample correlation matrix can be defined as \( \sum_{n=0}^{\infty} (y[n] * h[n])(y[n] * h[n])' \).
Using this notation, it is easily verified that the sliding window approach corresponds to a FIR filter with response

\[ h[n] = \begin{cases} 1 & \forall 0 \leq n < L \\ 0 & \text{otherwise.} \end{cases} \quad (5.8) \]

The block algorithm can be verified to be a decimating version of the same filter, where the decimation factor corresponds to the length of the filter. Many of the properties of FIR filters should also apply to the resulting formulations. In particular, the linear phase of the rectangular FIR filter implies that all the frequencies in the time delay should appear in the output estimate with the same delay. It is well-known that the group delay for the above filter is half the length of the filter. Hence the time delay estimate computed at any time instant should be applied to the input data obtained at the time instant corresponding to the mid-point of the observation interval. Another property which we would expect to see is the computational efficiency of decimation filters. Since the block methods correspond to a decimation filter, we expect computational efficiencies similar to those for polyphase implementation of decimation filters.

The exponential windowing approach (or alternately the lossy integrator) is a familiar IIR filter with an impulse response:

\[ h[n] = \beta^n \quad \forall n \geq 0. \quad (5.9) \]

Once again the familiar properties of the IIR filter should re-surface in the estimated time delays. Different frequencies in the time delay will appear with different delays as evident by the non-linear phase response of the filter. This makes it more difficult to determine which data vectors correspond to the estimate of the time delay at any time instant. One approach to alleviate this problem would be to restrict the variation in the time delay to within a small frequency band within which the phase response may be approximated as linear. The corresponding group delay is \( \beta/(1 - \beta) \). Such a restriction would prevent degradation of the performance due to the incorrect application of the correctly estimated
time delays. Although it is true that in most signal processing applications the FIR filters would require memory while the IIR filters do not, the same property does not extend to our application. While it is true that no memory of previous observations is needed for the computation of the estimates of the time delay, memory equivalent to the group delay is still needed at the output where the estimates are used to process an older observation.

It is evident that through a combination of different filters and decimation factors a wide variety of formulations are possible. However, from an implementation viewpoint all filters are not created equal. More complex filters merely increase the complexity of the computation because they may not allow a simple recursive computation for the SVD. Hence, from an implementation viewpoint it still makes sense to view them as separate formulations which require widely different implementations.

5.5 Improved algorithms for the frontend

5.5.1 A practical way to compute EVD of the correlation matrix

The original formulation of the subspace-based methods in the previous sections requires the computation of the sample correlation matrix and then performing an eigenvalue decomposition. However, the use of a well known trick from numerical algebra, allows the computation of the eigenvalues and eigenvectors directly from the observation matrix. This alternate formulation exploits the structure of the correlation matrix, \( \hat{\mathbf{R}}_p \), which is defined in terms of an observation matrix, \( \mathbf{Y}_p \) as:

\[
\hat{\mathbf{R}}_p = \mathbf{Y}_p^\prime \mathbf{Y}_p.
\] (5.10)

It is well-known that the singular value decomposition of \( \mathbf{Y}_p \) provides all the information necessary to compute the eigenvalue decomposition of the sample correlation matrix.

The SVD of \( \mathbf{Y}_p \) is defined as

\[
\mathbf{Y}_p = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^\prime.
\] (5.11)
where, \( U \) and \( V \) are unitary matrices and \( \Sigma \) is a diagonal matrix of singular values. From this it follows that

\[
R_p = Y_p'Y_p = V\Sigma U'U\Sigma V' = V\Sigma^2 V',
\]

which is the eigenvalue decomposition of the sample correlation matrix. Hence, the eigenvalues and eigenvectors can be obtained directly from the SVD of the observation matrix instead of computing them from an intermediary correlation matrix. For a given wordlength, this alternate formulation would result in superior numerical performance since the computation of the correlation matrix results in a loss of precision.

5.5.2 Numerically robust algorithms for the frontend

There are a variety of algorithms that have been developed in recent years to perform what is known as subspace tracking. Table 5.2 shows the large number of options available and the quantities they can be used to compute. We classify the algorithms broadly as recursive and non-recursive algorithms. The recursive algorithms may be used when an older decomposition is updated whenever new data is received. The non-recursive algorithms on the other hand recompute the desired quantities from scratch when new data is received. The non-recursive algorithms in general result in less computation per observation than the recursive techniques. Hence they are attractive when parameters change very slowly and a lower cost solution is desired.

Each of these classes is further subdivided into algorithms which compute the complete decomposition, algorithms which compute only the required orthonormal bases for the signal and noise subspaces, or even just a few of the largest singular vectors.

Many of these algorithms also have different degrees of parallelism. Hence, some of the algorithms are amenable to implementation on linear arrays of dimension proportional to the problem size, to result in similar throughput increases. Other algorithms exhibit even higher degrees of parallelism and allow implementation on arrays whose size is limited by the square of the dimension of the problem, with corresponding throughput increases.
### Table 5.2: Classification of frontend algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computation/observation</th>
<th>Computed quantities</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-recursive subspace tracking</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QR algorithm [29, 30, 32]</td>
<td>$O(n^2)$</td>
<td>$\Sigma, V$</td>
<td>-</td>
</tr>
<tr>
<td>Jacobi method [25, 11]</td>
<td>$O(n^2 \log n)$</td>
<td>$\Sigma, V$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Hestenes method [37, 10]</td>
<td>$O(n^2 \log n)$</td>
<td>$\Sigma, V$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>One-sided Jacobi method [22]</td>
<td>$O(n^2 \log n)$</td>
<td>$\Sigma, V$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Pipelined SVD [42]</td>
<td>$O(n^2 \log n)$</td>
<td>$\Sigma, V$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>RRQR [15]</td>
<td>$O(n^2)$</td>
<td>$V$</td>
<td>-</td>
</tr>
<tr>
<td>URVD [58], ULVD [58]</td>
<td>$O(n^2)$</td>
<td>$V$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Recursive subspace tracking</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD updating [12]</td>
<td>$O(n^3 \log n)$</td>
<td>$\Sigma, V$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Approximate SVD updating [49, 68, 50]</td>
<td>$O(n^2)$</td>
<td>$\Sigma, V$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>URV, ULV updating</td>
<td>$O(n^2)$</td>
<td>$V$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>RLS Subspace tracking [74]</td>
<td>$O(nr)$</td>
<td>$V_s$</td>
<td>-</td>
</tr>
<tr>
<td>Lanczos subspace tracking [16, 72, 73]</td>
<td>$O(nr)$</td>
<td>$V_s$</td>
<td>-</td>
</tr>
<tr>
<td>Spherical subspace tracking [17]</td>
<td>$O(nr)$</td>
<td>$V_s$</td>
<td>$O(r)$</td>
</tr>
<tr>
<td>Schur algorithm [65, 33]</td>
<td>$O(n^2)$</td>
<td>$V$</td>
<td>-</td>
</tr>
</tbody>
</table>
Algorithms to compute singular vectors and singular values

When the backend non-linear optimization requires both the singular values and singular vectors, it is necessary to restrict one’s attention to those frontend algorithms which perform a complete singular value decomposition. These algorithms are indicated in Table 5.2 as those which compute the singular value matrix $\Sigma$ and the corresponding singular vector matrix $V$.

The QR algorithm [31] coupled with the Golub-Kahan step is the fastest algorithm known to compute the SVD. However, the algorithm is inherently serial in nature and does not scale very well on parallel arrays. We do not imply that it is impossible to parallelize the algorithm. It is possible to get some speedup by parallelizing the algorithm. However, the performance of such an implementation would be too sensitive to actual implementation parameters and would not be predictable. We do not expect the parallelized versions of such algorithms to achieve the same speedups as systolic algorithms for a given problem size. For real-time applications, we believe the additional effort to develop systolic algorithms with scalable parallelism is time well spent.

A variant of the QR algorithm which provides reasonable estimates of the noise subspace in signal processing problems and can be parallelized is described by Moonen, van Dooren and Vanpoucke [51]. This variant operates on a triangular matrix as opposed to a bidiagonal matrix, and skips the shifting operation which is needed for faster convergence. Hence, the convergence of this algorithm is not guaranteed in theory.

The Jacobi algorithm, on the other hand, has been found to be attractive for implementation on parallel machines. The Jacobi method with a variety of orderings may be implemented on systolic arrays. The Brent, Luk and Van Loan algorithm using a tournament ordering [11], a caterpillar track ordering by Modi and Pryce [48], the odd-even ordering used by Stewart [62], a minimal ordering for implementation on the hypercube by Gao and Thomas [26], a minimal ordering for implementation on a toroidal mesh by Delosme [18], and yet another minimal ordering by Eberlein and Park [22] are examples of
the variations possible on the same theme. The minimal orderings\(^2\) require the minimum computation and communication steps for implementation in an in-core fashion. All these orderings may be used to implement the Jacobi method in square arrays with dimensions upto \(N/2 \times N/2\), where \(N\) is the dimension of the matrix. These methods assume a pre-QR step in the case where the number of rows is greater than the number of columns. The triangular array by Luk [46] combines the two steps in an array where the triangular nature of the matrix is preserved by using odd-even ordering.

The Hestenes method [37] and the one-sided Jacobi method [22] allow implementation on a linear array of upto \(N/2\) processors. Any of the above orderings may be used to implement the one-sided methods resulting in a variety of flavors of the same algorithm. A variation of the one-sided methods using a different ordering allows pipelining of the host to array communication along with the internal communication and computation [42]. We have shown that this array, in addition to being simple to implement, can outperform the in-core methods using minimal orderings for any given set of technology parameters when the host-to-array communication is amortized over multiple SVD computations.

For a recursive implementation, the original work by Bunch and Neilsen [12] would result in essentially a \(O(N^3)\) algorithm with a small constant for the exact SVD update. However, it has been shown [49] that essentially a constant number of Jacobi steps per update are sufficient to maintain a reasonable approximation to the SVD. The resulting array [50] is an extension of the triangular array using odd-even ordering in which all the operations are completely pipelined to result in a total of \(O(N^2)\) operations per update and a constant throughput independent of the problem size on an array with \(O(N^2)\) processors assuming the host to array communication bandwidth is scaled linearly with the problem size. A numerically robust version of this array, which may in addition be implemented in CORDIC has been developed by Vanpoucke, Moonen and Deprettere [68].

\(^2\)These orderings have been called “optimal” in these papers. However, they are optimal only in the class of “in-core” algorithms as shown by Kota and Cavallaro [42]. So we will use the term “minimal” to describe these orderings.
these arrays have been developed for the exponential downdating formulation, essentially the same array may be used for sliding window SVD.

Algorithms to separate the signal and noise subspaces

Significant computational savings may be achieved if the backend estimator does not require the singular values and relies only on the singular vectors. For such estimators, it is enough to separate the bases for the signal and noise subspaces. Sometimes it may not even be necessary to compute an orthonormal basis, though that would be preferable. In Table 5.2 the algorithms which fall in this category are shown to compute only the matrix \( V \), which is an orthogonal basis with the signal and noise subspaces separated.

The rank-revealing QR decomposition (RRQR) [15, 7] performs a QR decomposition with column pivoting. The most elementary form of pivoting where the column to exchange is determined by the largest element along the diagonal, is really a rudimentary condition estimator [6]. This form of QR decomposition with column pivoting is described in most texts [32]. However, the separation of the signal and noise subspaces obtained using this simplistic method is not sufficient for our applications. Using another form of the condition estimator, Bischof and Shroff derived an algorithm to perform rank-revealing QR decomposition. The columns to permute are determined by a condition estimator. The resulting algorithms though attractive on serial machines, are not easily parallelizable.

The URV and ULV decompositions [58, 59, 60] are really a class of decompositions. These are two-sided decompositions with triangular factors. The RRQR may be regarded as a special case where the right hand factorization is restricted to be a permutation, while the SVD may be considered a special case where the triangular factor is restricted to be diagonal. Algorithms for the URV and ULV decomposition exhibit \( O(N) \) parallelism and may be implemented on a linear array of processors\(^3\).

\(^3\)It is not clear at this time, if the condition estimation step can be pipelined with the remaining computation. The algorithm described by Stewart [60] does not explicitly consider parallelization or the pipelining of the condition estimator.
An alternative to these algorithms are the Schur methods to compute the subspace [65, 33]. These methods may be compared in complexity to the RRQR algorithms. These algorithms could be useful in practical signal processing systems. However, the numerical properties of these algorithms are still under investigation.

The spherical subspace tracking algorithms are another class of low complexity algorithms [17]. A stable and parallel version of these algorithms has also been reported [67].

All these factorizations are still applicable with the recursive formulations. In such cases it is possible to update the resulting subspaces to reflect new data.

**Algorithms to compute the signal subspace**

A careful examination of backend algorithms such as MUSIC reveals that such algorithms typically make use of only one of either the signal subspace or the noise subspace. Hence, further computational reductions can be achieved through algorithms which can compute only the singular vectors corresponding to signal subspace or the noise subspace. Several algorithms have been developed based on Lanczos procedures. A good review of these algorithms can be found in Comon and Golub [16]. A similar algorithm called RLS-based subspace tracking [74] algorithm has recently been developed and may be used to track a signal subspace of a predetermined rank. The resulting algorithm may be viewed as a stochastic optimization technique, with a unique solution under stationary conditions. In the CDMA acquisition problem this algorithm would be applicable if the signal subspace is used instead of the noise subspace.

**5.5.3 A qualitative evaluation of front-end algorithms**

The algorithms for the frontend and the situations which will be encountered in practice, are so varied that there isn’t a single acceptable solution that is applicable in all situations. In this section, we will present a qualitative evaluation of the various algorithms when examined from various viewpoints including numerical stability, accuracy of the subspace
estimates, computational complexity, hardware complexity, maximum available parallelism and implementation complexity. It should be noted that although a non-implementation specific quantitative comparison is possible to a certain degree, a useful comparison is possible only when all the parameters of the system are completely specified.

In this section, we do not attempt to describe the details of the algorithms except for a brief sketch of each algorithm. However, the reader is expected to be familiar with all the details of the algorithm being discussed in order to appreciate many of the subtle distinctions and comparisons presented herewith. The references cited in each subsection are an excellent source for details of the original algorithms.

**QR Algorithm**

The QR algorithm is a popular algorithm for implementation of the SVD on a serial machine. We will not consider this algorithm in detail, since such a treatment is beyond the scope of this work. The interested reader is referred to Golub and van Loan [32] for references to the detailed steps of this algorithm. An important substep of this algorithm to ensure convergence is the shift strategy. However, this adds a backward dependence which doesn't allow the algorithm to be parallelized effectively. Implementation of the algorithm in parallel without shifts (or equivalently a shift of zero) is possible as shown by Moonen, van Dooren and Vanpoucke [51]. In spite of its limitations, such an algorithm could suffice in practical signal processing applications.

One of the practical considerations of such an implementation is the complexity of the controller, since it is necessary to track the global timesteps as well as the local timesteps. The processors need the information about the overall problem size in order to locally compute the number of substeps to perform. This implies that it would be necessary in practice to setup the problem size statically or at least minimize the number of times the problem size is varied.
Another practical issue is how the required singular vectors are piped out of the array. The information regarding the desired singular vectors is available in the singular value information along the diagonal. The singular values get scrambled every time step due to the outer rotations used to ensure that all possible pivot pairs of the odd-even ordering occur every sweep. This requires global information, since it is not possible to predict when and where the smallest singular values will occur along the diagonal.

Hence, although the QR algorithm with bidiagonalization followed by the Golub-Kahan step is attractive for serial implementation, the same algorithm does not exhibit the fine-grained parallelism required for practical signal processing applications. Even the best parallel version of the algorithm developed to date, does not exhibit any advantages over an implementation of the Jacobi method and hence is not recommended for parallel implementation.

In-core Jacobi methods

There is extensive literature dedicated to the development of in-core Jacobi methods. These algorithms typically assume that a matrix of given dimensions is available and it is necessary to compute the SVD of this matrix. In addition, it is assumed that the matrix has already been loaded into the array and distributed among the processors. The final result will also be distributed among the processors of the array.

One of the reasons the Jacobi algorithm has attracted a lot of attention is the high-degree of parallelism it exhibits. The parallelism arises from the variety of orderings under which the algorithm converges. In practice, Forsythe and Henrici [25] have conjectured that any cyclic ordering in which every pivot occurs will result in convergence. Theoretically, only some of these orderings have been shown to converge. There do exist orderings for which the Jacobi algorithm has been shown not to converge [35, 36, 47], but these non-convergence examples are extremely rare and are unlikely to occur in practice.
In addition to these practical considerations, numerically the Jacobi method has been shown to be superior to the QR algorithm [19]. However, such superiority is only of theoretical interest to someone trying to implement practical signal processing applications, since the superior numerical performance is dwarfed by the errors which arise from other sources. Hence, in effect it is akin to solving the wrong problem extremely accurately. So we can discount the superior numerical performance of Jacobi methods as a non-issue as far as our application is concerned.

The basic Jacobi algorithm consists of a series of sweeps, each of which consists of a series of \(n(n-1)/2\) subsweeps which differ only in the pivot pair chosen for that step. Each subsweep consists of computing two-sided Givens rotations which reduce the magnitude of the pivot pair, and applying these rotations. The rotations are computed and applied to the result of the previous subsweep. Parallelism results when the pivot pairs are chosen such that successive subsweeps may be performed in parallel. Several such sweeps are required for convergence. It has been conjectured that \(O(\log n)\) sweeps suffice in practice and hence may be pre-determined [11].

At each subsweep there are two solutions for the pair of Givens rotations [25] which completely annihilate the off-diagonal elements. If complete annihilation is not necessary (for instance, if it is computationally efficient to merely reduce the off-diagonal elements instead of completely reducing them to zero), then an infinite number of solutions exist. The two solutions which completely annihilate the off-diagonal pivot pair, are related in that one of them can be considered a permutation of the other. These two rotations are called inner rotations and outer rotations. Sometimes a cyclic ordering is implicit through the use of outer rotations, while on other occasions, the cyclic ordering is explicit through the use of inner rotations. In general, any Jacobi ordering which converges using outer rotations has a corresponding converging Jacobi ordering using inner rotations. For the interested reader, the seminal paper by Forsythe and Henrici [25] and an early paper on parallel Jacobi methods by Brent, Luk and van Loan [11] are recommended.
For implementation we do not recommend the early parallel arrays and many of the successors, which differ primarily in the pivot ordering. However, these arrays are important from a historical viewpoint since they directly led to the development of more recent arrays which are arguably some of the most efficient available for the problem at hand. We will evaluate some of these algorithms and arrays in the following sections.

The pipelined SVD algorithm

If a reasonable assumption is made that the SVD is one stage of a larger signal processing system, there are two ways in which the entire system can be implemented and thus effect the SVD algorithm. If all the available processors are used to implement all portions of the overall signal processing, it will be necessary to synchronize the processors after each step, redistribute the data (since it is generally not the case that all the steps work with the same distribution of the data) and following the redistribution, start the processors on the following stages. Alternately, if the overall signal processing has been distributed among a disjoint set of processors, the processors implementing the SVD have to be synchronized at the beginning of each set of data, the results flushed, and a new matrix loaded before the next SVD computation is started. In either case, the data loading and unloading operations are unavoidable and should form an integral part of the SVD algorithm and hence not ignored in any valid comparison.

The pipelined SVD algorithm [42] alleviates this disadvantage of the in-core Jacobi algorithms: it incorporates the input matrix loading and the result vector unloading operations as part of the algorithm. The algorithm uses a different ordering of the pivot pairs in the Jacobi method to achieve the desired effect. The resultant algorithm is extremely simple and shows asymptotic speed advantages especially in communication limited systems.

The algorithm maps most naturally onto a linear array of processors with unidirectional links. Input vectors enter the system from one end and the results flow out of the other. The maximum number of processors in the linear array for a given sized problem is $NS/2$,
where $N$ is the number of columns in the matrix, $S$ is the number of sweeps to be performed (typically on the order of $\log N$, though the exact number of sweeps for a given problem size requires simulations). For practical sized problems and practical throughputs, we rarely expect to require more processors. Hence, the problem in practice is not how to extract more parallelism, but rather how one maps this "logical" array onto fewer processors.

A simple example will help substantiate our claim. A typical requirement in the near-far resistant CDMA synchronization problem is an orthogonal basis of a $32 \times 32$ matrix where columns of the matrix are received at the rate of 28800 columns/sec, i.e. 1.8432 Mbytes/sec assuming 16 bit precision at the input (this corresponds to a 9600 bit/sec communication link for cellular phone quality speech, followed by a rate 1/3 convolution code). If we decide to use an off-the-shelf processor such as the TMS320C40, we can expect a computational speed of 40 MFlops, and a communication speed of 20 Mbytes/sec. This gives us $40 \times 10^6/28.8 \times 10^3 = 1380$ cycles per communication step. Using a Hestenes algorithm, the computation for a pair of columns is one inner product (32 multiply-accumulate operations), an inverse-tangent computation (approximately 10 cycles) and 34 vector rotation operations (4 multiplies and 2 additions per each rotation). Using handcoded assembly code, we should be able to support at least 3 column pairs and at most 6 pairs per processor.

The simplicity of the "logical" array of $NS/2$ processors allows several different ways to map the array onto fewer processors. One way is to view the physical array as a portion of the fully unrolled array. Multiplexors at the two ends decide whether to recycle the output from one end of the array back into the input of the array, or allow external communication. The advantage of this form of mapping is that the processors do not need to know the exact number of columns in the matrix and hence dynamically varying the number of columns corresponding to variations in input would require only the switches to be reconfigured. On the flip side, the communication links need to support the full communication rate required in the fully unrolled array.
Another mapping strategy available is to map $m$ successive processors of the logical array onto one processor. This now reduces the communication requirements by the factor $m$, since only one column is transferred by any given processor for every $m$ pairs. This has the advantage that it eliminates the need for the switches. However, a change in the number of columns in the matrix requires all the processors to be reconfigured.

One of the principal disadvantages of the pipelined SVD algorithm is that the number of sweeps has to be an even number. This is because, the desired pipelining property happens only when the number of sweeps is even. If an odd number of sweeps achieves the desired accuracy, then the additional sweep could represent a significant increase in the computation requirements.

Another disadvantage occurs when the number of columns exceeds the number of rows. The number of sweeps performed is a function of the number of columns (not the number of rows). Hence, if the number of columns exceeds the number of rows (as is always the case in subspace-based signal processing), this represents an additional source of wasted computation. To reduce the processor requirements, it is suggested that the SVD be performed on the transpose of the matrix. The implications of this transpose are: a potential increase in the overall latency and the need to compute both left and right singular vectors. First, it is necessary to include a buffer at the input to store the columns of the matrix until all the vectors are received before the matrix can be transposed and the SVD computation can be started. This is because the natural order in which data is received in the subspace-based synchronization problem is by columns, rather than by rows. Hence there is a latency associated with this buffer. The other source of increase in latency is due to the requirement that we compute the column space of the original matrix, which is the row space of the transposed matrix. Hence, it now becomes necessary to design the array to compute the complete SVD decomposition, not just the left singular vectors and the singular values. This represents additional computation and adds to the overall latency. On the other hand, there is a decrease in latency due to the fact that the implicit eigenvalue decomposition of
a much smaller matrix is being computed. Although, this certainly reduces the processor
requirement, we expect to see an overall latency increase.

Rank-revealing QR algorithms

The rank-revealing algorithms arise from the observation that frequently, it is not neces-
sary to compute the complete SVD, but rather separating the signal and noise subspaces
is sufficient. The usual QR decomposition provides the basis for the the column space of
a matrix, without revealing the bases of the signal and the noise subspaces. When column
pivoting is incorporated, however, the QR decomposition can be made to separate these two
subspaces. A key part of the rank-revealing QR algorithms, then is to determine the column
permutation required to force the QR decomposition to exhibit the desired separation. The
fundamental idea here is that there exist cheap condition estimators, which can estimate
the smallest singular value in practice with approximately $O(N^2)$ computations. This in-
formation can be used to determine whether a column should belong to the signal subspace
or the noise subspace. A limitation of this approach is that there is a limit on the degree of
separation that one can expect since the right-hand side orthogonal matrix is constrained
to be a permutation matrix. The other disadvantage is that no parallel algorithm has been
described for this operation. It appears that the condition estimation requires global in-
formation and an efficient scalable parallel algorithm is difficult to obtain. The algorithm
extends to recursive formulations as demonstrated by Bischoff and Schroff [7]. Some degree
of parallelism is achievable through local pivoting [4]. However, this algorithm is difficult to
extend to custom implementations. In practice, our conjecture is that since the right-hand
side decomposition is restricted to a permutation, the separation in the subspaces will not
be sufficient for our problem.
The URV and ULV decompositions

The URV [58] and the ULV decompositions [59] are similar, but not the same, from a practical point of view. The URV decomposition should result in a good estimate of the right singular space, while the ULV decomposition should result in a good estimate of the left singular space. The complexity of these algorithms is comparable to the rank-revealing QR decompositions, while providing excellent separation of the signal and the noise subspaces. A computation versus accuracy tradeoff is possible because of the notion of "refining" iterations, which can improve the quality of the decomposition. The algorithm once again relies on condition estimators to make key decisions, and hence can be only as reliable as the underlying condition estimator. However, we do not expect this to be a problem in practice.

A more practical issue though is that the URV and ULV decompositions require an estimate of the noise variance in order to determine if a column belongs to the signal or noise subspaces. Since this value is not known a priori, and could change with time, the use of URV and ULV decompositions would be restricted to high SNR problems, where an error in estimating the noise variance need not be catastrophic.

Although, a parallel implementation of the algorithm has been previously described [60], the implementation ignored the parallelization of the condition estimator. It can be argued, however, that the parallelization of the condition estimator only results in a constant factor reduction in the overall computation and hence is not essential.

For recursive formulations, both the URV and ULV decompositions may be updated as described by Stewart [58, 59]. The URV decomposition is slightly easier to update than the ULV decomposition. Hence, there is a tradeoff between the accuracy of the noise subspace estimate and the complexity of updating.

Qualitatively, there does not appear to be any reason not to use the URV decomposition whenever it is applicable provided there is sufficient signal to noise ratio. It provides considerable reduction in computation. On the flip side, it is difficult to analytically estimate
how much the inability to pipeline the condition estimator will offset these gains and hence
we reserve our comments on that.

The approximate SVD updating algorithm

The approximate SVD updating algorithm [49] stands out among many of the other alter-
natives. It builds upon the triangular array for the SVD described by Luk [46]. It
elegantly combines a matrix-vector multiplication array, the QR updating algorithm by
Gentleman and Kung [27], with the parallel Jacobi algorithm of Luk using the odd-even
ordering [62]. The key idea which reduces the complexity of SVD updating considerably
is that the iterations of the Jacobi algorithm may be spread out over a large number of
updates. The excellent convergence properties of the Jacobi algorithm ensure that the re-
sulting updated matrix is an approximate SVD decomposition. This idea of spreading the
sweep among a large number of updates results in an algorithm that requires only $O(n^2)$
operations per updates as opposed to $O(n^3)$ which would be required if a complete decom-
position to machine precision is to be performed at each update [12, 13]. A systolic array
can be designed [50] to implement the approximate updating algorithm exploiting $O(n^2)$
parallelism to result in an array with $O(1)$ throughput provided the I/O is scaled as $O(n)$,
i.e. linearly with the size of the problem. Though the original systolic array included a
Gram-Schmidt re-orthogonalization, a later improvement which stores the singular vector
matrix as a product of Givens rotations stabilizes the algorithm elegantly without recourse
to re-orthogonalization [68]. One of the advantages of the approximate SVD updating algo-


rithm is that it provides approximate values of the singular values in addition to the singular
vectors. In addition, a tradeoff between computation and the degree of approximation is
possible. Hence, the algorithm is applicable when a recursive formulation is used with
a backend that utilizes the maximum-likelihood MUSIC technique, or any other backend
estimator that requires both the singular values and singular vectors.
There are situations when the approximate SVD updating algorithm is the only algorithm applicable. In the other situations, it is difficult to analytically gauge the effect that some of the disadvantages cited above will have on the overall system design, to compare it to the other algorithms. However, there do exist situations when a cheaper solution will suffice.

5.6 Quantitative analysis

5.6.1 Rectangular vs. Exponential window

Fig. 5.2 and Fig. 5.3 show the results of simulations performed to determine the effect of using a rectangular sliding window and exponential window and their corresponding updating versions of the algorithms. The goal of these algorithms is twofold. First, we want to compare the result of using a rectangular window versus an exponential window, since they are clearly not equivalent although they may be used for the same task. The second goal is to determine how much degradation to expect when the approximate decompositions are used.

Since the backend is not linear, comparing the error in the computed subspaces using the different methods may not reveal all the possible sources of error. Hence these simulations include the effect of the backend. We have arbitrarily chosen the deterministic MUSIC algorithm which will be discussed in great detail in the next chapter.

The sliding window methods were simulated using a window length, \( L \), of 200 samples. The exact method uses the linpack SVD algorithm to compute the exact SVD of randomly generated data.

The data corresponding to the exact sliding window method was obtained by performing an exact SVD (in floating point) using the Linpack routines available in MATLAB and using the computed singular vectors to estimate the time delays. The observation vectors are scaled by the appropriate power of the forgetting factor \( \beta \).
Figure 5.2: Probability of acquisition and standard deviation of the error in the delay estimate over a range of signal to noise ratios for different frontend algorithms (K=5).
Figure 5.3: Probability of acquisition and standard deviation of the error in the delay estimate over a range of signal to noise ratios for different frontend algorithms ($K=13$).
The updating version of the rectangular window implements the algorithm described in Chapter 4 with the hyperbolic rotations computed using the Chambers method (fully floating-point implementation).

The updating version of the exponential window implements the algorithm described by Moonen, van Dooren and Vandewalle [49].

The probability of error and the standard deviation of the delay estimates were obtained through 1000 trials for each value of SNR from 0 to 20dB for a given number of users $K$. Each trial involved the following tasks:

- The spreading waveforms for the users were chosen randomly from the 33 possible gold codes for $N = 31$.
- Random time delays uniformly distributed between 0 and 31 were independently generated for each user.
- Random data for $L$ samples were generated for all the users. Each data element is generated as an independent random variable with equal probability of a $-1$ or a 1.
- The CDMA observation vectors obtained assuming a rectangular received waveform were then generated. In general, the observation over each chip interval involves contribution from two adjacent chips for each user and the observation vector itself consists of the contributions from adjacent bits for each user.
- The delay estimate for user 1 at time $t = 200$ was computed for all the different methods and the statistics accumulated.
- The probability of error is estimated as the fraction of times the user was acquired, where a user is said to be acquired only if the estimated delay is within one chip of the actual delay for that user.
• If the user is acquired, the delay estimate is used to update the standard deviation of the error in the estimate. Hence, the standard deviation curves are believable only at the higher signal to noise ratios when the user is acquired most of the time.

• All users were assumed to be transmit at the same power relative to the noise power.

5.6.2 Group delay of the estimates

The plots shown in the previous sections use arbitrarily chosen values for the size of the sliding window, \( L \), and the forgetting factors. However, such a comparison is not fair since by using a \( \beta \) very close one, the curves for the two methods can be brought arbitrarily close. The true differences between the two methods surface not in the experiment described above but when the time delays vary with time.

One of the important considerations to be used for such a comparison is the group delay. When the delay is varying, since a different estimate of the delay is computed at each observation instant, it is important that the appropriate computed delay be used for any given sample during detection. From the properties of the filters which each of these methods correspond to, we would expect this delay to be the same as the group delay of the corresponding filters.

Table 5.3: Comparison of a rectangular FIR transfer function and an IIR transfer function with one pole. \( L \) is the length of the FIR. \( \beta_n \) is the location of pole that results in the closest transfer function from \( \omega = 0 \) to \( \omega = \omega_{3dB} \). \( \beta_d \) is the location of the pole which results in the same delay as the FIR for small frequencies.

<table>
<thead>
<tr>
<th>( L )</th>
<th>( \beta_n )</th>
<th>( \beta_d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>.9001</td>
<td>.9394</td>
</tr>
<tr>
<td>63</td>
<td>.9495</td>
<td>.9692</td>
</tr>
<tr>
<td>127</td>
<td>.9744</td>
<td>.9845</td>
</tr>
<tr>
<td>200</td>
<td>.9841</td>
<td>.9901</td>
</tr>
<tr>
<td>255</td>
<td>.9873</td>
<td>.9922</td>
</tr>
<tr>
<td>511</td>
<td>.9937</td>
<td>.9961</td>
</tr>
<tr>
<td>1023</td>
<td>.9968</td>
<td>.9980</td>
</tr>
</tbody>
</table>
As described earlier in this chapter, the rectangular window should exhibit the a delay of $L/2$ for all frequencies in the time delay until a maximum frequency after which the performance degrades to an unacceptable level. The exponential window should exhibit a delay of approximately $\beta/(1 - \beta)$, for small frequencies. However, the maximum frequency upto which the exponential window method can provide acceptable performance is determined both by the magnitude response and the phase response, while we can expect it to be determined purely by the magnitude response for the rectangular window.

One way to determine a starting point at which the two responses should give us similar performance is by computing the values of $L$ and $\beta$ which minimizes the norm of the difference between the two magnitude responses. Table 5.3 tabulates some numerically computed values of $\beta_n$ which minimizes the norm of the difference between the computed FFTs for some given values of $L$. Although this table optimized the $\beta_n$s to minimize the difference between the two transfer functions upto the 3dB point of the FIR, in practice, given the largest frequency component in the time delay, the comparison should be upto that frequency, since that would constitute the passband. The table also tabulates the values of $\beta_d$ which results in the same group delay.

5.7 Summary

This chapter described how near-far resistant CDMA acquisition can be implemented using a number of algorithms which were originally developed for direction finding. We focussed on the frontend processing which involves estimation of the signal and/or noise subspaces. The design of these systems involves making decisions based on signal processing as well as computer architecture considerations. A summary of the various design decisions which need to be made in the implementation of frontend algorithms, is presented in Table 5.4 and Table 5.5.

A key signal processing decision is to decide at the highest level whether the problem should be posed using a recursive formulation or a non-recursive one. The non-recursive
Table 5.4: Frontend Design Space

Problem Formulation:
- Block formulation vs. Recursive Formulation
- Exponential Window (IIR) vs. Sliding Window (FIR)

Degree of Subspace decomposition required:
- Complete Decomposition - Jacobi SVD, QR SVD, Approx. SVD updating
- Separation of signal and noise bases - RRQR, URV, ULV, Spherical, Lanczos
- Signal subspace basis only - RLS-based updating, Lanczos

Degree of parallelism:
- Linear Arrays - Hestenes algorithm, 1-sided Jacobi, Pipelined SVD, URV
- Square arrays - 2-sided Jacobi, Approx. SVD updating

Degree of pipelining:
- Coarse grained - Few processors operating on entire subproblems
- Medium - Processors operate on submatrices/columns
- Fine grained - Processors operate on tiny blocks such as $2 \times 2$ submatrices
- Bit-level pipelining: Units of communication are bits

Hardware mapping strategy:
- GSLP vs. GPLS mapping
- Custom VLSI vs. General-purpose DSP
Table 5.5: Algorithm specific design options

Pipelined SVD:
  - Column basis computation vs. row basis computation

Approximate SVD updating:
  - Choice of number of Jacobi steps per update
  - Storage of subspace basis – factored vs. conventional

URV/ULV updating:
  - Choice of condition estimator
  - Choice of threshold (i.e. a priori SNR) for deflation
  - Number of refinement steps per update

RRQR updating:
  - Choice of condition estimator
  - Row basis vs. column basis
  - Orthogonal basis vs. non-orthogonal

RLS-based subspace updating:
  - Choice of algorithm to orthogonalize the basis
formulations will require less computation and hence are attractive from a hardware point of view. However, the advantage of recursive algorithms is that the estimates can be updated every sample time as opposed to once per block. This means that the recursive algorithms should in general perform better than the block methods when the parameter to be estimated is varying slowly over time. This decision of recursive versus block formulation is crucial since it ultimately determines the algorithms which are applicable for implementation. While any recursive algorithm can be used where a block method would suffice, there is invariably a hardware penalty associated with such a decision. Many of the block methods do not allow easy updating and consequently are not a good choice when a recursive formulation is desired. Thus a judicious distinction between the two approaches and avoiding the use of the wrong architecture for the problem can result in an order of magnitude reduction in computation.

If the problem constraints demand a recursive formulation there is one more decision to make as to whether a sliding window (FIR) or an exponential window (IIR) is required. Here numerical considerations favor the exponential window methods. However, if practical signal processing requirements such as a linear phase requirement impose a sliding rectangular window, implementation is still possible provided careful attention is paid to the numerical aspects.

The architectural decision as to the algorithm (and the array) to choose is highly problem dependent. The primary tradeoff here is the one of hardware versus throughput. At one extreme, a low cost solution would drive one to use as few processors (preferably a single processor) as the problem will allow. Linear arrays would be attractive in such applications since the very nature of these arrays keeps most of the communication localized within the processor. The other end of the spectrum is extremely high throughput which would require fine-grained parallelism supported by square arrays. Arrays have been developed which allow parallelism at the level of a single 2 x 2 Jacobi rotation on each processor or at an even finer levels where pipelining is at the bit level. Most applications can be expected to
lie somewhere in this vast spectrum. For instance, the CDMA acquisition problem results in matrix dimensions which allow implementation using fairly large linear arrays for practical throughputs.

Another decision which affects the frontend algorithm is the choice of the backend algorithm. In general, it is possible to trim all unnecessary computation, allowing a relaxation in any constraints to translate directly into hardware savings. For instance, if the backend does not require singular values, then there exist algorithms which allow one to merely separate the subspaces. If the backend utilizes only the signal subspace, then further savings are possible by using algorithms which compute precisely that and no more.

The qualitative analysis provided in this chapter of several frontend algorithms would serve to alert the reader to difficulties which are not apparent at first glance. Much work remains to be done to perform meaningful quantitative analysis based on specific problem requirements. An early offshoot of this work is the performance analysis of the QR algorithm in CDMA acquisition performed by Haller [34]. In the next chapter we will investigate the implementation options in the backend.
Chapter 6

Architectures for CDMA Acquisition - Backend

6.1 Introduction

The focus of this chapter is the second portion of the subspace-based algorithms for CDMA acquisition, which we call the *backend computation*. As was described earlier, subspace-based algorithms naturally divide into two subproblems with very distinct characteristics. The two subproblems are nearly independent of one another and are amenable to an independent treatment.

In this chapter we develop a parallel architecture which unifies three different formulations of the backend optimization for the CDMA acquisition problem. The architecture separates the backend problem into a matrix multiplication, inner product computation and a polynomial optimization subproblems which are all seamlessly pipelined on a rectangular array of processors.

In the past there have been few attempts to parallelize the backend portion of subspace-based algorithms. Although parallel algorithms have previously been developed to solve the generalized eigenvalue problem [45, 69], which forms a crucial part of the ESPRIT formulation used for direction finding, that formulation is not applicable in the backend of the CDMA acquisition problem because of the absence of a shift-invariance structure. There hasn’t been much focus on developing parallel algorithms for other non-linear optimization problems which constitute estimation procedures such as MUSIC or MODE for direction of arrival problems. This is primarily due to the absence of any application-independent structure in these problems.
Although the solution of a general non-linear optimization problem requires a global search for the solution, the complexity of the problem can be considerably reduced through the use of structure imposed by the specific problem at hand. In the CDMA acquisition problem, we exploit the structure of the problem to develop an architecture which facilitates implementation of the three different formulations we consider.

The estimation procedures we examine are what we call the deterministic MUSIC, maximum-likelihood MUSIC and the multipath-estimator which is an extension of the deterministic estimator to the multipath channel [2]. We consider the estimators described in this chapter to be similar to the MUSIC estimator [56] developed for direction of arrival problems. A key feature of these algorithms is that they decouple the users and estimate each user independent of all the other users. This decoupling results in significant savings in computation without appreciable degradation in the estimates of the time delays.

The deterministic MUSIC estimator results from non-stochastical properties of the estimated noise space basis vectors: the transmitted CDMA signals should have the least energy in the noise subspace, since the noise subspace is by definition a least squares estimate of the null space of the signal matrix. Hence, the deterministic estimator poses the estimation problem as determining the time delays which result in the least energy in the estimated noise subspace. The assumption involved in this formulation (and in the other formulations we consider) is that the mapping from the time delays to the matrix $A(\hat{r})$ is invertible.

The maximum-likelihood estimator utilizes additional knowledge about the probability distribution of the vector obtained through the projection of a vector into the estimated noise space of the observation matrix. This knowledge of the distribution of the vector allows the estimation of the parameters in a maximum-likelihood sense.

In a multipath channel, due to the complexity of the maximum-likelihood MUSIC formulation, the simpler formulation, namely deterministic MUSIC, is extended to result in a viable procedure for the delay estimation. The channel is assumed to consist of a finite
number of paths, each of which is modeled as an attenuation and an arbitrary delay. The procedure then estimates the strongest paths. The multipath procedure is separated into two parts to further simplify the problem. The first part estimates a channel without any constraints on the structure of the channel, which is then refined in the second step by a series of least square fits to result in a channel estimate in the desired form.

We discuss parallel algorithms in detail for the three formulations in the following sections. We make a simplifying assumption that the received baseband chip pulse is rectangular to facilitate a closed form solution.

6.2 Deterministic MUSIC estimator

To implement the deterministic MUSIC algorithm, it is necessary to solve the following optimization problem for each user \( k \):

\[
\hat{\tau}_k = \arg \min_{\tau \in [0, N T_d)} \| a_k(\tau)' S_{\perp} \|^2.
\]  

(6.1)

Here, \( \tau \) is the time delay variable that is to be estimated. \( S_{\perp} \) is a matrix whose columns form an orthonormal basis for the noise space. The vector/matrix \( a_k(\tau) \) consists of the columns of the matrix \( A(\tau) \) corresponding to user \( k \). From Eq. 2.37, \( a_k(\tau) \) can be written as:

\[
a_k(\tau) = \begin{pmatrix}
    a_k^{(N-(\tau_k)-1)} \phi_{r,k} + a_k^{(N-(\tau_k))} \phi_{l,k} & 0 \\
    a_k^{(N-(\tau_k))} \phi_{r,k} + a_k^{(N-(\tau_k)+1)} \phi_{l,k} & 0 \\
    \vdots \\
    a_k^{(N-1)} \phi_{r,k} & a_k^{0} \phi_{l,k} \\
    0 & a_k^{0} \phi_{r,k} + a_k^{1} \phi_{l,k} \\
    0 & a_k^{1} \phi_{r,k} + a_k^{2} \phi_{l,k} \\
    \vdots \\
    a_k^{(N-(\tau_k)-2)} \phi_{r,k} + a_k^{(N-(\tau_k)-1)} \phi_{l,k}
\end{pmatrix}.
\]  

(6.2)
From Eq. 2.35, for a rectangular baseband chip waveform \( g(t) \), the correlations \( \phi_{l,k} \) and \( \phi_{r,k} \) take the form:

\[
\phi_{l,k} = (1 - \delta) T_c \\
\phi_{r,k} = \delta T_c,
\]

where \( \delta = (\tau_k - n \tau_k)/T_c \) is the fractional part of the user's delay normalized to the chip duration, \( T_c \).

An alternative formulation of \( a_k(\tau) \) gives us a reduction in computation without any significant degradation in the estimated time delays. This alternate formulation described by Bensley and Aazhang [3], utilizes the fact that the two columns of \( a_k(\tau) \) are always nearly orthogonal due to the structure shown in Eq. 6.2. This alternative formulation defines \( a_k(\tau) \) as:

\[
a_k(\tau) = \begin{bmatrix}
a_k^{(N-\lceil \tau_k \rceil - 1)} \phi_{r,k} + a_k^{(N-\lceil \tau_k \rceil)} \phi_{l,k} \\
a_k^{(N-\lceil \tau_k \rceil)} \phi_{r,k} + a_k^{(N-\lceil \tau_k \rceil)} \phi_{l,k} \\
an_k^{(N-1)} \phi_{r,k} + a_k^0 \phi_{l,k} \\
an_k^2 \phi_{r,k} + a_k^1 \phi_{l,k} \\
\vdots \\
an_k^{(N-\lceil \tau_k \rceil - 2)} \phi_{r,k} + a_k^{(N-\lceil \tau_k \rceil - 1)} \phi_{l,k}
\end{bmatrix} = \begin{bmatrix}
a_k^{(N-\lceil \tau_k \rceil - 1)} \\
a_k^{(N-\lceil \tau_k \rceil)} \\
an_k^{(N-1)} \\
an_k^1 \\
\vdots \\
an_k^{(N-\lceil \tau_k \rceil - 2)}
\end{bmatrix} \delta + \begin{bmatrix}
a_k^0 \\
a_k^1 \\
a_k^2 \\
\vdots \\
a_k^{(N-\lceil \tau_k \rceil - 1)}
\end{bmatrix} (1 - \delta).
\]

(6.3)
For e.g., suppose the spreading code for user $k$ is $a_k = (1 \ 1 \ -1 \ 1 \ -1 \ 1 \ -1)'$ then for $\tau = 3.2$, the vector $a_k(\tau)$ may be derived as follows:

\[
\begin{pmatrix}
1 \\
-1 \\
1 \\
1 \\
-1 \\
1 \\
1
\end{pmatrix} = \begin{pmatrix}
1 \\
-1 \\
1 \\
1 \\
1 \\
1
\end{pmatrix} = \begin{pmatrix}
0.6 \\
-0.6 \\
0.6 \\
1 \\
0.6 \\
-0.6
\end{pmatrix}.
\]

For the rest of this chapter we will use this alternate formulation, since it considerably simplifies the presentation without any loss of generality.

Continuing with the optimization problem for deterministic MUSIC, we obtain for any integer $i \in [0, N)$, the minimization problem for $\tau \in [iT_c, (i + 1)T_c)$ as:

\[
\hat{\tau}_k^{(i)} = \arg \min_{\tau \in [iT_c, (i + 1)T_c]} \|a_k(\tau)'S_\perp\|_2^2
\]

\[
= T_c \times \left( i + \arg \min_{\delta \in [0,1]} \|((1 - \delta)a_k(iT_c) + \delta a_k((i + 1)T_c))'S_\perp\|_2^2 \right)
\]

If we denote $a_k(iT_c)'S_\perp$ by $b_{k,i}$, then

\[
\hat{\tau}_k^{(i)} = T_c \times \left( i + \arg \min_{\delta \in [0,1]} \|(1 - \delta)^2b_{k,i}'b_{k,i} + 2\delta(1 - \delta)b_{k,i}'b_{k,i+1} + \delta^2b_{k,i+1}'b_{k,i+1}\| \right)
\]

Hence once the $b_{k,i}'b_{k,i}$ and $b_{k,i}'b_{k,i+1}$ are computed for all $i$, the optimization within each chip reduces to the minimization of a quadratic polynomial, which may be computed in closed form.

The computation of these quantities may be achieved using the following steps:

- Form the matrix $A_k$ as

\[
A_k = (a_k(0) \ a_k(T_c) \ a_k(2T_c) \ \cdots \ a_k(NT_c - T_c)) \in \mathbb{R}^{N \times N}.
\]
This is a one-time data-independent step for each user. It is essentially a matrix whose columns are the shifted versions of the spreading code of the given user. This is known \textit{a priori} since this code is either assigned to a user statically at design time or dynamically when the user registers with the base station at the beginning of the communication.

- Project the matrix $A_k$ onto the basis for the noise subspace $S_\perp \in \mathbb{C}^{N \times (N-2K)}$.

\[
B_k = (b_{k,0} \ b_{k,1} \ b_{k,2} \ \cdots \ b_{k,N-1})' = A_k' S_\perp
\]  

(6.6)

Note that there is a tradeoff between computation and latency in this step. If the singular vectors in the frontend are not sorted, then it is necessary to wait until all the singular vectors have been computed to form the matrix $S_\perp$. However, at the expense of additional computation, it is possible to postpone the sorting step. Suppose $\Pi$ is the permutation that sorts the matrix of singular vectors, $V$, into the signal and noise subspaces. Then,

\[
V\Pi = \begin{pmatrix} S & S_\perp \end{pmatrix}
\]  

(6.7)

Note that $\Pi$ can be determined only after all the singular values have been computed. Then,

\[
A_k' V \Pi = A_k' \begin{pmatrix} S \\ S_\perp \end{pmatrix} = \begin{pmatrix} \cdot \\ B_k \end{pmatrix}
\]  

(6.8)

Hence, the matrix $B_k$ can be recovered from $A_k' V$ by applying the permutation $\Pi$. This allows the sorting to be postponed to after the matrix multiply. A computation increase occurs when the sorting is postponed because it becomes necessary to project the $A_k$ matrix onto all the singular vectors instead of just the noise vectors and then throw away part of the computation.

- Compute $b_{k,i}^i b_{k,i}$ and $b_{k,i}^i b_{k,(i+1)\%N}$ for all $i$

Fig. 6.1 shows an architecture of the backend for a single user to compute the deterministic MUSIC metric. A detailed description of the same array is provided in Fig. 6.2.
Figure 6.1: Parallel architecture to implement deterministic MUSIC backend for a single user

The maximum parallelism achievable using such an architecture is $O(N^2)$. However, practical systems would in simulate such a "virtual" array using fewer physical processors. The architecture can be easily mapped onto a linear array of processors.

The architecture essentially consists of a square matrix multiplication array (step 1), a linear array of processors to perform dot-products of vectors (step 2) and a linear array of processors to minimize quadratic polynomials (step 3). The computational complexity of each of these steps is shown in Table 6.2 and Table 6.3.

The matrix multiplication array stores the matrix $A_k$ distributed in the array. Columns of $S_\perp$ enter the array from the left (skewed to satisfy timing dependencies). The columns
Figure 6.2: Details of a parallel architecture to implement deterministic MUSIC backend for a single user.
of $B_k$ leave the array from the top: one element of the column vector from each column of the processors in the matrix multiplication array.

The data leaving each column of the matrix multiplication array is collected by a processor which computes the inner product of that vector. It is assumed that all the processors know the dimensions of the problem and hence can distinguish between data from successive iterations. The accumulated quantities are $b_{k,i}^t B_{k,i}$ for all $i$. In addition, these processors also receive the data from the right adjacent column (the rightmost processor receives data from the leftmost through an end-around connection) and accumulate a dot product of the two columns. This corresponds to the computation of $b_{k,i}^t B_{k,(i+1)%N}$ for all $i$.

The accumulated dot-products are then passed on to the processors in the topmost row in Figure 6.1. These processors solve a constrained quadratic polynomial minimization problem. The coefficients of the quadratic polynomial can be directly computed from the inner products. The resulting local minimum of the objective function is compared to the value received from the left processor and the smaller of the two values is selected. The selected value and the corresponding time delay estimate are then passed to the processor on the right, which performs a similar procedure. Hence the global estimate of the time delay is obtained from the rightmost processor on the top.

The entire computation for a single processor is thus pipelined and successive iterations result in newer estimates of the time delay. Once a user has been acquired, tracking would require all this computation only in two or three adjacent chips. This translates into two or three vertical slices of the above architecture. However, occasionally it may be necessary to re-acquire a user, in which case the optimization needs to be performed over the entire range of possible time delays.

6.3 Maximum-likelihood MUSIC

The ML-MUSIC algorithm and approximate ML-MUSIC algorithm require additional computation in the backend. This algorithm is equivalent to the MODE estimator presented
Figure 6.3: Parallel architecture to implement Maximum-likelihood MUSIC backend for a single user

by Stoica and Sharman [63]. The log likelihood function for a given user is [3]:

$$\Lambda(a_k) = -a(N - 2K)\log(a_k)'Wa_k) - L\frac{a_k(S_k'S_k'a_k)}{a_k)'Wa_k} \quad (6.9)$$

where,

$$\eta = \frac{1}{N - 2K} \sum_{k=2K+1}^{N} \sigma_k^2 \quad (6.10)$$

$$W = \eta \left[ \sum_{k=1}^{2K} \frac{\sigma_k^2}{(\eta - \sigma_k^2)^2} u_k v_k' \right] \quad (6.11)$$

$$= S \left( \begin{array}{c} \eta \sigma_1^2 \\ \eta \sigma_2^2 \\ \vdots \\ \eta \sigma_{2K}^2 \end{array} \right) \left( \begin{array}{c} (\eta - \sigma_1^2)^2 \\ (\eta - \sigma_2^2)^2 \\ \vdots \\ (\eta - \sigma_{2K}^2)^2 \end{array} \right) S' \quad (6.12)$$
This estimator results in a third-degree polynomial optimization problem within each chip and in that sense is similar to the deterministic MUSIC estimator. The quantities that need to be computed to form the third-degree polynomial within each chip are:

\[ w_{i,i} = a_k(iT)'W a_k(iT) \]

\[ w_{i,i+1} = a_k(iT)'W a_k(iT) \]

\[ n_{i,i} = a_k(iT)'S_{ik}'S_{ik} a_k(iT)' \]

\[ n_{i,i+1} = a_k(iT)'S_{ik}'S_{ik} a_k((i+1)T)' \]

Notice that \( n_{i,i} \) and \( n_{i,i+1} \) are the same quantities as \( b_{k,i}'b_{k,i} \) and \( b_{k,i}'b_{k,i+1} \) computed in the previous section. The additional quantities \( w_{i,i} \) and \( w_{i,i+1} \) may be computed in a similar manner by observing the following:

\[ w_{i,i} = a_k(iT)'W a_k(iT) \]

\[ = a_k(iT)'S \text{ diag} \left( \frac{\eta \sigma_i^2}{(\eta - \sigma_i^2)^2}, \frac{\eta \sigma_j^2}{(\eta - \sigma_j^2)^2}, \ldots, \frac{\eta \sigma_{2K}^2}{(\eta - \sigma_{2K}^2)^2} \right) S' a_k(iT) \]

Hence an essential computation required to compute the ML-MUSIC estimate is the matrix-matrix multiplication \( A_k'V \). The elements of each row can then be separated into signal and noise vectors. Then an operation similar to dot-products may be used to compute \( w_{i,i} \) and \( w_{i,i+1} \) from the signal vectors, while dot products of the noise vectors results in \( n_{i,i} \) and \( n_{i,i+1} \).

The architecture for ML-MUSIC (see Fig. 6.3), would then be identical to the architecture for deterministic-MUSIC (see Fig. 6.1) with the following exceptions:

- The frontend has to compute all the singular vectors and singular values. This rules out some of the frontend options such as URV, ULV, RLS-subspace tracking and rank-revealing QR.

- ML-MUSIC obviates the need for a sorter between the frontend and the backend portions of the system. All the right singular vectors will be pipelined into the matrix-matrix multiply array.
• The inner-product processors have to wait until all the elements of the rows of product $A_k V$ have been computed. The elements of each row can then be separated into signal and noise components.

• The signal components are accumulated using a weighted inner product, where the weight for each element is $\frac{\sigma^2}{(\eta - \sigma_k^2)^2}$. The noise components are accumulated using the usual inner product.

• The processors which perform the optimization have a slightly more complicated minimization to perform, since the polynomial is cubic instead of quadratic. Otherwise, the processors operate exactly as in the deterministic-MUSIC architecture.

### 6.4 Estimator for a Multipath Channel

In this section we will extend the architecture of the previous two sections to a multipath channel. The estimator for the multipath channel could be classified as an extension of the deterministic-MUSIC metric, with a parametric least-squares fit to estimate the multipath channel [2].

For this section we define two matrices constructed from shifted versions of the user's spreading code. The matrices, $A^r_k$ and $A^l_k$ are Toeplitz matrices, which are lower triangular and upper triangular respectively. The matrices are defined as:

$$A^r_k = \begin{pmatrix} 0 & a^k_{N-1} & a^k_{N-2} & \cdots & a^k_1 \\ 0 & 0 & a^k_{N-1} & \cdots & a^k_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ a^k_0 & 0 & \cdots & 0 \end{pmatrix}$$

$$A^l_k = \begin{pmatrix} 0 & a^k_1 & a^k_0 & \cdots & 0 \\ a^k_1 & a^k_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a^k_{N-1} & a^k_{N-2} & \cdots & a^k_0 \end{pmatrix}$$
Figure 6.4: Parallel architecture to implement Multipath MUSIC backend for a single user

The channel impulse response is

$$h(t) = \sum_{p} \alpha_p \delta(t - \tau_p),$$

(6.15)

where, $p$ is the number of paths. It is assumed that the largest $\tau_p$ is less than half the symbol period $NT_c$, where $N$ is the spreading gain and $T_c$ is the chip rate [2]. The deterministic MUSIC estimator is obtained by projecting the response of a given user's spreading code onto the null space and minimizing the resulting objective function:

$$\hat{h} = \arg \min_{\|a\|_2 = 1} \left\| \left( S_A^\dagger A_k \right) h \right\|^2_2$$

(6.16)

The optimum occurs when $h$ is the right singular vector corresponding to smallest singular value of $(A_k' S \quad A_k' S')'$. Note that an alternative formulation of the deterministic-
MUSIC metric for the single path case occurs when the channel response, $h$, is restricted to the family of all unit vectors aligned with the axes.

Once the estimate of the channel, $\hat{h}$, has been computed, the strongest $r$ paths may be recovered by a least-squares fit to this estimate. Estimates of these $r$ paths may then be used by an $r$-path RAKE receiver. Each path may be successively recovered by performing the following operations:

$$[\alpha_\nu, \gamma_\nu] = \arg \min_{\alpha, \gamma \in [0, 1]} \left( \hat{h}_{k, \nu} - (1 - \gamma)\alpha \right)^2 + \left( \hat{h}_{k, \nu+1} - \gamma\alpha \right)^2$$

$$\hat{\nu} = \arg \max_{\nu \in \{0, 1, \ldots, N-1\}} |\alpha_\nu|$$

$$\hat{\tau} = \hat{\nu} + \gamma_\nu$$

$$\hat{\alpha} = \alpha_\nu$$

(6.17)

At each iteration the estimated path is subtracted from $\hat{h}$ and the above operations repeated.

An architecture to implement this algorithm is shown in Fig. 6.4. The computation of $(A_k^t S_\perp \quad A_k^t S_\perp)^t$ requires two matrix-matrix multiplications. However, by exploiting the triangular structure of the $A_k^t$ and $A_k^t$ matrices, the total computation can be reduced by half to be equivalent to a single matrix-matrix multiplication. The matrix-multiplication array in Fig. 6.4 is composed of two complementary arrays. Rows of $(A_k^t S_\perp \quad A_k^t S_\perp)^t$ then leave the array from the top.

The inner-product processors shown in the architecture in Fig. 6.1 are replaced by an array of processors to compute the right singular vector corresponding to the smallest singular value. Although a pipelined SVD array can be used to achieve this, it is faster and cheaper to use an algorithm that computes only the smallest singular vector. One possibility which merits further investigation is the class of condition estimators which are designed for precisely this kind of computation [6, 38].

The singular vector is passed to an array of processors which compute the $r$ paths to be passed to the detector. A processor array analogous to those in the previous two sections can be used for this purpose. Hence, the overall architecture is very similar to the architecture for the single-path case with the following exceptions:
• The matrix-multiply array is now composed of two complementary triangular arrays.

• The inner-product array is replaced by an array to compute the smallest singular vector of an orthogonal basis spanning the vectors which leave the matrix-multiply array.

• Since the objective function is different, the topmost array of processors in Fig. 6.4 consist of processors which perform a least-squares fit as described by the sequence of steps in Eq. 6.17.

6.5 Computational complexity of the backend processing

Table 6.2 shows the computational complexity of acquisition at each step for a single user, for the three estimators presented above. The same algorithms may also be used for tracking, albeit with less complexity. Table 6.3 shows the computational reductions gained when the search is restricted to lie close to the estimate computed at the previous step. These numbers show a reduction of order $N$ over the acquisition complexity. The same reduction, however, is not possible for the multipath estimator. For deterministic MUSIC and multipath estimators, if the sorting of null vectors is postponed to after step 1 as discussed in Sec. 6.2, then the complexity of the acquisition will be the same as that for ML-MUSIC.

<table>
<thead>
<tr>
<th>Block SVD</th>
<th>$O(N^2L \log N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelined SVD</td>
<td>$O(N^2L \log L)$</td>
</tr>
<tr>
<td>Pipelined SVD with pre-QR</td>
<td>$O(N^2L \log N)$</td>
</tr>
<tr>
<td>Moonen's SVD updating</td>
<td>$O(N^2L)$</td>
</tr>
<tr>
<td>Stewart's URV or ULV</td>
<td>$O(N^2L)$</td>
</tr>
<tr>
<td>RLS subspace tracking</td>
<td>$O(2NLK)$</td>
</tr>
</tbody>
</table>
Table 6.2: Complexity of acquisition of a single user

<table>
<thead>
<tr>
<th></th>
<th>D-MUSIC</th>
<th>ML-MUSIC</th>
<th>Multi-path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>$O(N^3 - 2N^2K)$</td>
<td>$O(N^3)$</td>
<td>$O(N^3 - 2N^2K)$</td>
</tr>
<tr>
<td>Step 2</td>
<td>$O(N^2 - NK)$</td>
<td>$O(N^2)$</td>
<td>$O(2N^2 - 4NK)$</td>
</tr>
<tr>
<td>Step 3</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(Nr)$</td>
</tr>
</tbody>
</table>

Table 6.3: Complexity of tracking a single user

<table>
<thead>
<tr>
<th></th>
<th>D-MUSIC</th>
<th>ML-MUSIC</th>
<th>Multi-path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>$O(N^2 - 2NK)$</td>
<td>$O(N^2)$</td>
<td>$O(N^2 - 2NK)$</td>
</tr>
<tr>
<td>Step 2</td>
<td>$O(N - K)$</td>
<td>$O(N)$</td>
<td>$O(2N^2 - 4NK)$</td>
</tr>
<tr>
<td>Step 3</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(r)$</td>
</tr>
</tbody>
</table>

The complexity of the frontend computation for each step of the backend is shown in Table 6.1. For the recursive formulations, $L$ denotes the number of observations that are received for each update of the delay estimates. The RLS subspace tracking is assumed to be used to track the $2K$ signal vectors which are then used in the backend estimators.

Note that the computation required to acquire a single user is of the same order as the frontend computation. Hence, the computation required when $K$ users are acquired is an order of magnitude larger than the computation in the frontend. However, when the system is in the tracking mode all $K$ users are being tracked, the two parts of the system require similar amount of computation \(^1\). In the multipath case, since the computation does not go down during tracking, the backend computation is always an order of magnitude greater than the frontend computation. This disparity in the computation requirements will play an important role when these algorithms are extended for use in a fading channel.

\(^1\)We are ignoring the overhead of dynamic scheduling required in the backend.
6.6 Summary

The deterministic MUSIC estimator can be implemented using an architecture which consists of the following subarrays:

- Project $A_k$ onto the noise subspace basis vectors, $S_\perp$.

- Compute the inner products of the columns of the resulting matrix.

- Use the inner products to form the coefficients of quadratic polynomials to be minimized within each chip interval. A different polynomial needs to be optimized within each chip interval.

- Select the chip and the estimated delay which minimizes the above objective function.

The ML-MUSIC estimator is similar. It is implemented on an array which performs the following substeps:

- Project $A_k^r$ and $A_k^n$ onto the singular vectors (both signal and noise space vectors).

- Separate the elements of each column of the resulting matrix into the signal and noise components. Form the inner products of all the noise components in each column. Similarly form a weighted inner product of the signal components.

- Use these inner products to compute the coefficients of cubic polynomials within each chip interval and minimize them.

- Select the time delay which results in the smallest value for the polynomial in its chip interval.

The multipath MUSIC estimator is implemented using the following steps:

- Project $A_k^r$ and $A_k^n$ onto the singular vectors (both signal and noise space vectors).

- Use a condition estimator to compute the smallest singular vector from the singular basis which spans the columns of a resulting matrix.
- Successively extract the strongest paths using a least squares fit to the singular vector within each chip interval.
Chapter 7

Summary and Future Work

There are three primary contributions of this thesis. The first contribution is a pair of algorithms to add to the repertoire of parallel algorithms for subspace-based signal processing. The new algorithms developed in this thesis are a pipelined SVD algorithm and a mixed fixed-point/floating-point algorithm to implement sliding window SVD updating. The second contribution of this thesis is in pointing out the vast number of parallel and pipelined algorithms which are applicable to the CDMA acquisition problem. This thesis makes a first effort to compare and contrast a wide variety of algorithms in a qualitative manner. The third contribution of this thesis is in presenting parallel algorithms for the backend portions of the CDMA acquisition.

The pipelined SVD algorithm is unique among SVD algorithms in that it allows a truly systolic system in which all aspects of the system are pipelined. None of the other algorithms allow such a simple aesthetically pleasing system. This thesis showed that such a system can provide asymptotic gains for any given technology related communication and computation parameters. This efficiency is really a result of the degree of pipelining achieved by this array. As with any pipelined system, the efficiency of this array is contingent on keeping the pipeline full. From that point of view, the pipelined array and the subspace-based algorithms form a perfect match, since this is exactly the type of system in which pipelined architectures excel. Although the degree of parallelism available in the pipelined SVD algorithm is only linear as opposed to the $O(n^2)$ parallelism available from other SVD algorithms, the degree of parallelism offered by this array is more than sufficient for the foreseeable future in many applications.
The hybrid algorithm for sliding window SVD extends a notion that fixed-point is often adequate in signal processing systems. Even if floating-point systems are as cheap as fixed-point systems, secondary considerations such as power consumption can tilt the balance in favor of fixed-point computation. Floating-point systems due to their inherent complexity are not desirable in cost-sensitive signal processing applications. Although the algorithm described in this thesis is not truly fixed-point, it does keep the floating-point computation to the bare minimum. The key here is the notion of relational stability, which ensures that the result of a large number of computations may be accurate even if some intermediate quantities are inaccurate. This thesis showed that this notion is not inherently limited to floating-point and similar, albeit weaker, results extend to fixed-point computation. Once it is shown that hyperbolic rotations are not inherently inferior to other downdating methods, we showed that a parallel architecture similar to the one for exponential downdating can efficiently pipeline the computation. However, it should be realized that relational stability is only one piece of the puzzle. There still remains the fact that intermediate results can be inaccurate. In a system such as the CDMA system, such inaccuracies need not be catastrophic, though they will certainly degrade the performance. The decision of whether to use the sliding window SVD would depend on other considerations in addition to numerical stability. If other considerations make the use of a sliding window system desirable, it should be possible to implement such systems with the appropriate exception handling built in. Also, relational stability does not guarantee that the errors will not slowly build up over time, thus requiring that the system be reset periodically. Such exceptions increase the complexity of a system based on sliding window SVD making them less desirable unless other considerations are overwhelming.

The fact that the vast body of research in array signal processing algorithms is applicable through a paradigm shift, to the CDMA acquisition problem, was suggested by the thesis by Bensley. This thesis serves as a bridge between the two areas and thus paves the way towards efficient implementation of near-far resistant CDMA systems. Although,
this thesis does not deal with any particular CDMA system, it does point out the subtle
differences between the different algorithms which are not apparent at first glance. Small
differences in the system specifications can completely change the tradeoffs required in the
implementation of a system. As a consequence, this thesis deliberately does not commit
to any particular algorithm. It is opined that unless the practical system constraints are
completely defined, the exact design process which involves making the appropriate trade-
offs need not be attempted. This thesis attempted to broadly classify the algorithms and
in addition identified a distinct division which enables almost independent development of
algorithms, through the notion of frontend and backend computation. All the previously
developed parallel algorithms are applicable primarily in the frontend.

The third contribution of this thesis is to develop parallel algorithms for the backend
computation. Because of the absence of the rich structure available in the frontend and
the application dependent nature of this computation, not much work has previously been
attempted in implementing backend algorithms. In this thesis a compound architecture has
been identified which appears to unify several of the possible options applicable in the back-
end. This compound architecture is obtained by combining a matrix vector multiplication
array, an inner product array and an array to implement simple second-order or third-order
polynomial optimization.

7.1 Future Work

7.1.1 Delay-locked Loop for acquisition

A quick analysis of the subspace-based algorithms described in this thesis reveals that prac-
tical systems based on these ideas are still impractical even accounting for the phenomenal
advances expected in VLSI technology. The algorithms still need a lot of refining to reduce
complexity.

A powerful idea which reduces complexity by an order of magnitude though requiring
greater care in implementation is the idea of stochastic optimization. The idea is very simple.
Instead of trying to be perfect and perform the optimization in one step, one spreads out the optimization over time. In other words, the optimization function is not a constant over the entire optimization. Instead the "optimization" is expected to succeed only in a stochastic sense. We see this idea in play in a number of algorithms such as delay-locked loops, LMS algorithm for adaptive filters, the SVD updating algorithm described in this thesis among others.

A very good example which illustrates the gains one may obtain from the application of this principle is the LMS algorithm for adaptive filters which reduces complexity from $O(n^2)$ for the RLS method to $O(n)$. Such a phenomenal reduction in complexity is what enables implementations such as echo cancellers which would otherwise be relegated to theory. In addition, real-life is rarely perfect and as a result sophisticated analyses break down. For instance, in real speech, a careful implementation of the LMS algorithm can perform as well as a system based on the RLS algorithm for a fraction of the cost, obviating the need for the latter in the echo canceller applications.

Another place the idea of stochastic optimization yields similar efficiencies is the delay-locked loop which is used for acquisition in conventional CDMA systems [64, 70]. An extension of the PLL idea to near-far resistant CDMA should produce equally impressive reductions. The key components to such an algorithm appear to be the input sequence to train the DLL and a strategy to prevent the DLL from locking onto local minima. The resulting algorithm will most likely not be amenable to a detailed analysis, but would still be a significant step towards enabling practical implementations.

### 7.1.2 Relational stability of CORDIC-based hyperbolic downdating

The downdating algorithm described in Chapter 4 takes a step in reducing complexity of implementations by demonstrating the relational stability of a hybrid algorithm where except for the computation of the rotation, fixed-point arithmetic is used throughout. It is well-known that CORDIC can be used to compute and apply hyperbolic rotations. It would
be interesting to investigate the relational stability properties of CORDIC in the hyperbolic mode or a variation of the same. One obvious idea to investigate is if the the basic CORDIC iterations can be re-ordered as done by Chambers to stabilize the algorithm. A related idea to pursue is whether it is possible to obtain relational stability from a purely fixed-point implementation.

This thesis took a first step in the direction of implementation of near-far resistant CDMA systems. The implementation problems inherent in these systems are far from being solved. Significant reductions in complexity are required to enable a realistic system. It is hoped that this thesis can serve as a starting point for someone embarking on this task.
Bibliography


