

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

A HIGH PERFORMANCE PARALLEL ARCHITECTURE FOR ADAPTIVE BEAMFORMING

by

HAK-LIM KO

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

Raleigh

1995

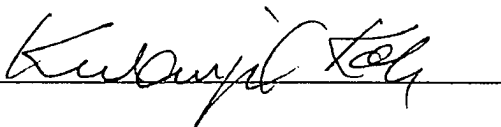
APPROVED BY:




Sarah Kapala



Thomas K. Mura



K. S. Lee



Chair of Advisory Committee

UMI Number: 9604819

UMI Microform 9604819

Copyright 1995, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized
copying under Title 17, United States Code.

UMI

300 North Zeeb Road
Ann Arbor, MI 48103

Abstract

KO, HAK-LIM. A High Performance Parallel Architecture for Adaptive Beamforming (Under the direction of Dr. Winser E. Alexander).

The objective of this dissertation is to implement a special purpose architecture for the enhanced direction-of-arrival system (adaptive beamforming). This research consists of two parts: the development of a new method for adaptive beamforming and the implementation of the new method on a high performance parallel architecture.

First, we propose a modified eigenvector method (MEVM) to enhance the resolution for direction-of-arrival (DOA) estimation. MEVM uses the weighted forward-backward covariance matrix to improve the estimation of the covariance matrix. This dissertation analyzes the effect of using the weighted forward-backward covariance matrix on the performance of the eigenvector method (EVM). We compare the conventional covariance matrix method (EVM) and the weighted forward-backward covariance matrix method (MEVM). The simulation results show that MEVM is more accurate and has better resolution than the conventional EVM under the same noise conditions.

Second, we have conceptually designed a high performance architecture for the implementation of MEVM. In the architecture, several modules cooperate in a pipelined manner. However, one of the modules is very computationally intensive because the eigenvalues and the corresponding eigenvectors of the forward-backward covariance

matrix have to be computed. This may be the bottleneck of the system. We investigate the use of the Block Data Parallel Architecture (BDPA) to avoid this bottleneck.

The BDPA is a parallel architecture that achieves high system throughput and high system efficiency. The architecture uses a globally asynchronous and locally synchronous clock distribution scheme to avoid the clock skew problem which may occur in a large synchronous system. The control of the interconnection network is simple and easy to implement because the data transfers between processors are local and uni-directional. The architecture is flexible in the sense that the same number of processors can be used to solve a complex set of problems whose input matrix size may vary.

We have simulated the architecture with a programmable simulator using the timing characteristics of the TMS320C40 digital signal processor. The simulation results consistently show that the parallel architecture can provide high performance, high efficiency, and almost linear speedup for adaptive beamforming.

BIOGRAPHY

Hak-Lim Ko was born in Korea, on Oct. 30 1960. He received his B.S. degree in Electronics from Soong Sil University, Seoul, Korea in 1983 and his M.S. degree in Electrical Engineering from Fairleigh Dickinson University, Teaneck, NJ, in 1986.

From 1983 to 1984, he was a lecturer of the DaiHeon Junior College, Incheon, Korea. During 1987-1991, he was a communication officer in the Republic of Korea Air Force.

He was married to Jin-Kyeong Kim in July, 1984. They have a son, Jeong-Gil, and a daughter, Anna.

His main research interests are in multidimensional signal processing, and the development of algorithms and special purpose architectures for digital signal processing.

ACKNOWLEDGEMENTS

I am greatly indebted to my advisor, Professor Winser E. Alexander for his guidance and support throughout the course of this work. I have benefited much from his keen insights and critical suggestions. I would also like to thank the other members of my committee, Dr. Sarah A. Rajala, Dr. Tom K. Miller, and Dr. Kwangil Koh who have most generously offered their advice and help.

I would like to express gratitude to my colleagues Vincent Wilburn and Sonetra Howard for their invaluable discussion and assistance.

Finally, I would like to express my deepest appreciation to my parents, my parents-in-law, my wife, Jin-Kyeong, my son, Jeong-Gil, and my daughter, Anna for their love, support, and sacrifice throughout the course of this work.

TABLE OF CONTENTS

List of Tables	vi
List of Figures	viii
List of Symbols	xi
List of Abbreviations	xiii
1 Introduction	1
2 A High Resolution Adaptive Beamforming Algorithm	7
2.1 Problem Formulation	7
2.2 Comparison of Spectral Estimates	10
2.3 The Modified Eigenvector Method (MEVM)	19
2.4 Implementing the MEVM	32
3 The Partial Eigenvalue Solution Algorithm	35
4 Block Data Parallel Architecture	42
4.1 The Advantages of the BDPA	42
4.2 BDPA Configuration	47
4.2.1 Input Module	47



4.2.2	Processor Array	49
4.2.3	Output Module	52
5	A High Performance Parallel Architecture	54
5.1	The System Building Block	54
5.2	Iteration Module	55
5.3	QR Decomposition Module1	60
5.4	Back-Substitution Module	76
5.5	QR Decomposition Module2	82
5.6	Matrix-Matrix Multiplication Module	85
5.7	$K \times K$ Eigenvalue Solution Module	90
5.8	The Real-Time Rate for the Overall System	91
6	Summary and Future Research	94
6.1	Summary	94
6.2	Future Research	95
7	Bibliography	97

List of Tables

2.1	A comparison of weighting functions	13
2.2	The comparison of the noise-subspace perturbation angle	23
3.1	QR-inverse iteration method	37
3.2	Simultaneous iteration method	38
3.3	Subspace iteration with Rayleigh-Ritz method	38
3.4	Convergence comparison	39
3.5	Partial eigenvalue solution algorithm	40
4.1	The comparison of a systolic array, a wave-front array, and a BDPA .	47
5.1	Number of iterations required to compute the four largest eigenvalues and the corresponding eigenvectors from 64×64 matrices	59
5.2	The number of clock cycles required for a floating-point operation, based on the TMS320C40	69
5.3	The rate of computational clock cycles to data movements	70
5.4	Simulation result for QRM ₁ with 2 PEs	71
5.5	Simulation result for QRM ₁ with 4 PEs	71
5.6	Simulation result for QRM ₁ with 6 PEs	71
5.7	Simulation result for QRM ₁ with 8 PEs	72
5.8	Simulation result for QRM ₁ with 10 PEs	72



5.9	Simulation result for QRM ₁ with 16 PEs	72
5.10	Simulation result for QRM ₁ with 22 PEs	73
5.11	Simulation results for QRM ₁ with different number of processing elements	74
5.12	The utilization comparison for different matrix sizes. The number of eigenvalues to be solved is assumed to be 4. Four PEs are used.	75
5.13	The utilization comparison with varying of the number of eigenvalues to be solved. E is a 64×64 matrix and B and D are the $64 \times K$ matrices where K is the number of eigenvalues to be solved. Four PEs are used.	76
5.14	Simulation result for BSM with 2 PEs	79
5.15	Simulation results for BSM with different number of processing elements	80
5.16	The utilization comparison for different matrix sizes. The number of eigenvalues to be solved is assumed to be 4. Four PEs are used.	81
5.17	The utilization comparison with varying of the number of eigenvalues to be solved. R is a 64×64 matrix and B and U are the $64 \times K$ matrices where K is a number of eigenvalues to be solved. Four PEs are used.	81
5.18	Simulation results for QRM ₂	85
5.19	Simulation result for MM with 2 PEs	87
5.20	Simulation results for MM with different number of processing elements	89
5.21	Simulation result for ESM with 8 PEs	92
5.22	Real-time rate to extract the 4 smallest eigenvalues and the corresponding eigenvectors from a 64×64 matrix	92



List of Figures

2.1	Performance comparison of algorithms	15
2.2	Comparison of weights on the eigenvectors, (a) the number of signals is assumed to be known, (b) the number of signals is unknown.	16
2.3	Performance comparison of algorithms, SNR = -10 dB	19
2.4	The relation between a noise-free eigenvector and a noisy eigenvector	20
2.5	The perturbation angle between the noise-free subspace and the noisy subspace when w varies. The smallest perturbation angle is obtained at $w = \frac{1}{2}$	22
2.6	Comparison of the noise-subspace perturbation angle with SNR equal to 5 dB.	24
2.7	Comparison of the noise-subspace perturbation angle with SNR equal to 0 dB.	24
2.8	Comparison of the noise-subspace perturbation angle with SNR equal to - 5 dB.	25
2.9	Comparison of the noise-subspace perturbation angle with SNR equal to - 10 dB.	25
2.10	Comparison of the spectral estimates with SNR equal to 0 dB. The two sources are resolved well.	27
2.11	Comparison of the spectral estimates with SNR equal to - 5 dB. The two sources are not resolved as the SNR decreases to - 5 dB.	27
2.12	Comparison of the spectral estimates with two sources are at -2° and 2°	29

2.13	Comparison of the spectral estimates with two sources are at -2° and 1.5°	29
2.14	The estimated spectra for the conventional EVM. Two sources are not resolved.	31
2.15	The estimated spectra for MEVM. Two sources are resolved well. . .	31
2.16	Conceptual Design of the Overall System	33
4.1	Block Diagram for the BDPA	48
4.2	Input Module with 2 FIFO buffers	49
4.3	Block Diagram for the Processor Array	50
4.4	Output Module	52
5.1	Pipelined subspace iteration algorithm flow	55
5.2	Pipelined submodules to solve the partial eigenvalue solution algorithm	56
5.3	Register Module	57
5.4	Block Diagram for the Register Module	58
5.5	Iteration modules for the partial eigenvalue solution algorithm	58
5.6	QR decomposition Module1	65
5.7	A processing element in QRM_1	65
5.8	The speedup of QRM_1	75
5.9	Back-Substitution Module	77
5.10	A processing element in BSM	78
5.11	The speedup of BSM	80
5.12	QR decomposition Module2	83
5.13	A processing element in QRM_2	83
5.14	Matrix-matrix Multiplication Module	88

5.15 The speedup of MM	89
5.16 Eigenvalue Solution Module	90

List of Symbols

t	time index
i	general index
j	general index
θ	direction-of-arrival or source direction
K	number of input signals
M	number of array elements
N	number of snapshots
σ^2	noise power
κ	wavelength
λ_i	an eigenvalue
\mathbf{v}_i	an eigenvector, $M \times 1$
w	weighting coefficient
\mathbf{w}	weight vector, $M \times 1$
$s_i(t)$	input signal
$\mathbf{s}(t)$	input signal vector, $K \times 1$
$x_i(t)$	received or observed data at the i th sensor
$\mathbf{x}(t)$	received or observed data vector, $M \times 1$
$\mathbf{n}(t)$	a noise vector, $M \times 1$
$\mathbf{a}(\theta_i)$	a steering vector, $M \times 1$
A	direction matrix, $M \times K$
R_n	noise only covariance matrix

P_s	signal power
X	data matrix, $M \times N$
R	data covariance matrix, $M \times M$
C	forward-backward covariance matrix, $M \times M$
I	identity matrix
J	inversely diagonal matrix
$P(\theta)$	estimating spectrum
E	$M \times M$ matrix
D	initial trial matrix, $M \times K$
H	intermediate matrix to find the partial eigenvalue solution of E , $K \times K$
Λ_s	diagonal matrix where the diagonal elements being the eigenvalues of H , $K \times K$
Y_s	eigenvector matrix of H , $K \times K$
Q_1	orthogonal transformation of E ($E = Q_1 R_1$), $M \times M$
R_1	upper triangular matrix of E , $M \times M$
c_{ij}, s_{ij}	multiplication pair to reduce the matrix E to an upper triangular matrix R_1
U	intermediate matrix to find the partial eigenvalue solution of E , $M \times K$
Q_2	orthogonal transformation of U ($U = Q_2 R_2$), $M \times K$
R_2	upper triangular matrix of U , $K \times K$
B	updated matrix of S by Q_1 ($B = Q_1^T S$), $M \times K$
I_i	i th block of input data
O_i	i th block of output data
In_Bus	input data bus
Out_Bus	output data bus

List of Abbreviations

BDPA	block data parallel architecture
BSM	back-substitution module
DeMUX	demultiplexer
DOA	direction-of-arrival
DSBM	data sampling and buffering module
DSP	digital signal processing
EDM	eigen-decomposition module
ESM	eigenvalue solution module
EVM	eigenvector method
FBCM	forward-backward covariance matrix module
FIFO	first input first output
IM	input module
I/O	input/output
ITM	iteration module
MM	matrix-matrix multiplication module
MUSIC	multiple signal characterization
MUX	multiplexer
MVE	minimum variance estimator
OM	output module
PA	processing array
PE	processing element

QRM	QR decomposition module
RM	register module
SEM	spectral estimation module
SHIRE	stable high resolution estimator
SIMD	single instruction multiple data
SM	submodule
SNLM	stable nonlinear method
SNR	signal-to-noise ratio

1

Introduction

High resolution direction-of-arrival (DOA) estimation is widely used for many multi-sensor systems for applications such as sonar, radar, and seismic exploration. Many high resolution algorithms based on the eigen-decomposition analysis of the observed covariance matrix, such as MVE [24], MUSIC [50], EVM [25], SNLM [6], and SHIRE [32] have been proposed for solving this problem. These algorithms emphasize specific eigenvectors of the observed covariance matrix to obtain the best spectral estimate which provides the DOA information. However, the resolution of the spectral estimates for these algorithms is severely degraded when the signal-to-noise ratio (SNR) is low and/or the arrival angles are close to each other. This is because the observed sample covariance matrix is perturbed by correlated noise.

The quality of the estimation of the covariance matrix is very important to get the best performance for these algorithms. Since the observed sample covariance matrix is perturbed by correlated noise, the performance estimate of the spectrum using this conventional covariance matrix is degraded when the signal-to-noise ratio (SNR) is low and/or the arrival angles are close to each other. Therefore, Raghunath [45] and Rao [47] used spatial smoothing to enhance the covariance matrix. Moghaddamjoo

[39] used spatial filtering and Du and Kirilin [11] used temporal correlations between multiple snapshots to estimate the covariance matrix. Basically, these methods use a similar idea. They extract more information from the given data set.

We introduce a modified eigenvector method (MEVM). MEVM uses the weighted forward-backward covariance matrix to improve the estimation of the covariance matrix. Likewise, this method extracts more information from the given data set. Furthermore, MEVM reduces the phase error of the estimated DOAs.

In this thesis, we first compare and analyze the typical high resolution algorithms noted above. Our results show that the EVM [25] provides the best performance when we consider the signal-to-noise ratio (SNR) of the input data and whether or not we know the number of sources. Then, we evaluate the effect of using the forward-backward covariance matrix on the performance of the EVM under various conditions. Our simulation results show the performance improvement for the DOA estimation obtained using the MEVM.

Computing eigenvalues and the corresponding eigenvectors is a necessary part of the high performance adaptive beamforming algorithms. These computations are also used for the recent high-resolution algorithms for many digital signal processing applications, such as array signal processing [35], system identification [8] [38], image processing [44], spectrum estimation [14], and filter design [36]. These signal processing algorithms require extracting the partial eigenvalues and the corresponding eigenvectors from a large space. These methods usually employ an eigenvalue decomposition. However, the eigenvalue decomposition method requires $O(M^3)$ floating point operations (flops) for an $M \times M$ matrix. Although these algorithms that use eigenvalue

decomposition can usually achieve much better performance than traditional least squares methods, their heavy computational load often makes them difficult to implement in real-time. Since these algorithms require computation of only a few of the eigenvalues and the corresponding eigenvectors from a large space, it may not be necessary to compute all of the eigenvalues and the corresponding eigenvectors. Several methods exist for the partial solution of the eigenvalue problem. QR-inverse iteration method [3], simultaneous iteration method [23] [49], and subspace iteration method [42] [46] are frequently used. We compared these partial eigenvalue solution algorithms based on the computational requirements and speed of convergence. Using the simulation results as a means for our comparison, we propose the best algorithm and implement that on a parallel architecture.

Systolic array architectures [5] [37] [43] [48] are deeply pipelined which provide a high degree of parallelism. However, these architectures use a fixed-number of processing elements (PEs) for a fixed-sized matrix. For example, the QR decomposition of a 16×16 matrix requires a 16×16 array of PEs for systolic array implementation, while the QR decomposition of a 30×30 matrix requires a 30×30 array of PEs. Therefore, the systolic array architecture is suitable for computing the full eigenvalues and the corresponding eigenvectors of the 16×16 or the 30×30 system that it was designed for.

However, in many digital signal processing applications, the number of eigenvalues required for the partial solution as well as the size of the input matrix may vary. These properties make the use of the systolic array inadequate. For example, in adaptive beamforming, the number of the partial eigenvalues involved varies with the

number of signals present. The input matrix size can vary depending on the particular applications.

In our parallel architecture, the number of PEs does not depend on the size of the input matrix and/or the number of eigenvalues required for the partial solution. For example, it can be used to solve for the eigenvalues for a 64×64 matrix and also a 75×75 matrix using the same number of PEs (say 8) with high efficiency and high throughput. Also, it can extract any number of the smallest (largest) eigenvalues and the corresponding eigenvectors without having to change the number of PEs in the processing array.

In the systolic array implementation, the PE receives input data by element, computes a small number of floating-point operations, and sends out the results by element. Therefore, the total number of operations in a PE may not be much larger than the total number of input and output operations. Sometimes, it may be an I/O-bound system [26] [29] (*i.e.*, the total number of operations is smaller than the total number of input and output operations). One of the important aspects of utilizing a high performance architecture effectively is to increase the ratio of floating-point operations to data movements to balance the computations and the data movements.

Dongarra and Duff [10] show that currently the processing speed for modern computer systems is increasing at a much higher rate than the speed of the data bus. Therefore, it is important to improve the ratio of floating-point operations to data movements to fully utilize the hardware. In our parallel architecture, we improve the ratio by processing the data by block (multiple data elements). By partitioning the input data into blocks, each PE processes a block of data, while reusing the data inter-

nally as appropriate. The reuse of data reduces the total number of data movements, therefore increasing the ratio of floating-point operations to data movements.

Data movements and interprocessor communications play an important role in parallel architecture design [33]. The systolic implementation requires complicated communication connections between PEs. However, in our architecture, the data transfer between PEs is local and uni-directional. This makes the control of the interconnection architecture simple and easy to implement.

We have simulated the architecture with the Erg programmable simulator [2] using the timing characteristics of the TMS320C40 digital signal processor [21]. The simulation results consistently show that our parallel architecture provides high performance, high efficiency, and almost linear speedup for many applications. Along with these desirable results, the architecture is flexible and can easily be changed to meet the demands of many different applications. The contents of the remaining chapters of this dissertation are briefly described below.

Chapter 2 describes the enhanced DOA system. We first compare and analyze the typical high resolution algorithms for DOA. Then, we propose a modified eigenvector method (MEVM) in order to enhance the resolution of the system using the forward-backward covariance matrix.

Chapter 3 shows the comparison of the typical partial eigenvalue solution algorithms. We compared these partial eigenvalue solution algorithms based on the computational requirements and speed of convergence. Using the simulation results for comparison, we propose the best algorithm to implement on the block data parallel architecture (BDPA).

Chapter 4 presents the architectural features of the block data parallel architecture (BDPA) and explains how these features make contributions to the implementation for the digital signal processing applications.

Chapter 5 shows the implementation of the partial eigenvalue solution algorithm on the BDPA. In this chapter, the modules used to compute the partial solution of the eigenvalues and the corresponding eigenvectors are discussed, and the simulation results are given.

Finally, chapter 6 gives a summary of the results. It also includes recommendations for future research.

2

A High Resolution Adaptive Beamforming Algorithm

2.1 Problem Formulation

The problem of finding the direction-of-arrival (DOA) of multiple plain waves using sensor arrays, and the related problem of estimating the parameters of multiple superimposed sinusoidal signals from noisy measurements have recently received significant attention in the signal processing area. The delay-and-sum beamformer was originally used to solve this problem. However, the conventional delay-and-sum beamformer achieves relatively poor estimates for the direction-of-arrival because of its wide bandwidth. Recently many algorithms, such as MVE [24], MUSIC [50], EVM [25], SNLM [6], and SHIRE [32] have been proposed for improving the resolution of an array beyond that which can be obtained with conventional beamforming.

In the high-resolution algorithms, we assume that K narrow-band signals with unknown constant amplitudes and phases, and additive noise impinge on an array consisting of M sensors. The transmission medium is assumed to be isotropic and nondispersive so that the radiation propagates in straight lines. The sources are assumed to be in the far-field of the array. We assume that the K signals $s_1(t), s_2(t), \dots, s_K(t)$

are zero-mean, stationary, and ergodic complex Gaussian random processes. Let

$$\mathbf{x}^T(t) = [x_1(t), x_2(t), \dots, x_M(t)] \quad (2.1)$$

be the observation vector at the array of sensors (the superscript T denotes transpose) and

$$\mathbf{s}^T(t) = [s_1(t), s_2(t), \dots, s_K(t)] \quad (2.2)$$

be the vector of source signals. Then the sampled wavefronts received at the M array of sensors are linear combinations of the K incident wavefronts and noise.

$$\mathbf{x}(t) = A\mathbf{s}(t) + \mathbf{n}(t) \quad (2.3)$$

where $\mathbf{n}(t)$ is the additive noise vector. $\mathbf{n}(t)$ is assumed to be a zero-mean, stationary, and complex ergodic white Gaussian random process. A is the M by K direction matrix

$$A(\theta) = [\mathbf{a}(\theta_1), \mathbf{a}(\theta_2), \dots, \mathbf{a}(\theta_K)] \quad (2.4)$$

where $\mathbf{a}(\theta_1), \mathbf{a}(\theta_2), \dots, \mathbf{a}(\theta_K)$ are the steering vectors [50]. The steering vector $\mathbf{a}(\theta_i)$ is completely determined by the sensor directivity patterns and the array geometry. In this paper, for simplicity, discussions deal with single-dimensional parameter space, e.g. azimuth-only direction finding of far-field point sources, since the basic concepts are most easily visualized in such spaces. For example, if we assume narrow-band signals, far field sources and a uniform linear array with $\kappa/2$ element spacing, the steering vectors have the following form

$$\begin{bmatrix} 1 \\ \exp^{-i\pi \sin \theta} \\ \exp^{-i2\pi \sin \theta} \\ \vdots \\ \exp^{-i(M-1)\pi \sin \theta} \end{bmatrix} \quad (2.5)$$

A class of algorithms for estimating θ in equation (2.1) is based on the eigen-decomposition of the sample covariance matrix of $\mathbf{x}(t)$. Therefore, $N (> M)$ snapshots are collected in a large data matrix

$$X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \quad (2.6)$$

When the elements of the noise vector are assumed to be identically distributed and uncorrelated both with each other and the signal, the covariance matrix X is defined by

$$\begin{aligned} R &= E\{XX^*\} \\ &= \frac{1}{N}XX^* \\ &= A(\theta)P_sA^*(\theta) + \sigma^2R_n \end{aligned} \quad (2.7)$$

where P_s and σ^2 denote the signal and noise power, respectively, and $*$ denotes complex-conjugate transpose.

When the noise is uncorrelated, the noise only covariance matrix R_n becomes the $M \times M$ identity matrix I . This Hermitian matrix is then decomposed via a spectral decomposition into

$$R = \sum_{i=1}^M \lambda_i \mathbf{v}_i \mathbf{v}_i^* \quad (2.8)$$

where $\lambda_1 \geq \lambda_2 \geq \dots \lambda_K > \lambda_{K+1} = \dots = \lambda_M = \sigma_n^2$ are the real eigenvalues of R and $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_M$ are the corresponding orthonormal eigenvectors. Also, K eigenvectors ($\mathbf{v}_i \ i \leq K$) related to the K maximum eigenvalues span the K dimensional signal subspace. The other $(M - K)$ eigenvectors $\{\mathbf{v}_i \ K+1 \leq i \leq M\}$ related to the $(M - K)$ minimum eigenvalues span a $(M - K)$ dimensional noise subspace and are orthogonal to

each of the K signal eigenvectors. Likewise, most of the eigen-decomposition methods decompose the observed covariance matrix into two orthogonal spaces (the signal and noise subspaces) and estimate the DOA's from one of these spaces. The different algorithms are now examined.

2.2 Comparison of Spectral Estimates

One of the important high resolution algorithms is the minimum variance estimator (MVE) [18] [24] [56]. The idea of this algorithm is to minimize the total output power subject to a constraint of unity undistorted signal response from the desired look direction. This algorithm may be stated as follows:

$$\begin{aligned} & \underset{\mathbf{w}}{\text{minimize}} \quad \mathbf{w}^* \mathbf{R} \mathbf{w} \\ & \text{subject to} \quad \mathbf{w}^* \mathbf{a}(\theta) = 1 \end{aligned} \quad (2.9)$$

where \mathbf{w} is the complex weight vector for the adaptive beamformer and $\mathbf{a}(\theta)$ is the steering vector. The spatial output spectrum for this algorithm is given by

$$\begin{aligned} P(\theta)_{\text{MVE}} &= [\mathbf{a}(\theta)^* \mathbf{R}^{-1} \mathbf{a}(\theta)]^{-1} \\ &= \left[\sum_{i=1}^M \frac{1}{\lambda_i} |\mathbf{a}(\theta) \mathbf{v}_i|^2 \right]^{-1} \end{aligned} \quad (2.10)$$

It can be seen from equation (2.10) that the MVE weights the eigenvectors \mathbf{v}_i by $\frac{1}{\lambda_i}$ in both the signal and the noise subspaces.

The multiple signal characterization (MUSIC) algorithm was the first one to show the benefits of using a subspace based approach. The MUSIC algorithm computes a spatial spectrum from the noise subspace, and determines the DOA's from the dominant peaks in the spectrum. Let's assume that $\hat{\theta}$ is the direction-of-arrival.

When $\theta = \hat{\theta}$, $\mathbf{a}(\theta)$ is in the signal subspace. Since the signal subspace is orthogonal to the noise subspace (spanned by the eigenvectors \mathbf{v}_i , $K + 1 \leq i \leq M$), when $\theta = \hat{\theta}$

$$\sum_{i=K+1}^M |\mathbf{a}^*(\hat{\theta})\mathbf{v}_i|^2 = 0 \quad (2.11)$$

Therefore, if the estimating spectrum is formed as the inverse of equation (2.11), there will be a peak corresponding to that direction.

The spectrum estimation by MUSIC [50], [51] is given by

$$P(\theta)_{\text{MUSIC}} = \left[\sum_{i=K+1}^M |\mathbf{a}^*(\theta)\mathbf{v}_i|^2 \right]^{-1} \quad (2.12)$$

As a result, MUSIC weights the eigenvectors corresponding to the noise eigenvalues by unity while weighting the eigenvectors corresponding to the signal eigenvalues by 0 to deemphasize the eigenvectors corresponding to the signal eigenvalues.

The eigenvector method (EVM) [25] is similar to the MUSIC algorithm except that the EVM emphasizes the eigenvectors corresponding to the noise eigenvalues by weighting them by $\frac{1}{\lambda_i}$ to combine the advantages from the MVE and MUSIC.

The estimated spectrum for EVM is

$$\begin{aligned} P(\theta)_{\text{EVM}} &= \left[\mathbf{a}(\theta)^* R_n^{-1} \mathbf{a}(\theta) \right]^{-1} \\ &= \left[\sum_{i=K+1}^M \frac{1}{\lambda_i} |\mathbf{a}^*(\theta)\mathbf{v}_i|^2 \right]^{-1} \end{aligned} \quad (2.13)$$

where R_n^{-1} is an estimate of the noise-only covariance matrix. Since the eigenvalues have been sorted in decreasing order, the EVM weights the noise eigenvectors in increasing order.

In the presence of spherical isotropic noise and system phase errors, the signal eigenvectors are stable but can not be used to provide high resolution whereas the

middle eigenvectors (corresponding to eigenvalues just below the signal) are stable with potentially high resolution (less sensitive to the noise, i.e. these eigenvectors have less perturbed DOA information than the remaining eigenvectors) [6]. The noise eigenvectors, for i near M , are unstable and sensitive to environmental perturbations. Using this idea the stable nonlinear method (SNLM) [6] was introduced. The SNLM uses different weights to emphasize the middle eigenvectors, while deemphasizing the eigenvectors for $i = 1, \dots, K$ (eigenvectors in the signal subspace) and the eigenvectors for i near M (unstable eigenvectors in the noise subspace). The spectrum for SNLM is given by

$$\begin{aligned} P(\theta)_{\text{SNLM}} &= \left[\mathbf{a}^*(\theta) R^{-1/2} (R + \alpha^2 R^{-1})^{-k} R^{-1/2} \mathbf{a}(\theta) \right]^{-1} \\ &= \left[\sum_{i=1}^M \frac{\lambda_i^{k-1}}{(\lambda_i^2 + \alpha^2)^k} | \mathbf{a}^*(\theta) \mathbf{v}_i |^2 \right]^{-1} \end{aligned} \quad (2.14)$$

Here k and α are parameters to be chosen by the user. The α has to be chosen to be greater than the magnitude of the eigenvalues of the unstable eigenvectors, but not greater than the magnitude of the smallest signal eigenvalue (λ_K). Therefore, the SNLM requires prior information on the distribution of the noise eigenvalues. By increasing the value of k we increase the relative emphasis placed on the eigenvectors with λ_i near α (the stable noise eigenvectors). Note that $k = 0$ gives the MVE, so the SNLM is a generalization of the MVE.

The stable high resolution estimator (SHIRE) [32] adds the $\mathbf{w}^* \bar{R} \mathbf{w}$ term to preserve the unperturbed signal subspace (which is essential for high resolution processing) while constraining the sidelobes. The spectrum for SHIRE is

$$P(\theta)_{\text{SHIRE}} = \left[\sum_{i=1}^M \frac{1}{\lambda_i^4 + \lambda_{M-i+1}^4} | \mathbf{a}^*(\theta) \mathbf{v}_i |^2 \right]^{-1} \quad (2.15)$$

We can see that these algorithms have a common factor which is the square magnitude of the product of the steering vector, $\mathbf{a}(\theta)$, and the eigenvector, \mathbf{v}_i . The only difference in each algorithm is a weighting function for this magnitude. Table 2.1 shows the different weighting functions.

	weighting function
MVE	$w_i = \frac{1}{\lambda_i}$
SHIRE	$w_i = \frac{1}{\lambda_i^4 + \lambda_{M-i+1}^4}$
SNLM	$w_i = \frac{\lambda_i^{k-1}}{(\lambda_i^2 + \sigma^2)^k}$
MUSIC	$w_i = 0$ for $i = 1, \dots, K$, $w_i = 1$ for $i = K + 1, \dots, M$
EVM	$w_i = 0$ for $i = 1, \dots, K$, $w_i = \frac{1}{\lambda_i}$ for $i = K + 1, \dots, M$

Table 2.1: A comparison of weighting functions

From Table 2.1, we can see that the MVE weights $|\mathbf{a}^*(\theta)\mathbf{v}_i|^2$ by $\frac{1}{\lambda_i}$, SHIRE weights this factor by $\frac{1}{\lambda_i^4 + \lambda_{M-i+1}^4}$, SNLM weights it by $\frac{\lambda_i^{k-1}}{(\lambda_i^2 + \sigma^2)^k}$, MUSIC weights the signal terms ($i \leq K$) by 0 (*i.e.*, same as if the K largest signal eigenvalues are infinity) and the noise terms ($K + 1 \leq i \leq M$) by unity, and EVM weights the signal terms by 0 and the noise terms by $\frac{1}{\lambda_i}$.

The estimating spectra has the following general form:

$$\begin{aligned}
 P(\theta) &= \frac{1}{\sum_{i=1}^M w_i |\mathbf{a}^*(\theta)\mathbf{v}_i|^2} \\
 &= \frac{1}{\sum_{i=1}^M \mathbf{a}^*(\theta)w_i\mathbf{v}_i\mathbf{v}_i^*\mathbf{a}(\theta)} \\
 &= \frac{1}{\sum_{i=1}^K w_i |\mathbf{a}^*(\theta)\mathbf{v}_i|^2 + \sum_{i=K+1}^M w_i |\mathbf{a}^*(\theta)\mathbf{v}_i|^2}
 \end{aligned} \tag{2.16}$$

The first-term (for $i = 1, \dots, K$) of equation (2.16) relates to the signal-subspace and the second-term (for $i = K + 1, \dots, M$) relates to the noise-subspace. The

eigenvectors in the signal-subspace (signal-eigenvectors) cause the magnitude of the spectrum to be large for the angles corresponding to the DOA, while the eigenvectors in the noise-subspace (noise-eigenvectors) cause the magnitude of the spectrum to be large for the angles not corresponding to the DOA. Therefore, the spectral estimate which is obtained using the noise-eigenvectors or using the signal-eigenvectors has better resolution and accuracy than the spectral estimate which is obtained using all the eigenvectors of the covariance matrix. Combining the signal-eigenvectors and the noise-eigenvectors degrades the performance because of the complementary relationship between the two sets of the eigenvectors. Only one set of the eigenvectors is needed for the best estimate of the DOA. However, the signal-eigenvectors are easily distorted by the noise [6]. Therefore, MUSIC and EVM use only the noise-eigenvectors to obtain the direction-of-arrivals of the target sources.

The comparison of the above algorithms is plotted in Fig. 2.1. We used the same conditions for each algorithm in the simulations. We used a linear array with 10 elements and uniform spacing of 0.5 wavelength between successive sensors. The element spacing has to be smaller than or equal to $\kappa/2$, but has to be large for small beamwidth [12]. So, we choose the spacing to be $\kappa/2$. Two sources are located at -2° and 3° . The noise is correlated and the input SNR for the array is 0 dB.

For the correlated noise case, we assumed that each sensor gets additive noise which has $N(0,1)$ magnitude and the phase of the noise is uniformly distributed in $[-\delta, \delta]$, where $N(0,1)$ represents the Gaussian normal distribution with mean 0 and standard deviation 1. We set $\delta = 3^\circ$. We collected one hundred snapshots of data and used in each simulation run. The number of signals was assumed to be known

or could be estimated [57] [59] for MUSIC and EVM. The distribution of the noise eigenvalues was assumed to be known and $k = 10$ for SNLM.

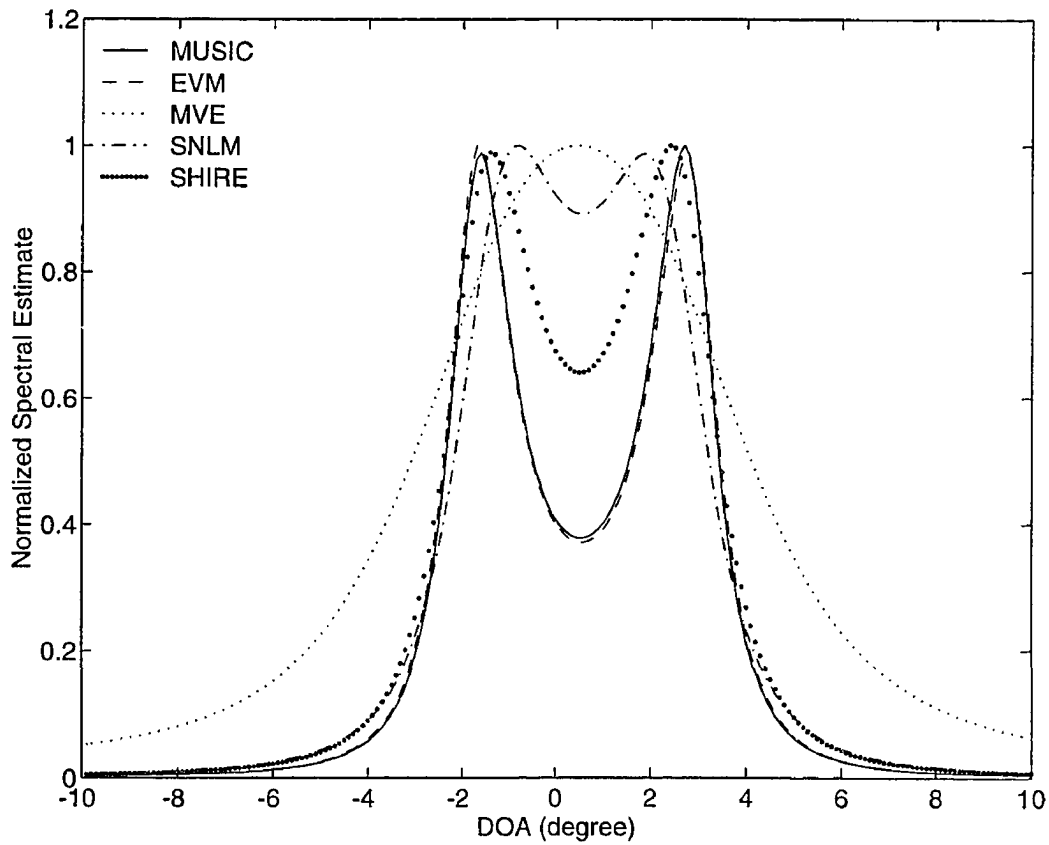


Figure 2.1: Performance comparison of algorithms

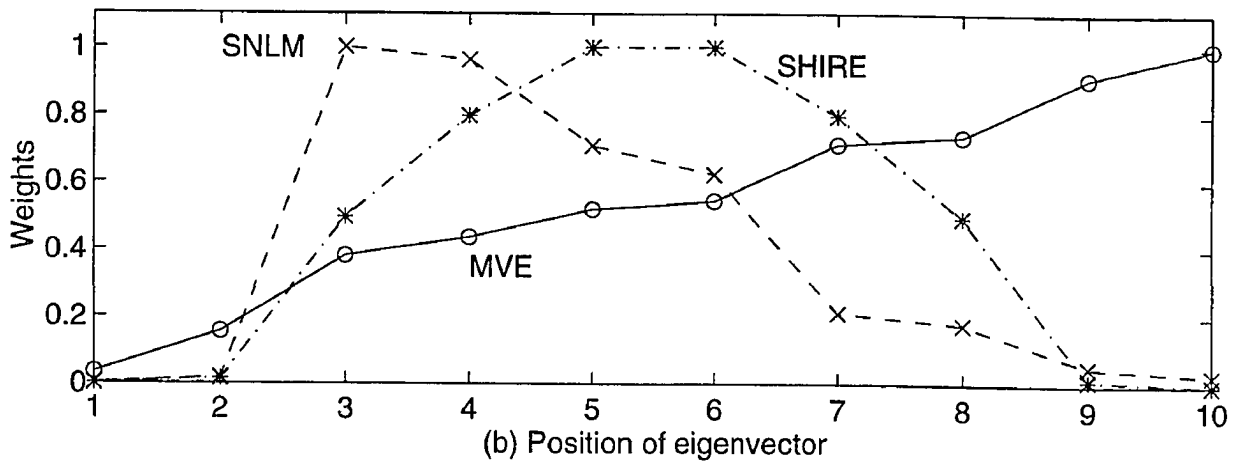
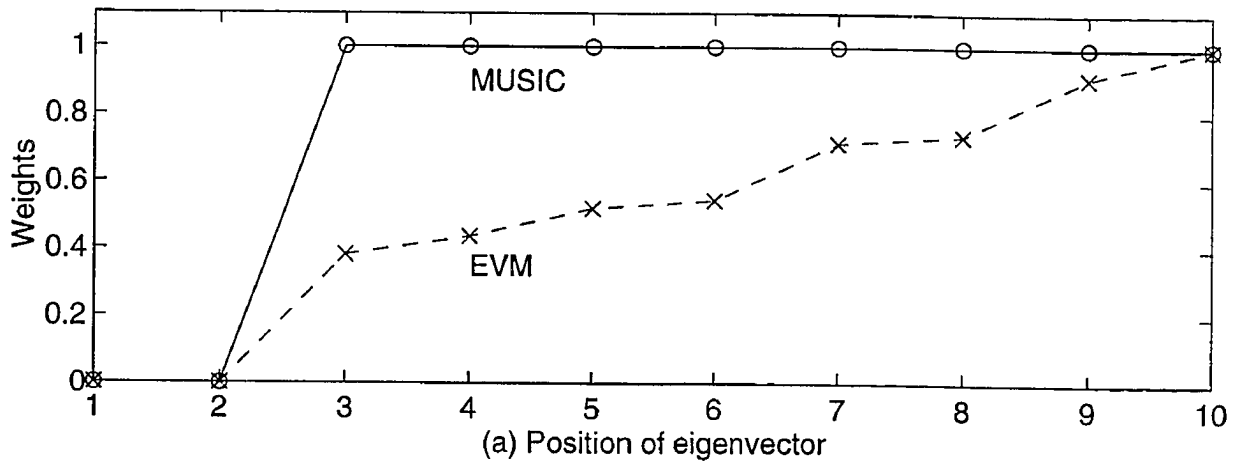


Figure 2.2: Comparison of weights on the eigenvectors, (a) the number of signals is assumed to be known, (b) the number of signals is unknown.

From Fig. 2.1, EVM and MUSIC have better resolution and accuracy than the other algorithms. This is because MUSIC and EVM do not combine the signal-eigenvectors and the noise-eigenvectors. Only the noise-eigenvectors are used to estimate the spectra for these algorithms. Fig. 2.2 shows the comparison of the weighting on the eigenvectors. The number of signals is assumed to be known in Fig. 2.2 (a). In this case, the weighting of the signal-eigenvectors (eigenvector 1 and 2) is zero. Therefore, the estimated spectrum is obtained using only the noise-eigenvectors. The spectrum estimated by these eigenvectors has better resolution and accuracy than the other algorithms (see Fig. 2.1). Fig. 2.2 (b) shows that even though the weighting of the signal-eigenvectors is small for the MVE, it has the worse resolution performance (see Fig. 2.1, MVE). From now on, our discussion is limited to the algorithms which use only the noise-eigenvectors because these algorithms have better resolution and accuracy even though they require more computations to estimate the number of signals.

What happens if the estimation of the number of the signals is incorrect? There are two cases for the wrong estimation: over estimation and under estimation. For the over estimation case, the performance degrading effect will not be a serious problem. The estimated spectra for MUSIC and EVM are obtained using the weighted sum of the noise-eigenvectors (1 for MUSIC and $\frac{1}{\lambda_i}$ for EVM). When the estimated number of signals is $K+1$ (assume that the number of signal is K), the resulting spectra will be the weighted sum of the remaining $M-K-1$ noise-eigenvectors (when the estimate of the number of signals is good, it will be the weighted sum of the $M-K$ noise-eigenvectors). Therefore, the spectral estimate is not degraded severely for over estimation of the

number of signals.

For the under estimation case, the spectral estimate is severely degraded. This is because the noise-eigenvectors are combined with the signal-eigenvectors. In this case the weighting is an important factor [53]. In the EVM, each eigenvector is weighted by the inverse of its eigenvalues $\frac{1}{\lambda_i}$ (i.e. the noise-eigenvector is weighted by the inverse of the noise-eigenvalue and the signal-eigenvector is weighted by the inverse of the signal-eigenvalue). The signal-eigenvalues have greater magnitude than the noise-eigenvalues. Therefore, in the EVM algorithm, the contribution of the signal-eigenvector is small relative to that for the noise-eigenvector. In the MUSIC algorithm, the signal-eigenvectors and the noise-eigenvectors are equally weighted. Therefore, the contribution of the signal-eigenvector is greater than with the EVM method. Based upon this comparison of results, we choose to evaluate the effect of using the forward-backward covariance matrix on the performance of the EVM.

Even if the estimation of the number of signals is correct, the estimated spectra for MUSIC and EVM are degraded as the SNR becomes low. Fig. 2.3 shows the spectral estimate of MUSIC and EVM when the SNR is minus 10 dB for our simulation when all of the other parameters are the same.

From Fig. 2.3, the estimated spectra for MUSIC and EVM can not distinguish the DOAs under these conditions. This is because the eigenvectors of the observed covariance matrix R are severely degraded by the correlated noise. The details will be further discussed in the next section. We introduce the weighted forward-backward covariance matrix to reduce this effect.

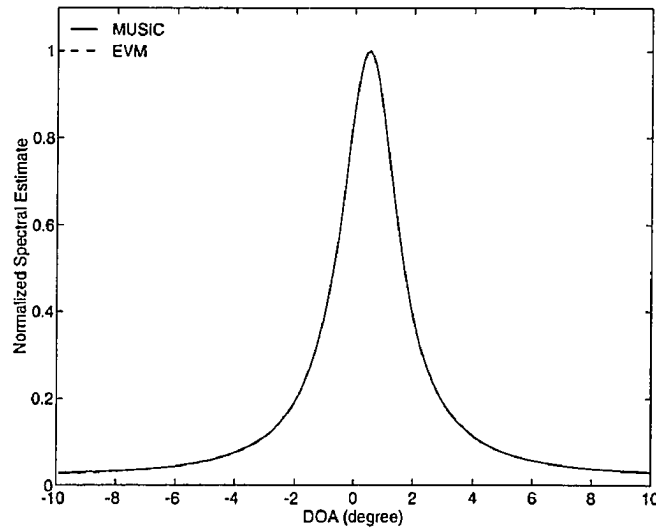


Figure 2.3: Performance comparison of algorithms, SNR = -10 dB

2.3 The Modified Eigenvector Method (MEVM)

As discussed in Section 2.2, the eigenvectors of the covariance matrix are divided into two groups. The eigenvectors related to the largest K eigenvalues span the signal-subspace. The remaining $M-K$ eigenvectors related to the smallest $M-K$ eigenvalues span the noise-subspace. In the ideal situation, the eigenvectors have the exact DOA information and they span two disjoint subspaces (the signal-subspace and the noise-subspace). However, the eigenvectors are often perturbed by correlated noise. Therefore, the subspaces that are spanned by these eigenvectors are also perturbed. Fig. 2.4 shows the relation between a noise-free eigenvector and a noisy eigenvector.

In Fig. 2.4, the eigenvector \mathbf{v} becomes \mathbf{v}' because of the correlated noise. Note that, the perturbed eigenvector \mathbf{v}' is no longer in the noise-free subspace. Since the basis of the subspace is the eigenvectors of the covariance matrix, the perturbed

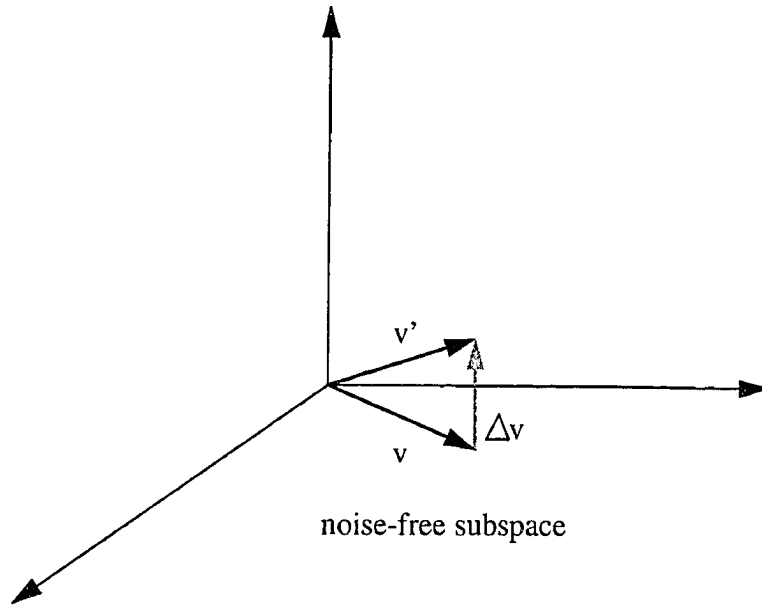


Figure 2.4: The relation between a noise-free eigenvector and a noisy eigenvector. Eigenvectors span the other subspace (called perturbed subspace). In other words, the perturbed noise-eigenvectors are in the perturbed noise-subspace and the spectrum which is obtained using these eigenvectors does not have good resolution and accuracy. Therefore, if we can develop a new subspace which is less perturbed by the same correlated noise and use the eigenvectors in this subspace, then the spectrum will have better resolution and accuracy.

Here, we introduce a weighted forward- backward covariance matrix. The weighted forward-backward covariance matrix has the following form:

$$C = wR + (1 - w)J\bar{R}J \quad (2.17)$$

Where, w is a weighting coefficient and when $w = 1$, C becomes a conventional

covariance matrix R . Therefore, the proposed method is a generalization of the conventional covariance matrix method. Also, \bar{R} is the complex conjugate of R and

$$J = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 \\ \vdots & \vdots & & & \vdots \\ 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix} \quad (2.18)$$

Note that, $J^2 = I$.

For the same noise condition, the noise-subspace of the weighted forward-backward covariance matrix is less perturbed. From Fig. 2.4, the angle between the perturbed eigenvector \mathbf{v}' and the noise-free eigenvector \mathbf{v} represents the perturbation angle of eigenvector \mathbf{v} . Similarly, when the noise-subspace is in 2 dimensions, the perturbation angle between two planes (One plane is spanned by the noise-free eigenvectors and the other plane is spanned by the perturbed eigenvectors) is the angle between the two planes. It is clear that a small angle represents a small perturbation of the noise-subspace. We simulated the perturbation angle of the noise-subspace of the weighted covariance matrix with various values of w to find the best one. The results are plotted in Fig. 2.5.

In the simulations, the different correlated noises were used with the signal-to-noise ratios (SNRs) equal to minus 10 dB, 0 dB, and 10dB. Fig. 2.5 shows the perturbation angle between the noise-free noise-subspace and the perturbed noise-subspace when w varies. From Fig. 2.5, it is shown that the smallest perturbation angle is obtained when $w = \frac{1}{2}$ regardless of the SNRs. Note that, the perturbation angle at $w = 1$ is the same as the angle of the conventional covariance matrix. This perturbation angle is always larger than that for the weighted forward-backward covariance matrix.

Therefore, we can see that the perturbation (caused by the correlated noise) for the weighted forward-backward covariance matrix is always less than that for the conventional covariance matrix. We will use $w = \frac{1}{2}$ with the forward-backward covariance matrix for the rest of this paper because our simulations show its use results in the smallest perturbation angle.

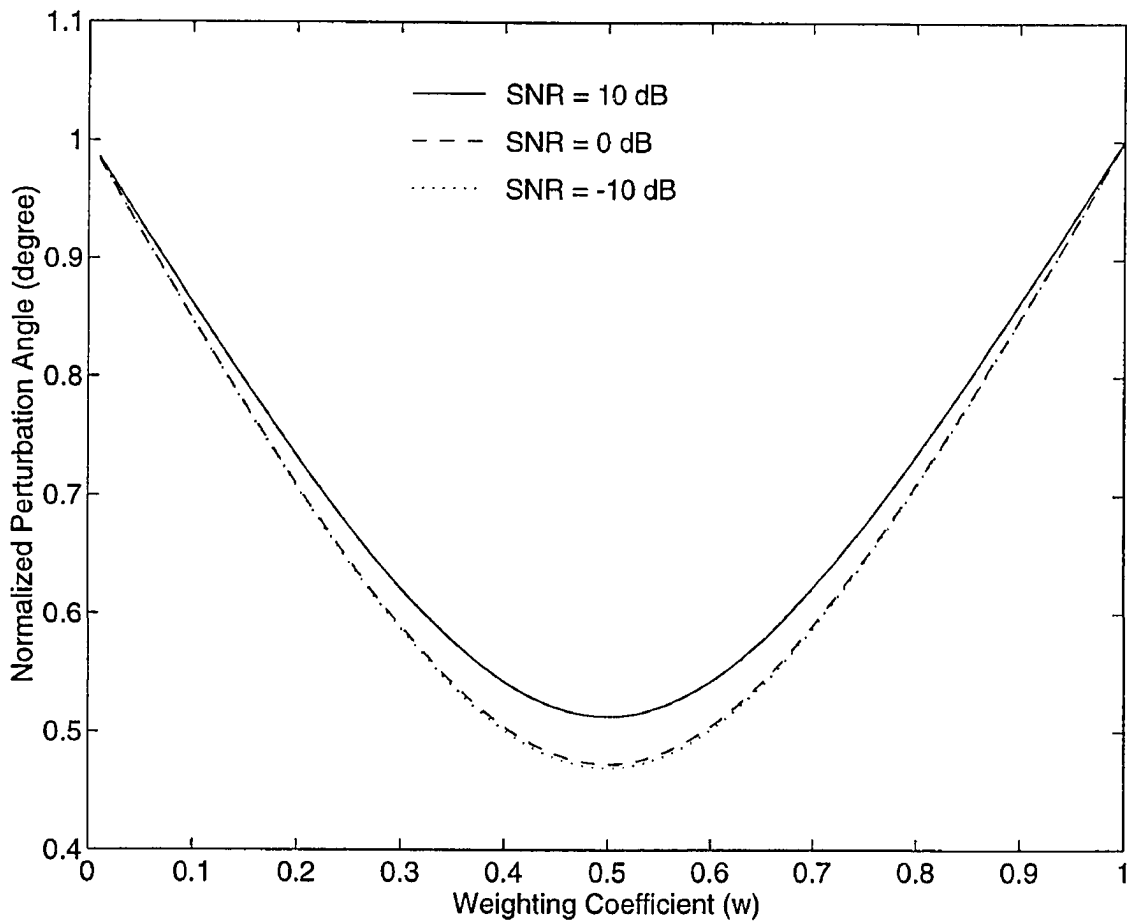


Figure 2.5: The perturbation angle between the noise-free subspace and the noisy subspace when w varies. The smallest perturbation angle is obtained at $w = \frac{1}{2}$.

We compared the noise-subspace perturbation angle for the forward-backward covariance matrix (C) and for the conventional covariance matrix (R) with different SNRs. In the simulations, we used a linear array with 10 elements and uniform spacing of 0.5 wavelength between successive sensors. One source was located at 3° . The noise was correlated and the SNRs were 5 dB, 0 dB, minus 5 dB, and minus 10 dB respectively. We collected one hundred snapshots of data and used in each simulation run. We ran the simulation one hundred times for each SNR. Fig. 2.6 shows the noise-subspace perturbation angle for C and R with the SNR equal to 5 dB. Fig. 2.7 shows the comparison of the noise-subspace perturbation angle with the SNR equal to 0 dB. Fig. 2.8 shows the comparison with the SNR equal to minus 5 dB. The comparison with the SNR equal to minus 10 dB is plotted in Fig. 2.9.

These figures show that the noise-subspace perturbation angle for the weighted forward-backward covariance matrix is always less than that for the conventional covariance matrix. They also show that as the SNR decreases, the perturbation angle for the noise-subspace becomes large as expected. Table 2.2 shows the mean-values of the perturbation angles for the noise-subspace.

SNR (dB)	Mean 1 (degrees)	Mean 2 (degrees)
5	0.5347	0.2983
0	1.7041	0.9278
-5	5.4483	2.9362
-10	18.2472	9.8402

Table 2.2: The comparison of the noise-subspace perturbation angle

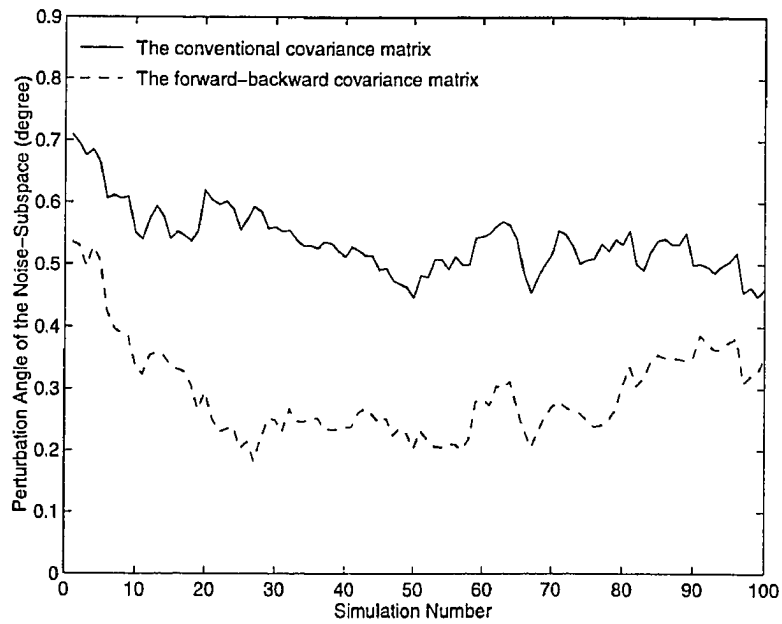


Figure 2.6: Comparison of the noise-subspace perturbation angle with SNR equal to 5 dB.

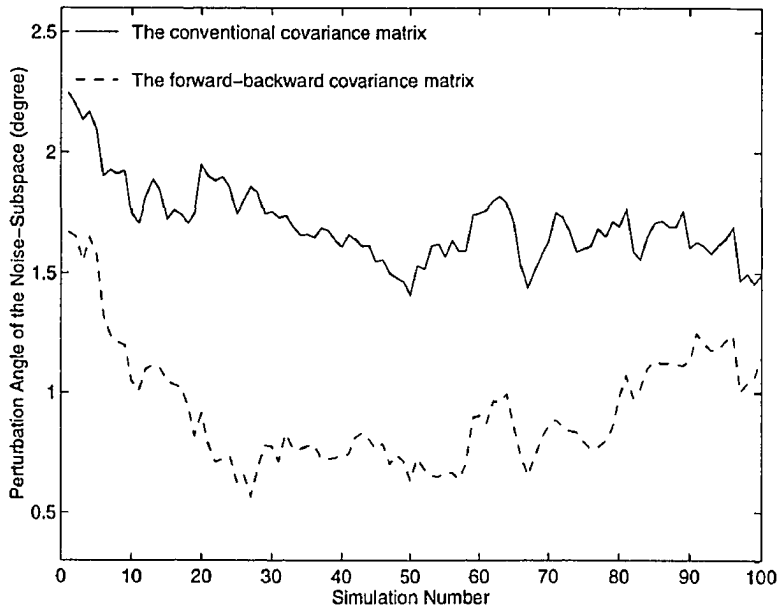


Figure 2.7: Comparison of the noise-subspace perturbation angle with SNR equal to 0 dB.

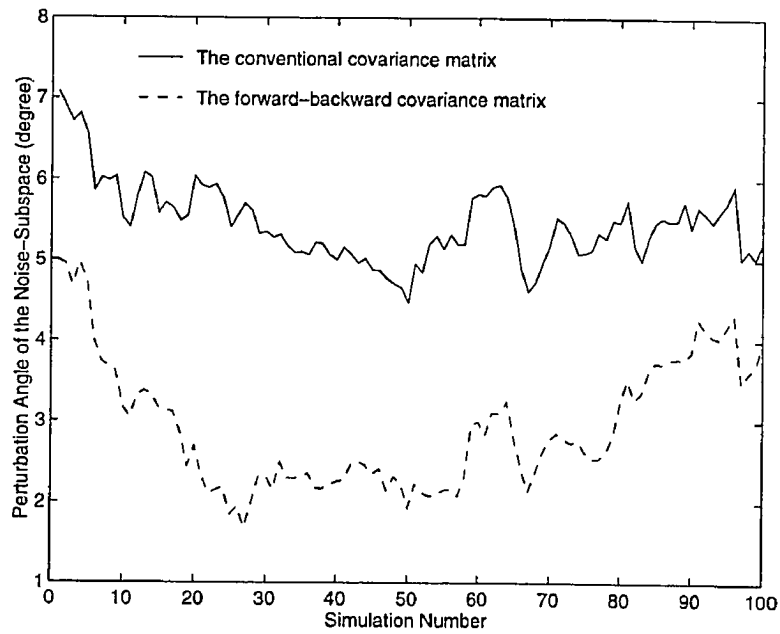


Figure 2.8: Comparison of the noise-subspace perturbation angle with SNR equal to - 5 dB.

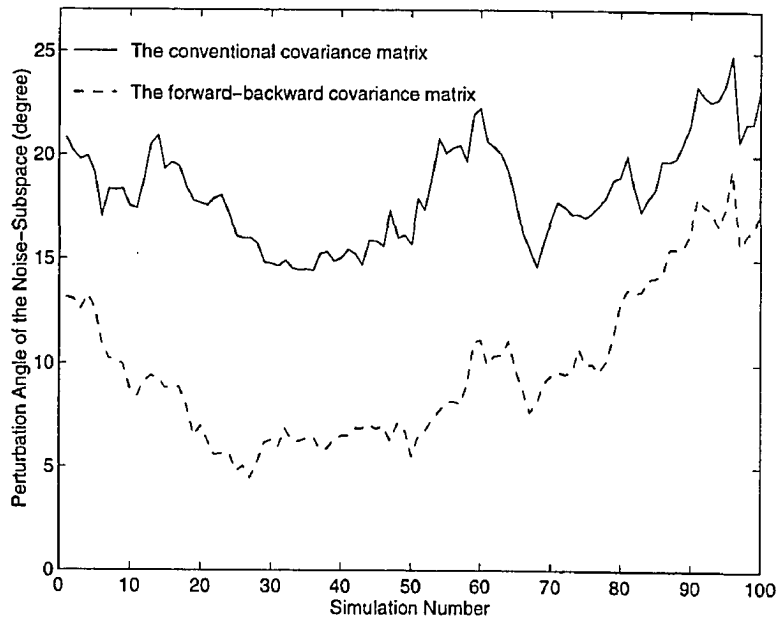


Figure 2.9: Comparison of the noise-subspace perturbation angle with SNR equal to - 10 dB.

The values, mean 1 and the mean 2, are the mean values of the perturbation angles for the noise-subspace for the conventional covariance matrix (R) and for the proposed enhanced covariance matrix (C), respectively. Table 2.2 shows that the noise-subspace using the forward-backward covariance matrix has a smaller perturbation angle than that for the noise-subspace using the conventional covariance matrix. Therefore, if we use the eigenvectors in the noise-subspace which have the smaller perturbation angle, then the spectral estimate obtained using these eigenvector will be more accurate and have better resolution.

We propose a modified eigenvector method (MEVM) to enhance the resolution of the spectral estimate for finding the direction-of-arrival of the sources. The MEVM is similar to the eigenvector method (EVM) except it uses the forward-backward covariance matrix (C) to enhance the estimates of the covariance matrix, where

$$C = \frac{1}{2}(R + J\bar{R}J) \quad (2.19)$$

Therefore, the spectral estimate of the modified eigenvector method (MEVM) is

$$P(\theta)_{\text{MEVM}} = \left[\sum_{i=K+1}^M \frac{1}{\lambda_i} | \mathbf{a}^*(\theta) \mathbf{v}_i |^2 \right]^{-1} \quad (2.20)$$

where λ_i s and \mathbf{v}_i s are the eigenvalues and the corresponding eigenvectors of the forward-backward covariance matrix C .

We simulated the MEVM using matlab to compare it with the performance of the conventional EVM. Fig. 2.10 and Fig. 2.11 show the comparison of the spectral estimates for MEVM and for the conventional EVM as the SNR decreases. In the simulations, we used a linear array with 10 elements. Two sources were located at -2° and 3° . The noise was correlated and the SNR equaled to 0 dB and minus 5 dB.

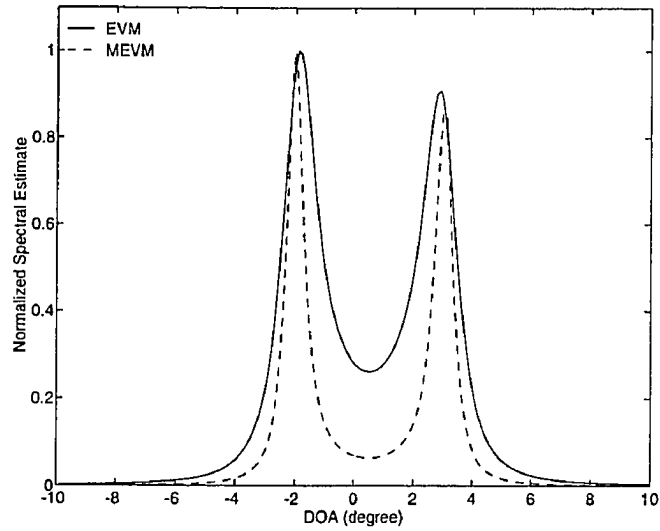


Figure 2.10: Comparison of the spectral estimates with SNR equal to 0 dB. The two sources are resolved well.

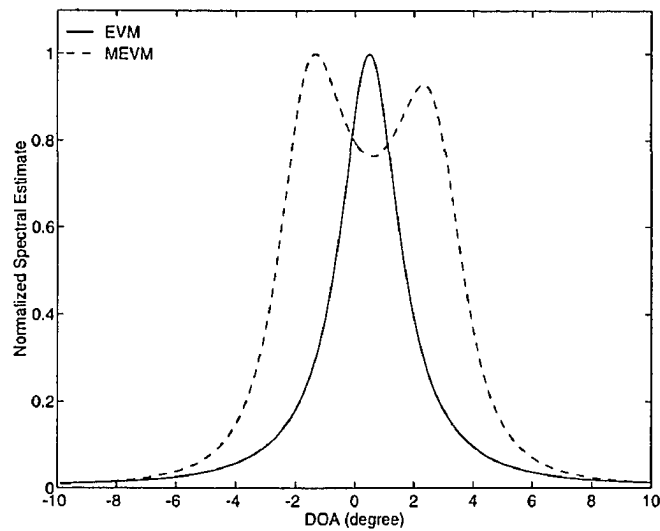


Figure 2.11: Comparison of the spectral estimates with SNR equal to - 5 dB. The two sources are not resolved as the SNR decreases to - 5 dB.

We collected one hundred snapshots of data and used them in the simulation runs.

Fig. 2.10 shows that the DOA for the two sources is resolved well for both methods when the SNR was 0 dB. As the SNR decreases to -5 dB (see Fig. 2.11), the spectral estimate (EVM) obtained using the conventional covariance matrix is distorted and can not be used to distinguish the DOA of the two sources while the spectral estimate (MEVM) obtained using the forward-backward covariance matrix can still be used to distinguish the DOAs. This is because the eigenvectors used in MEVM are less distorted than the eigenvectors used in EVM under the same noise conditions. Therefore, it is shown that the spectral estimates for MEVM has better resolution and accuracy than the spectral estimates for EVM as the SNR decreases.

We performed other simulations to show the effect when the two sources are close to each other. We used the same linear array with 10 elements. The noise was correlated and the SNR was set to 0 dB. We collected one hundred snapshots of data and used them in the simulation runs. Fig. 2.12 shows the estimated spectra when two sources are at -2° and 2° . The estimated spectra are plotted in Fig. 2.13 when two sources are closer to each other (-2° and 1.5°).

Fig. 2.10 shows that the DOA for the two sources is resolved well for both methods when two sources were located at -2° and 3° . As the two sources were moved closer to each other (-2° and 2° , see Fig. 2.12), the spectral estimates for EVM began to converge while the spectral estimate for MEVM still can be used to detect the two sources. As the two sources were more closer to each other (-2° and 1.5° , see Fig. 2.13), the spectral estimate for EVM converged and could not distinguish the DOA of the two sources while the spectral estimate for MEVM still can be used to detect

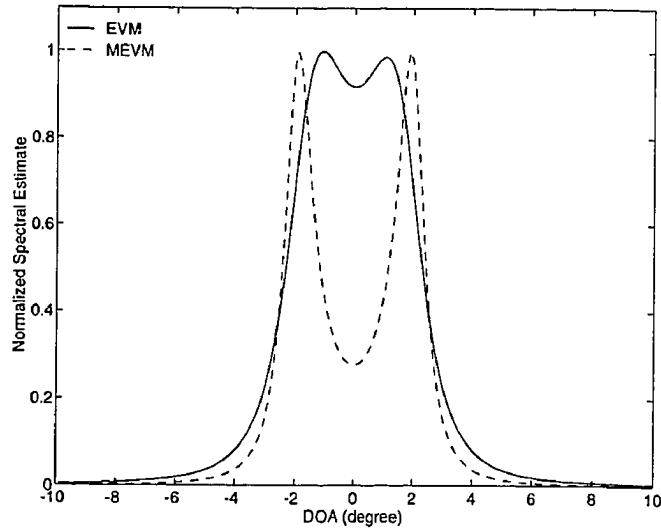


Figure 2.12: Comparison of the spectral estimates with two sources are at -2° and 2°

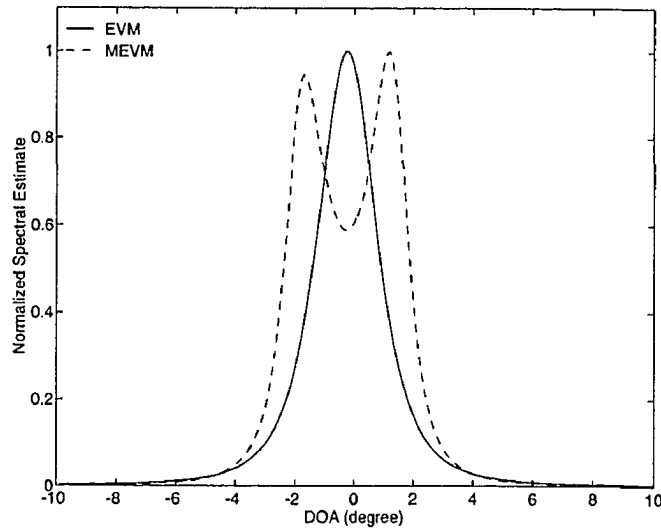


Figure 2.13: Comparison of the spectral estimates with two sources are at -2° and 1.5°

the DOA for the two sources. Therefore, MEVM can still be used to distinguish the DOAs even though the two sources are close to each other.

We have shown with the simulation results that the spectral estimate obtained using the noise-eigenvectors for MEVM has better performance than the spectral estimate obtained using the noise-eigenvectors for the conventional EVM. The MEVM is less sensitive to correlated noise and small differences in DOA (*i.e.*, the spectral estimate for MEVM has better resolution and accuracy than the spectral estimate for EVM under the same noise condition and/or when two sources are close to each other).

We performed the simulations to compare the resolution for the conventional EVM and MEVM. Fig. 2.14 and Fig. 2.15 show the comparison of two methods for several simulations. We used a linear array with 10 elements. Two sources were located at -2° and 4° . The noise was correlated and the SNR was set to minus 6 dB. We collected one hundred snapshots of data and used them in each simulation run. Estimated spectra from 20 runs using the conventional EVM are plotted in Fig. 2.14. This figure shows that the two sources are not resolved. The results using MEVM are shown in Fig. 2.15, wherein the the two sources are resolved. The simulation results show that the MEVM consistently has better resolution and accuracy than the conventional EVM.

So far, we have presented the modified eigenvector method (MEVM) to enhance the resolution of spectral estimate for finding the direction-of-arrival of the sources. We evaluated the performance of MEVM and compared the results with the conventional EVM. The comparisons show that the spectral estimate obtained using

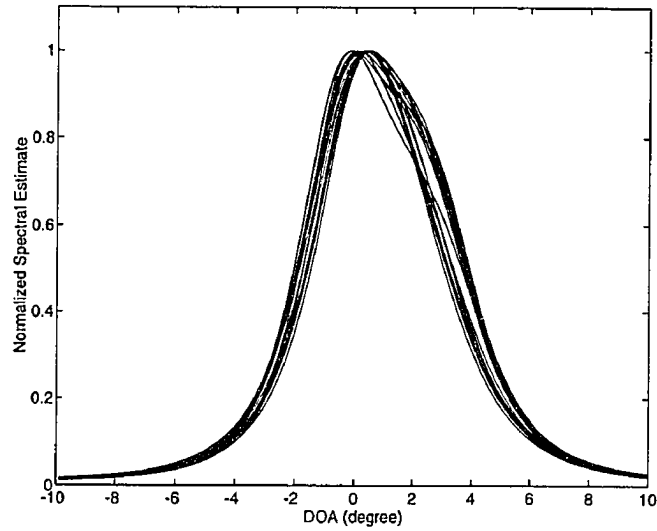


Figure 2.14: The estimated spectra for the conventional EVM. Two sources are not resolved.

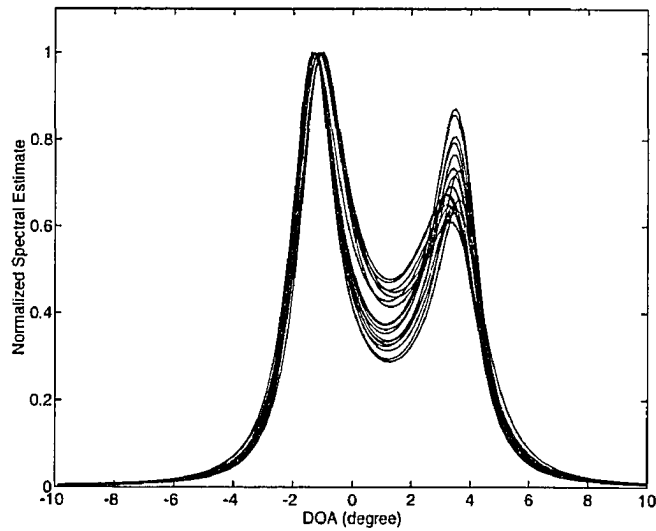


Figure 2.15: The estimated spectra for MEVM. Two sources are resolved well.

MEVM is less distorted than the spectral estimate obtained using the conventional EVM under the same noise conditions. Furthermore, MEVM has better resolution and accuracy than the conventional EVM when the sources are close to each other. We now consider how to implement this algorithm for real-time processing.

2.4 Implementing the MEVM

Section 2.3 gave the enhanced algorithm (MEVM), which uses the forward-backward covariance matrix for the direction-of-arrival system. The MEVM has 4 steps.

1. Form the data matrix X where

$$X(k) = \begin{bmatrix} x_1(k) & x_1(k-1) & \cdots & x_1(k-N+1) \\ x_2(k) & x_2(k-1) & \cdots & x_2(k-N+1) \\ \vdots & \vdots & \ddots & \vdots \\ x_M(k) & x_M(k-1) & \cdots & x_M(k-N+1) \end{bmatrix} \quad (2.21)$$

2. Compute the forward-backward covariance matrix C where

$$R(k) = \frac{1}{N} X(k) X^*(k) \quad (2.22)$$

$$C(k) = \frac{1}{2} (R(k) + J \overline{R(k)} J)$$

$$J = \begin{bmatrix} 0 & \cdots & 0 & 0 & 1 \\ 0 & \cdots & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

3. Find the eigenvalues, λ_j , and the corresponding eigenvectors, v_j , of C .
4. Find the peaks of the function $1/P(\theta)$ where

$$P(\theta) = \sum_{j=K+1}^M \frac{1}{\lambda_j} |V_j(\theta)|^2 \quad (2.23)$$

$$V_j(\theta) = \sum_{k=1}^M v_{jk} \exp^{-i\pi \sin(\theta)(k-1)}$$

where

$$\mathbf{v}_j = [v_{j1} \ v_{j2} \ \cdots \ v_{jM}]$$

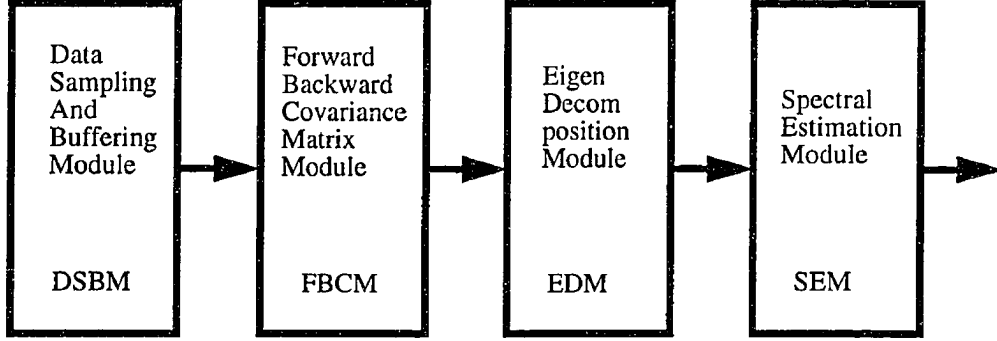


Figure 2.16: Conceptual Design of the Overall System

We are interested in developing a high performance system for implementing the MEVM. Fig. 2.16 shows the pipeline of modules required to implement the MEVM. Conceptually there are four modules in the system. The first is the data sampling and buffering module to form the data matrix. The second is the forward-backward covariance matrix module to compute the forward-backward covariance matrix from the data matrix. The third is the eigen-decomposition module to compute the required eigenvalues and the corresponding eigenvectors of the covariance matrix. The fourth is the spectral estimation module which computes the inverse of the weighted sum of the eigenvectors obtained from module 3.

The floating-point operations required in the DSBM, the FBCM, and the SEM are relatively small when compared to the number of floating-point operations required for the EDM. The EDM requires intensive computations to compute the eigenvalues

and the corresponding eigenvectors of the forward-backward covariance matrix. This is because computing eigenvalues and the corresponding eigenvectors requires lots of matrix operations (usually, $O(M^3)$ floating-point operations are required) even though we want only the partial solution of the eigen-decomposition problem. This makes it difficult to implement the whole system in real-time. We investigate the use of the block data parallel architecture (BDPA) [58] to avoid this bottleneck.

3

The Partial Eigenvalue Solution Algorithm

We have shown in chapter 2 that the spectral estimates (EVM) obtained using the noise-eigenvectors has better resolution and accuracy than the spectral estimate (MVE) obtained using all the eigenvectors of the covariance matrix. We further analyze this problem in this chapter.

The covariance matrix presented in chapter 2 can be represented by its eigenvalues and the corresponding eigenvectors.

$$\begin{aligned} C &= \sum_{i=1}^M \lambda_i \mathbf{v}_i \mathbf{v}_i^* \\ &= \sum_{i=1}^K \lambda_i \mathbf{v}_i \mathbf{v}_i^* + \sum_{i=K+1}^M \lambda_i \mathbf{v}_i \mathbf{v}_i^* \end{aligned} \quad (3.1)$$

The eigenvalues of the covariance matrix represent the power of the signals. For example, in adaptive beamforming, signal-eigenvalues represent the power of the input signal plus the input noise and the noise-eigenvalues represent the power of the input noise. Therefore, when K signals are input to the array, the K largest eigenvalues (signal-eigenvalues) are always larger than the remaining $M-K$ eigenvalues (noise-eigenvalues). The K eigenvectors (signal-eigenvectors) corresponding to the K largest eigenvalues have information related to the DOA of the sources, while the remaining

M-K eigenvectors (noise-eigenvectors), corresponding to the M-K smallest eigenvalues, have information related to the absence of the sources as discussed in chapter 2. Therefore, the spectral estimate obtained using the noise-eigenvectors or using the signal-eigenvectors has better resolution and accuracy than the spectral estimate which is obtained using all the eigenvectors of the covariance matrix. This is because the combination of the signal-eigenvectors and the noise-eigenvectors degrades the performance because of the complementary relationship between the two sets of the eigenvectors. Only one set of the eigenvectors is needed for the best estimate of the DOA.

Computing the partial solution of the eigenvalue decomposition problem is a necessary part of the high-resolution algorithm for an adaptive beamforming system. The partial eigenvalue solution algorithms are also used for digital signal processing applications, such as array signal processing [35], system identification [8] [38], image processing [44], spectrum estimation [14], filter design [36] , to name just a few. Although these algorithms that use eigenvalue decomposition can usually achieve much better performance than traditional least squares methods, their heavy computational load often makes them difficult to implement in real-time. Since these algorithms require computation of only a few of the eigenvalues and the corresponding eigenvectors from a large space, it may not be necessary to compute all of the eigenvalues and the corresponding eigenvectors. Several methods exist for the partial solution of the eigenvalue problem. QR-inverse iteration method [3], simultaneous iteration method [22] [23] [49], and subspace iteration method [42] [46] are frequently used. We compared these partial eigenvalue solution algorithms based on computational requirements and

speed of convergence.

The QR-inverse iteration method is a very efficient algorithm to find all of the eigenvalues for a matrix. The procedures are shown in Table 3.1. In the case where C

Equation	Notes
$C_k = Q_k R_k - \sigma_k I$	QR iterations with origin shift $k = 1, 2, \dots$
$C_{k+1} = Q_k^T C_k Q_k + \sigma_k I$	
$(C - \lambda_i I) \mathbf{y}_i^{(k+1)} = \mathbf{y}_i^{(k)}$	Inverse iterations $k = 1, 2, 3. i = 1, 2, \dots, K$

Table 3.1: QR-inverse iteration method

is symmetric, each of the C_k 's will also be symmetric and the sequence will converge to a diagonal matrix. After finding the eigenvalues using the QR iterations, the partial eigenvectors (say K) can be obtained using the inverse iteration method. The inverse iteration method requires only a few iterations (2 or 3). However, the QR iterations require $M-1$ Householder's transformations and a similarity transformation at each iteration and it is typically slow to converge. (Actually, the rate of convergence for λ_K depends on the ratio $\frac{\lambda_K}{\lambda_{K-1}}$ [17]). The number of iterations required to obtain the K partial eigenvalues from an $M \times M$ matrix will be explained later.

The simultaneous iteration method can be used to simultaneously find multiple eigenvalues and the corresponding eigenvectors. The procedure is shown in Table 3.2. In this Table, C is the forward-backward covariance matrix ($M \times M$) to be solved and D is an $M \times K$ initial trial matrix, where K is the number of eigenvalues to be solved. It requires four matrix-matrix multiplications, two forward-substitutions, one back-substitution, one Cholesky decompositions, and one full eigenvalue solution for

$U = CD$	matrix-matrix multiplication
$G = D^T D$	matrix-matrix multiplication
$H = D^T U$	matrix-matrix multiplication
$G = LL^T$	Cholesky decomposition
$LZ = B$	forward-substitution
$LH = Z^T$	forward-substitution
$HY_s = Y_s \Lambda_s$	full eigenvalue solution ($K \times K$)
$L^T Y = Y_s$	back-substitution
$D = UY$	matrix-matrix multiplication
Tolerance test	

Table 3.2: Simultaneous iteration method

a $K \times K$ matrix at each iteration.

Table 3.3 shows the procedure for subspace iteration using the Rayleigh-Ritz method. It requires four matrix-matrix multiplications, one QR decomposition, and

$U = CD$	matrix-matrix multiplication
$U = QR$	QR decomposition
$H = Q^T(CQ)$	2 matrix-matrix multiplications
$H = Y_s \Lambda_s Y_s^T$	full eigenvalue solution ($K \times K$)
$D^{(k+1)} = QY_s$	matrix-matrix multiplication
Tolerance test	

Table 3.3: Subspace iteration with Rayleigh-Ritz method

one full eigenvalue solution for a $K \times K$ matrix at each iteration.

We used computer simulations to evaluate the partial eigenvalue solution algorithms for the direction of arrival problem. We used matrices formed from simulated data consisting of four sinusoidal signals plus Gaussian noise. We used an equally spaced linear array with 64 elements and two hundred snapshots of data for each sim-

ulation. We tried to find four signal eigenvalues and the corresponding eigenvectors from a 64×64 matrix.

We used arbitrary orthonormal vectors for the starting vectors. The initial trial matrix D was

$$D = \begin{bmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \mathbf{d}_3 & \mathbf{d}_4 \end{bmatrix} \quad (3.2)$$

where \mathbf{d}_i is a 64×1 zero vector with the i th position equal to 1. In the simulations, the numerical stabilities are related to the number of iterations. Therefore, the estimated eigenvalues and the corresponding eigenvectors at each iteration are closer to the real eigenvalues and the corresponding eigenvectors as the number of iterations is increased. We set the allowable tolerance to 10^{-4} . The comparison of the convergence for each algorithm is given in Table 3.4.

	mean	std	min	max
OR iterations with origin shift	47.3	23.609	15	105
Simultaneous iteration	5.68	0.4899	5	7
Subspace iteration	4.68	0.4889	4	6

Table 3.4: Convergence comparison

These simulation results show that the subspace iteration with Rayleigh-Ritz method converges faster than the QR iteration method or the simultaneous iteration method. It also requires fewer operations (per iteration) than the other methods. Therefore, the subspace iteration with Rayleigh-Ritz method is the best choice for a partial eigenvalue solution algorithm when we consider the computational tasks and the rate of convergence.

The above algorithms are useful to find a few dominant eigenvalues and the corresponding eigenvectors. However, most of the signal processing algorithms require a few of the smallest eigenvalues and the corresponding eigenvectors. By modifying these algorithms, we can obtain the few smallest eigenvalues and the corresponding eigenvectors that are required for most of the signal processing algorithms.

The symmetric matrix C and the inverse of C , C^{-1} , can be represented by its eigenvalues and corresponding eigenvectors.

$$C = \sum_{i=1}^M \lambda_i \mathbf{v}_i \mathbf{v}_i^T \quad (3.3)$$

$$C^{-1} = \sum_{i=1}^M \frac{1}{\lambda_i} \mathbf{v}_i \mathbf{v}_i^T$$

When $\lambda_1 > \lambda_2 > \dots > \lambda_M$, λ_1 is a dominant eigenvalue. $\frac{1}{\lambda_M}$ is a dominant eigenvalue for C^{-1} . Therefore, if we compute a dominant eigenvalue and the corresponding eigenvector of C^{-1} , then that is the smallest eigenvalue and corresponding eigenvector of C . However, we have to compute C^{-1} to obtain the smallest eigenvalue and the corresponding eigenvector. This requires approximately M^3 multiplications. Moreover, the system may be defective if C is near singular. The alternate way to solve this problem is to solve $CU = D$ using back-substitution instead of solving

$C = Q_1 R_1$	QR decomposition
$R_1 U = Q_1^T D$	back-substitution
$U = Q_2 R_2$	QR decomposition
$H = Q_2^T (C Q_2)$	2 matrix-matrix multiplications
$H = P_s \Lambda_s P_s^T$	full eigenvalue solution ($K \times K$)
hline $D^{(k+1)} = Q_2 P_s$	matrix-matrix multiplication
Tolerance test	

Table 3.5: Partial eigenvalue solution algorithm

$U = C^{-1}D$. Therefore, if we replace the first step of Table 3.1, Table 3.2, and Table 3.3 by $CU = D$ then we obtain the smallest eigenvalues and the corresponding eigenvectors.

The resulting subspace iteration with Rayleigh-Ritz method is shown in Table 3.5. In the Table, C is an $M \times M$ matrix to be solved and D is an initial trial matrix ($M \times K$), where K is a number of partial eigenvalues and the corresponding eigenvectors to be solved.

4

Block Data Parallel Architecture

4.1 The Advantages of the BDPA

Computing eigenvalues and the corresponding eigenvectors of a matrix is very computationally intensive because of the required matrix operations. Matrix operations also find important applications in many digital signal processing areas such as array signal processing, system identification, image processing, spectrum estimation, data compression, and adaptive filtering. However, these operations require tremendous computations and this makes them difficult to implement in real-time.

Many parallel computer structures [52] [54] have been developed to solve this problem. Some of the traditional parallel computer structures such as bus-organized multiprocessor systems and SIMD (Single Instruction Multiple Data) array processor systems [13] are reliable and easy to extend. However, the processors' contention for shared resources (buses, memory) limits the number of processors which can be effectively used. SIMD array processors have the potential to provide the computational power. However, as the number of processors becomes large, the interconnection network often becomes cost-prohibitive. For some applications it is also difficult to balance the input/output bandwidth with the demands for processing capability.

Normally, signal processing tasks and matrix operations possess a large amount of inherent parallelism. Many parallel algorithms and parallel structures have been developed to exploit this parallelism [15]. However, most parallel algorithms are optimized for implementation on general purpose computers. General purpose computers can not achieve the high system throughput required for real-time processing because of limitations due to supervision overhead and data communication problems.

Most parallel multiprocessor system such as systolic arrays and hypercube multiprocessor systems have a synchronous structure. A synchronous system achieves its parallelism by synchronous clock-step operations [20]. This implies that all operands have to be ready before any processor can start its designated operation. This strictly synchronous operation imposes a severe timing restriction on the system design and causes implementation difficulties such as the clock skew problem for large scale systems (i.e. each processing element in the array may not receive the clock signal at the same time) [29] [31].

A wave-front array [28] [30] replaces the requirement for correct timing by a requirement for a correct sequence of operations to overcome the globally synchronous timing problem. However, if the handshaking for the wave-front array is done at the word level, then the resources required to implement the handshaking protocol limit the overall system efficiency and throughput.

Algorithms designed for systolic arrays and wave-front arrays use an algorithm partitioning strategy. In an algorithm partitioning strategy, each processor implements a part of the algorithm and then passes its intermediate results and possible the input data to other processors in the array. This strategy can be effectively used

to distribute the computational load in a processor array. However, the use of this strategy may lead to unnecessary data movement among processing elements. Moreover, only the edge processors of a systolic array or a wave-front array have access to input or output devices. Processing results go through processor by processor in order to reach the one which can interface to the output device. This unnecessary data movement may increase system management and data communication overhead, and may increase hardware complexity. It may also increase the data dependency among the processing elements.

In a data partitioning strategy [1] [58], each processor receives a different portion of the data and attempts to complete all the necessary computations for its assigned data partition. When it is necessary to communicate with other processors, this communication should be minimized. Therefore, the data partitioning strategy can be effectively used with many digital signal processing algorithms.

A block data parallel architecture (BDPA) [58] effectively uses both the data partitioning strategy and algorithm partitioning strategy. By effectively using both strategies, the BDPA achieves high system throughput and high system efficiency.

The BDPA uses a globally asynchronous and locally synchronous clock distribution scheme. Therefore, the BDPA uses FIFO buffers to relax synchronization requirements between the processing elements, between the input module and the processing array, and the processing array and the output module. Once a processing element detects that input data is available, it does not require any further checking for the presence of data until a block of input data has been transferred. This simple block data transfer protocol together with the use of the FIFO buffers essentially

eliminates the overhead associated with asynchronous data transfer.

In the systolic array implementation, each processing element (PE) in the array is connected to nearest neighbor PEs. In the array, processing elements work together using pipelining. Through pipelining, data enters the array through designated periphery processing elements and is piped according to a global clock through PEs one clock at a time. On each clock cycle a PE receives its input, performs its designated computation, and produces output to be used as input to neighbor PEs on the next clock cycle. Output from the array is received through designated periphery processing elements. However, the data transfer between PEs is performed element by element. The PEs receives input and produces output with a small number of operations. Therefore, the total number of operations in a PE is not much larger than the total number of input and output operations. Sometimes, this may result in an I/O-bound system [26] [29]. One of the important aspects of utilizing a high performance architecture effectively is to increase the ratio of floating-point operations to data movements to balance the computations and the data movements.

Currently the processing speed for processors is increasing at a much higher rate than the speed of the data bus. Therefore, it is important to improve the ratio of floating-point operations to data movements to fully utilize the hardware. In our parallel architecture, we improve this ratio by processing the data by block (multiple data elements). By partitioning the input data into blocks, each PE processes a block of data while reusing the data internally. The reuse of data reduces the total data movements, therefore increasing the ratio of floating-point operations to data movements.

Systolic array architectures [5] [37] [43] [48] offers a high degree of parallelism. These architectures use a fixed number of processing elements (PEs) to solve a fixed-sized problem. For example, the QR decomposition of a 16×16 matrix requires a 16×16 triangular array of PEs for systolic array implementation, while the QR decomposition of a 30×30 matrix requires a 30×30 triangular array of PEs. Therefore, the systolic array architecture is suitable for computing the full eigenvalues and the corresponding eigenvectors of the 16×16 or 30×30 system that it was designed for.

However, in many digital signal processing applications, the number of eigenvalues required for the partial solution as well as the size of the input matrix may vary. These properties make use of the systolic array infeasible. For example, in adaptive beamforming, the number of the partial eigenvalues involved varies with the number of signals present. The input matrix size can vary depending on the particular application.

In the BDPA, the different ranges of problem sizes can be mapped onto the array through a wrap-around interconnection of PEs. Therefore, the number of PEs does not depend on the size of the input matrix and/or the number of partial eigenvalues to be solved. For example, it can solve a problem for not only a 64×64 matrix but also for a 75×75 matrix using the same number of PEs (say 8) with high efficiency and throughput. Also, it can extract any number of the smallest (largest) eigenvalues and the corresponding eigenvectors without having to change the number of PEs in the processing array. The comparison between a systolic array, a wave-front array, and a BDPA shown in Table 4.1.

	systolic array	wave-front array	BDPA
strategy	algo. parti.	algo. parti.	data/algo. parti.
clocking	synch.	asynch./synch.	asynch./synch.
data-dependency	intensive	relaxed	relaxed
comm. between PEs	intensive	intensive	relaxed
control scheme	difficult	simple	simple
computations/data movement	low	low	high
granularity	low	low	high
processing element utilization	high	low	high

Table 4.1: The comparison of a systolic array, a wave-front array, and a BDPA

4.2 BDPA Configuration

As previously mentioned, the BDPA is based upon the use of block data processing and the block data flow paradigm [1] [58]. A BDPA system consists of three modules as shown in Fig. 4.1. The input device supplies a serial stream of data to the input module (IM) which groups the data into blocks to be sent to the processor array (PA). Each processing element in the PA works with its assigned block, sending intermediate values to the next processing element in the PA and sending outputs grouped in blocks to the output module (OM). The OM assembles these blocks into a serial output stream for the output device.

4.2.1 Input Module

The input module (IM) serves as a buffer between the host system (or external input device) and the processor array. It includes two or more FIFO buffers and it converts the input data stream into blocks of data. In designing the IM, we assumed that the input data blocks will be formed sequentially. Therefore, while FIFO₁ receives the

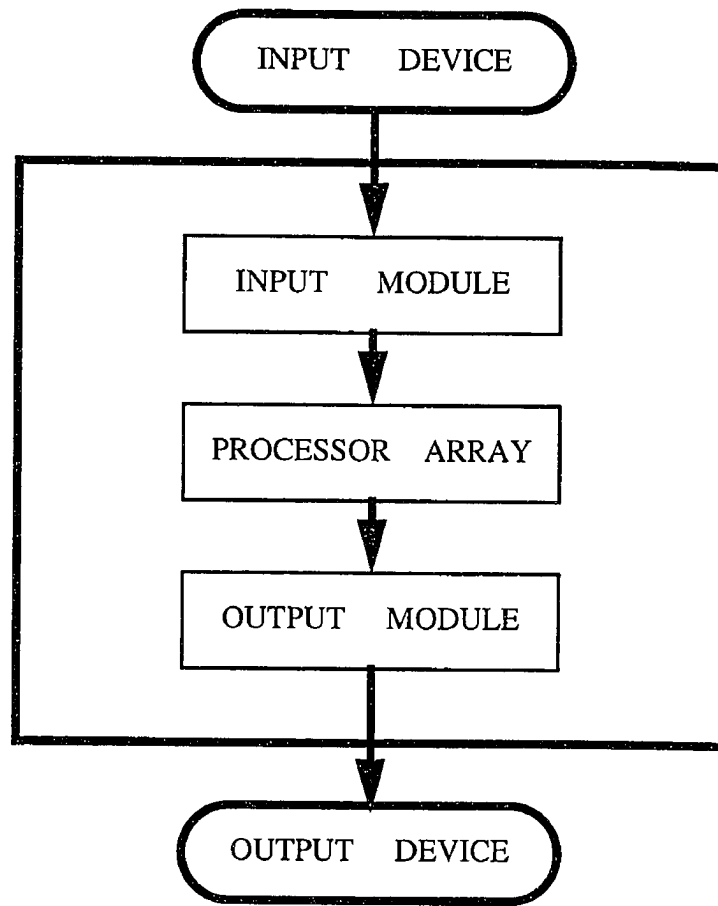


Figure 4.1: Block Diagram for the BDPA

data from the input device, FIFO₂ sends a block of data to the PA using one of the input data buses (In_Bus₂) and vice versus (the data distribution from the input data buses will be explained in Section 4.2.2). So, the IM continuously provides the input data to the PA. The block diagram of the IM with 2 FIFO buffers is shown in Fig. 4.2.

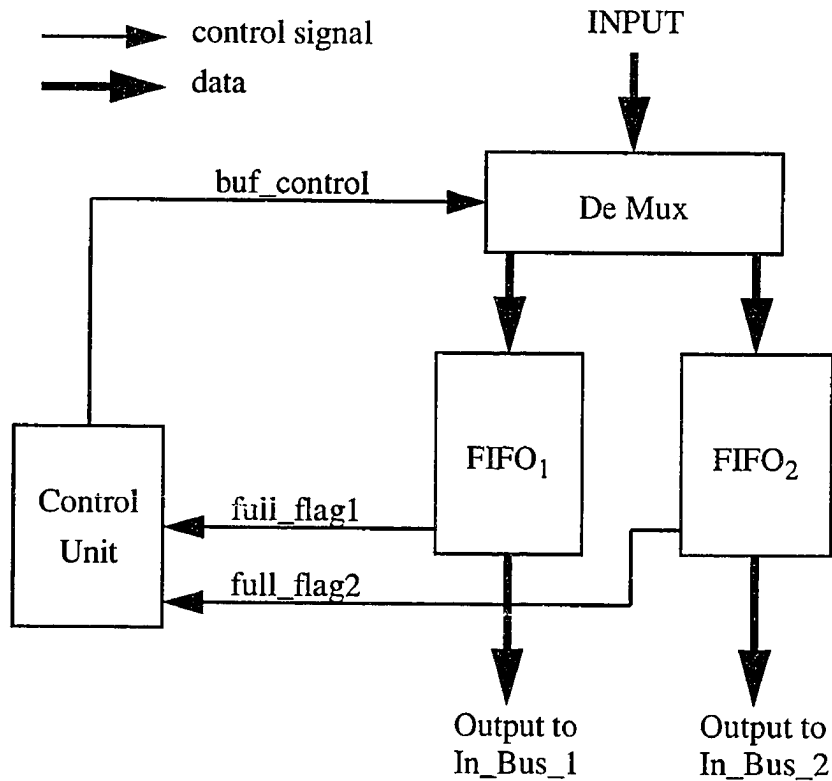


Figure 4.2: Input Module with 2 FIFO buffers

4.2.2 Processor Array

The processor array (PA) has a sufficient number of processing elements (PEs) to provide the necessary computational power for real-time signal processing or fast matrix operations. The block diagram for the PA is shown in Fig. 4.3.

The data communications between PEs are local and uni-directional by design. This permits the use of FIFO buffers for the data communications. The FIFO buffers also provide asynchronous data communication capability which in turn relaxes the requirement for strict timing between PEs. This is an important advantage as we

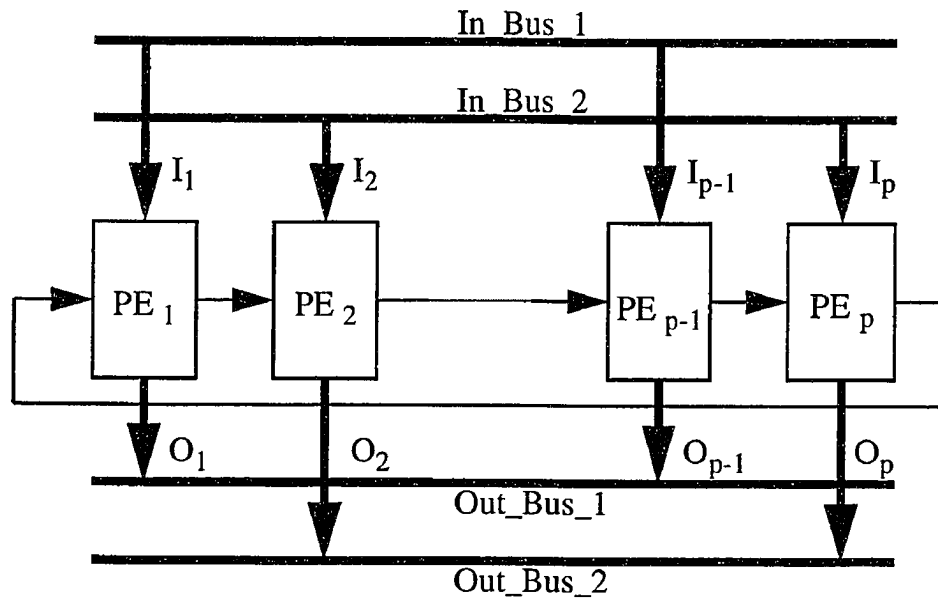


Figure 4.3: Block Diagram for the Processor Array

increase the number of PEs in the BDPA. Each PE has a separate input channel and a separate output channel. The PEs are divided into an odd number PE group and an even number PE group. Each PE group is connected to input FIFO buffers and output FIFO buffers through its assigned data buses. The data blocks are sequentially routed to the PEs. Thus, each input bus must be ready to handle every other data block. This permits time for the IM to route the previous data block to the appropriate PE. Also, the two input data buses and two output buses alleviate bus contention problems. This is true because the data communications in this architecture are regular, predictable, and sequential. This is a significant advantage for implementing large scale systems.

In the architecture, the input data is partitioned into blocks and each block of data is assigned to a PE through its assigned data bus (*i.e.*, block 1 is assigned in PE₁, block 2 is assigned in PE₂ and so on). A block of data could be a column, a row, or a submatrix of a large matrix. Each processing element (PE_{*i*}) receives block *i* from the IM (*i.e.*, NORTH direction) and receives intermediate results from the previous processor (PE_{*i-1*}, EAST direction). PE_{*i*} performs the computations required for the block, sends the intermediate results required to the next PE (PE_{*i+1*}, WEST direction), and sends the block of results to output module (OM, SOUTH direction).

Once data first becomes available for a given PE, then data is always available until its assigned block of data has been exhausted. Each PE which currently has data can perform computations at its maximum possible rate. This means that speedup is essentially linear for the BDPA.

After each PE finishes its computational tasks it receives the next block for its new assignment. This makes the system flexible with regard to varying the input matrix size. This is true because each PE receives the new data block as soon as it completes the computations for the previous block. For example, when the QRM₁ processes 61×61 matrices using four PEs, the last column of the first 61×61 matrix (say C_1) is assigned to the first PE and the first column of the second 61×61 matrix (say C_2) is assigned to the second PE, etc. Therefore, the number of PEs in the BDPA does not have to depend on the size of the input matrix. This is a significant advantage for implementing the system when the size of the input matrix varies with time. For example, the sizes of matrices U and S for the partial eigenvalue

decomposition problem depend on the number of eigenvalues to be solved and the number of eigenvalues to be solved varies with the number of input signals which varies with time. More details of the architectures will be discussed in chapter 5.

4.2.3 Output Module

Fig. 4.4 shows an example of the block diagram of an Output Module (OM). It collects processing results from each of the processing elements and converts the blocks of data

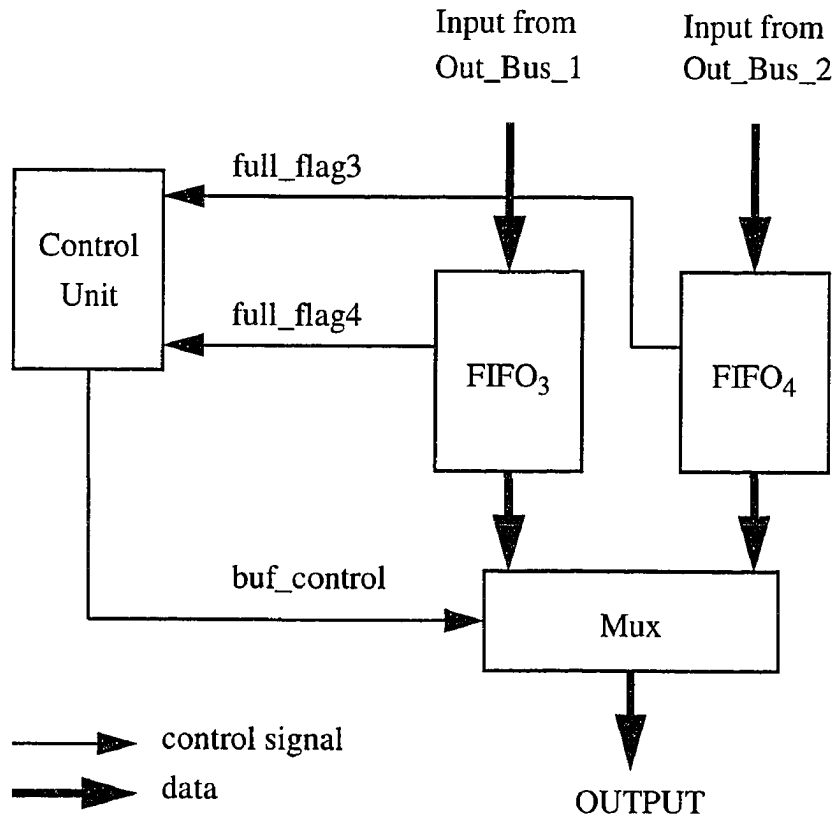


Figure 4.4: Output Module

into a synchronized output data stream for the output device.

5

A High Performance Parallel Architecture

5.1 The System Building Block

Our simulations show that the subspace iteration with Rayleigh-Ritz method has fast convergence and requires less computational time for the partial eigenvalue solution problem as compared to the other algorithms in our study. Therefore, we designed a parallel architecture to implement the subspace iteration with Rayleigh-Ritz method.

Fig. 5.1 shows the pipeline of modules required to implement the subspace iteration with Rayleigh-Ritz method. Conceptually there are $n+2$ modules in the proposed pipelined system, where n is the number of iterations required to compute K partial eigenvalues and the corresponding eigenvectors. The first is the input module (IM). The IM (see Fig. 4.2) converts the input data stream into blocks of data and continuously supports the block of input data to PEs in the iteration module (ITM). Each ITM performs the computations required to complete an iteration for the subspace iteration method. The OM (see Fig. 4.4) collects processing results from the PEs in the last ITM and converts the blocks of data into an output data stream in the appropriate format for the output device.

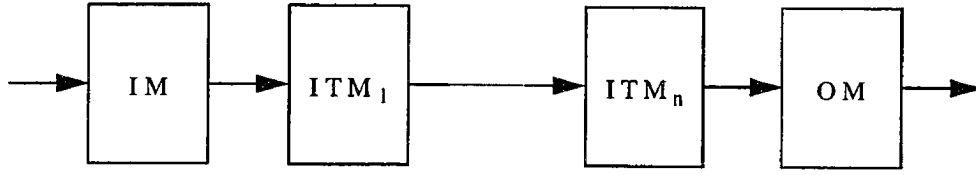


Figure 5.1: Pipelined subspace iteration algorithm flow

5.2 Iteration Module

The iteration module (ITM) performs the computations required to complete an iteration of the subspace iteration algorithm. An iteration of the subspace iteration algorithm proceeds by the following steps :

1. Compute the QR decomposition of E to find R_1 and update D .

$$E = Q_1 R_1 \quad (5.1)$$

$$Q_1^T E = R_1$$

$$Q_1^T D = B$$

2. Solve for U using back-substitution.

$$R_1 U = B \quad (5.2)$$

3. Compute the QR decomposition of U and find Q_2 .

$$U = Q_2 R_2 \quad (5.3)$$

4. Multiply Q_2 by E .

$$T = EQ_2 \quad (5.4)$$

5. Multiply T by Q_2^T .

$$H = Q_2^T T \quad (5.5)$$

6. Find the full eigenvalue solution of H .

$$H = Y_s \Lambda_s Y_s^T \quad (5.6)$$

7. Find the new trial matrix D by multiplying Y_s by Q_2 .

$$D = Q_2 Y_s \quad (5.7)$$

Our parallel architecture uses the BDPA in each submodule to implement one step of the algorithm. Several such submodules cooperate in a pipelined manner to implement a complete iteration of the subspace iteration algorithm. Fig. 5.2 shows the pipeline of submodules required to compute an iteration of the partial eigenvalue solution algorithm.

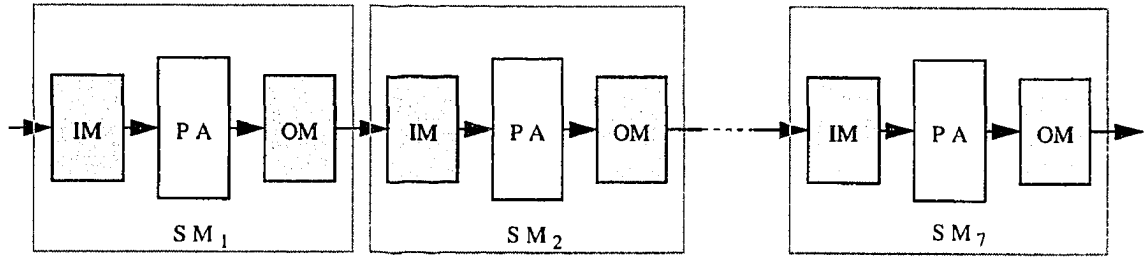


Figure 5.2: Pipelined submodules to solve the partial eigenvalue solution algorithm

An iteration of the subspace iteration algorithm proceeds by 7 steps. Therefore, there are 7 submodules (SMs) in the proposed pipelined parallel architecture implementation. Each submodule uses the BDPA to solve one step of the subspace iteration

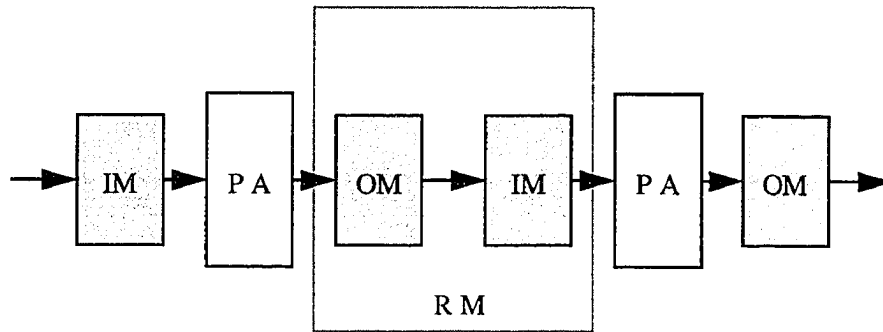


Figure 5.3: Register Module

algorithm. However, the IM and the OM are repeatedly used in the implementation. We combine the IM and the OM to form a register module (RM) as shown in Fig. 5.3. The RM continuously receives data from the processing elements in the previous submodule and provides the data to the next submodule. Therefore, the RM serves as a buffer between submodules. Fig. 5.4 shows a block diagram for the RM. From Fig. 5.4, while Buf₁ sends data to the processing elements in the next module, Buf₂ receives data from the processing elements in the previous submodule or vice versa. With this configuration, the RM can simultaneously receive data from the previous submodule while sending data to the next submodule.

The iteration modules for the partial eigenvalue solution algorithm are shown in Fig. 5.5. Seven submodules are required to compute an iteration of the algorithm.

The first is the QR decomposition module1 (QRM₁) which computes an upper triangular matrix (R_1) of E using Given's rotations. The trial matrix D is also updated during the process. The second is a back-substitution module (BSM) which solves for U where $R_1 U = Q_1^T D$. The third is the QR decomposition module2 (QRM₂)

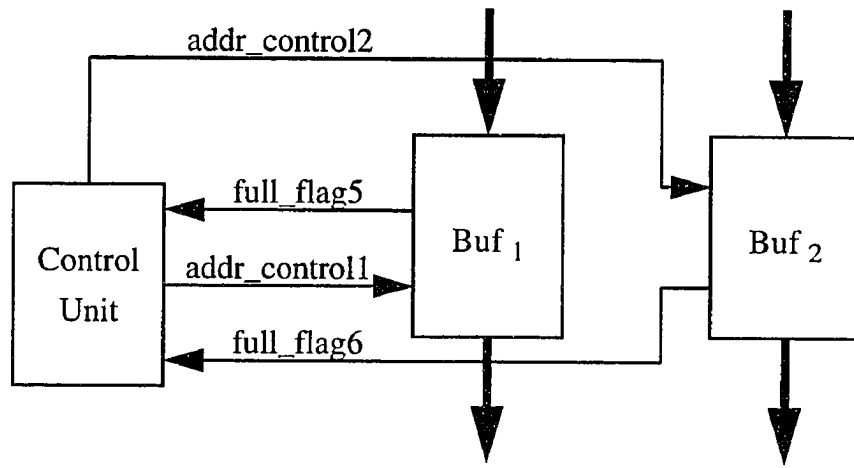


Figure 5.4: Block Diagram for the Register Module

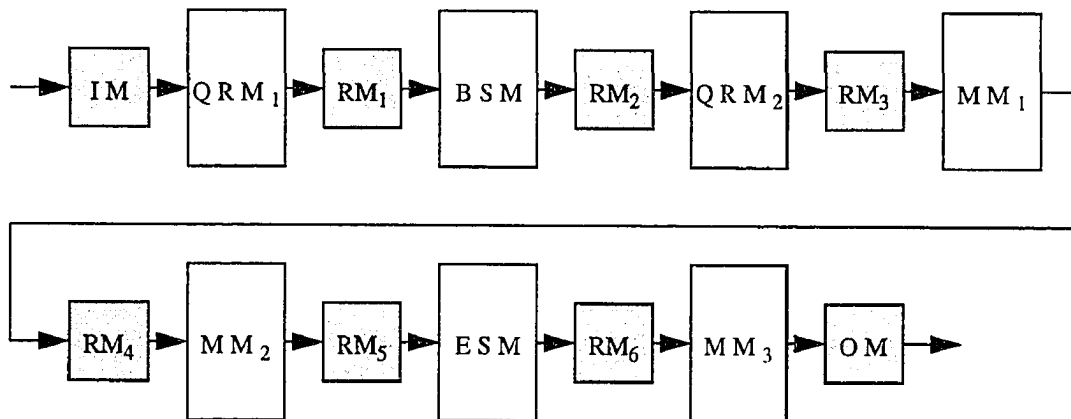


Figure 5.5: Iteration modules for the partial eigenvalue solution algorithm

which computes Q_2 from U using the modified Gram-Schmidts method. The fourth is a matrix-matrix multiplication module1 (MM_1) which computes T by multiplying Q_2 by E . The fifth is matrix-matrix multiplication Module2 (MM_2) which computes the $K \times K$ matrix H . The sixth is an eigenvalue solution module (ESM) which computes the eigenvalues and the corresponding eigenvectors of the $K \times K$ matrix H . The seventh is the matrix-matrix computation Module3 (MM_3) which computes the new trial matrix D . These processes are repeated in each ITM until it satisfies the convergence conditions.

In the architecture, the number of ITMs is related to the number of iterations required to obtain convergence. We simulated the algorithm to find the required number of ITMs when we extracted the four largest eigenvalues and the corresponding eigenvectors from a 64×64 matrix. In the simulations, we used 64×64 matrices formed from simulated data consisting of four sinusoidal signals plus Gaussian noise with signal-to-noise ratio (SNR) equal to 0 dB. We used one hundred simulation runs for the computations. Table 5.1 shows that 33 of the one hundred runs converge after four iterations. In the same manner, Table 5.1 shows that 66 of the one hundred

number of iterations	4	5	6
simulation runs	33	66	1

Table 5.1: Number of iterations required to compute the four largest eigenvalues and the corresponding eigenvectors from 64×64 matrices

runs converge after five iterations, and 1 of the one hundred runs converges after six iterations. The number of iterations are related to the number of ITMs. Therefore,

four iterations would have required four ITMs, etc. Based on these results, we consider six ITMs operating in a pipelined fashion to be adequate for this problem. In rare instances, if the convergence criteria is not satisfied within 6 iterations, the last ITM will be used to test for the convergence and then continue the iterations. Since the last ITM is the only one affected by this modified procedure, the other modules can stop processing and hold their current data until the last ITM has satisfied the convergence criteria. Then, the system can proceed with normal operation.

5.3 QR Decomposition Module1

The first step of the partial eigenvalue solution algorithm is the computation of the QR decomposition [16] [41] of a covariance matrix E to solve the linear equations $EU = D$. This module performs the QR decomposition of E to find R and B .

$$\begin{aligned} EU &= D & (5.8) \\ Q^T EU &= Q^T D \\ RU &= B \end{aligned}$$

Where, $E = QR$ and $B = Q^T D$.

We used Given's rotation for the QR decomposition. The purpose of the Given's rotation is to annihilate the subdiagonal elements of matrix E and reduce it to upper triangular form. For example, if we have a matrix E ,

$$\begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix} \quad (5.9)$$

We need to zero out $e_{21}, e_{31}, e_{41}, e_{32}, e_{42}, e_{43}$ in order to obtain the upper triangular matrix. We use row 1 as the reference row and form an orthogonal matrix Q_{21} to

annihilate an element e_{21} . The orthogonal matrix Q_{21} is

$$Q_{21} = \begin{bmatrix} c_{21} & s_{21} & 0 & 0 \\ -s_{21} & c_{21} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.10)$$

where

$$c_{21} = \frac{e_{11}}{\sqrt{e_{11}^2 + e_{21}^2}} \quad (5.11)$$

$$s_{21} = \frac{e_{21}}{\sqrt{e_{11}^2 + e_{21}^2}}$$

Multiplying E by Q_{21} gives

$$Q_{21}E = \begin{bmatrix} c_{21} & s_{21} & 0 & 0 \\ -s_{21} & c_{21} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix} = \begin{bmatrix} \dot{e}_{11} & \dot{e}_{12} & \dot{e}_{13} & \dot{e}_{14} \\ 0 & \dot{e}_{22} & \dot{e}_{23} & \dot{e}_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix} \quad (5.12)$$

As a result of this computation, the elements in row 1 are updated as follows:

$$\dot{e}_{11} = c_{21}e_{11} + s_{21}e_{21} \quad (5.13)$$

$$\dot{e}_{12} = c_{21}e_{12} + s_{21}e_{22}$$

$$\dot{e}_{13} = c_{21}e_{13} + s_{21}e_{23}$$

$$\dot{e}_{14} = c_{21}e_{14} + s_{21}e_{24}$$

The updated elements in row 2 are

$$\dot{e}_{21} = -s_{21}e_{11} + c_{21}e_{21} = 0 \quad (5.14)$$

$$\dot{e}_{22} = -s_{21}e_{12} + c_{21}e_{22}$$

$$\dot{e}_{23} = -s_{21}e_{13} + c_{21}e_{23}$$

$$\dot{e}_{24} = -s_{21}e_{14} + c_{21}e_{24}$$

In a similar way, the subdiagonal element, e_{31} , is annihilated. We use the orthogonal matrix Q_{31} to annihilate e_{31} .

$$Q_{31} = \begin{bmatrix} c_{31} & 0 & s_{31} & 0 \\ 0 & 1 & 0 & 0 \\ -s_{31} & 0 & c_{31} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.15)$$

where

$$c_{31} = \frac{\dot{e}_{11}}{\sqrt{\dot{e}_{11}^2 + c_{21}^e}} \quad (5.16)$$

$$s_{31} = \frac{e_{21}}{\sqrt{\dot{e}_{11}^2 + c_{21}^e}}$$

Multiplying $Q_{21}E$ by Q_{31} we obtain

$$Q_{31}Q_{21}E = \begin{bmatrix} c_{31} & 0 & s_{31} & 0 \\ 0 & 1 & 0 & 0 \\ -s_{31} & 0 & c_{31} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{e}_{11} & \dot{e}_{12} & \dot{e}_{13} & \dot{e}_{14} \\ 0 & \dot{e}_{22} & \dot{e}_{23} & \dot{e}_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix} = \begin{bmatrix} \ddot{e}_{11} & \ddot{e}_{12} & \ddot{e}_{13} & \ddot{e}_{14} \\ 0 & \dot{e}_{22} & \dot{e}_{23} & \dot{e}_{24} \\ 0 & \dot{e}_{32} & \dot{e}_{33} & \dot{e}_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix} \quad (5.17)$$

Row 1 and row 3 are updated as a result of this computation. Similarly, we can compute c_{41} and s_{41} and form Q_{41} to zero out e_{41} . Then, the subdiagonal elements of the row 2 (e_{32} and e_{42}) and the subdiagonal element of the row 3 (e_{43}) are zeroed in this way until the upper triangular matrix is obtained.

The resulting upper triangular matrix is

$$R = Q^T E = \begin{bmatrix} \hat{e}_{11} & \hat{e}_{12} & \hat{e}_{13} & \hat{e}_{14} \\ 0 & \hat{e}_{22} & \hat{e}_{23} & \hat{e}_{24} \\ 0 & 0 & \hat{e}_{33} & \hat{e}_{34} \\ 0 & 0 & 0 & \hat{e}_{44} \end{bmatrix} \quad (5.18)$$

where

$$Q = Q_{43}Q_{42}Q_{32}Q_{41}Q_{31}Q_{21} \quad (5.19)$$

Thus, in Given's rotation, the subdiagonal elements of the first column are annihilated first, then the elements of the second column, and so forth until an upper

triangular form is eventually reached. The matrix Q is a product of matrices used to zero elements of E . Each element e_{ij} is zeroed by multiplying E by an orthogonal matrix Q_{ij} . The matrix Q_{ij} is formed from the identity matrix by replacing the diagonal (i, i) and (j, j) elements by c_{ij} , the (i, j) element by s_{ij} , and the (j, i) element by $-s_{ij}$ where

$$\begin{aligned} c_{ij} &= \frac{e_{jj}}{\sqrt{e_{jj}^2 + e_{ij}^2}} \\ s_{ij} &= \frac{e_{ij}}{\sqrt{e_{jj}^2 + e_{ij}^2}} \end{aligned} \quad (5.20)$$

Multiplying E by Q_{ij} updates row i (\mathbf{e}_i) and row j (\mathbf{e}_j) of E :

$$\begin{aligned} \mathbf{e}_i &\leftarrow -s_{ij}\mathbf{e}_i + c_{ij}\mathbf{e}_j \\ \mathbf{e}_j &\leftarrow c_{ij}\mathbf{e}_i + s_{ij}\mathbf{e}_j \end{aligned} \quad (5.21)$$

The element e_{ij} becomes

$$\hat{e}_{ij} = -\frac{e_{ij}e_{jj}}{\sqrt{e_{jj}^2 + e_{ij}^2}} + \frac{e_{jj}e_{ij}}{\sqrt{e_{jj}^2 + e_{ij}^2}} = 0 \quad (5.22)$$

Therefore, a subdiagonal element, e_{ij} , is zeroed in this way, and the two rows \mathbf{e}_i and \mathbf{e}_j are affected for each element e_{ij} that is zeroed.

Systolic array implementations for QR decomposition have been developed by Chen and Yao [7], Gentleman and Kung [27] and M. Moonenn and J. Vandewalle [40]. These implementations require $O(n^2)$ processing elements (PEs) for matrix triangularization. The systolic array [7] consists of boundary PEs and internal PEs. In the arrays, each boundary PE computes the multiplication pairs (c_{ij}, s_{ij}) , and sends them to a neighboring internal PE. The internal PE receives the multiplication pairs and a corresponding row element, updates the corresponding row element and

computes the corresponding element of the upper triangular matrix. It also sends out the multiplication pairs and the updated row element.

In the Chen and Yao's systolic array [7], elements of the upper triangular matrix R are stored in the registers of their corresponding PEs. However, they have to be transferred to be used in the back-substitution module. Therefore, the total number of input and output operations for an internal PE is 8, while the operations required in the PE are 4 multiplications and 2 additions. This may result in an I/O-bound computation [26] [29], since the ratio of floating-point operations to the data movement [10] is low. We need more floating-point operations in a PE or we must reduce the data movements to improve the ratio.

Our parallel architecture (see Fig. 5.6) improves the ratio by processing the data by block, which reduces the number of data movements between each PE by reuse of the data internally. In the architecture, input data is partitioned into column blocks and each column block of E is assigned to a PE (*i.e.*, column 1 is assigned in PE_1 , column 2 is assigned in PE_2 and so on). Each PE receives the multiplication pairs from an EAST neighboring PE, updates the corresponding row elements, annihilates the elements of the subdiagonal, and sends the corresponding multiplication pairs to a WEST neighboring PE.

When we use 4 PEs to reduce the 64×64 matrix to an upper triangular matrix, it operates as follows :

1. PE_1 annihilates the subdiagonal elements of column 1 and passes multiplication pairs to PE_2 . After PE_1 finishes all the computations, it passes the results to the Register Module (RM) and takes row 5 as the new assignment.

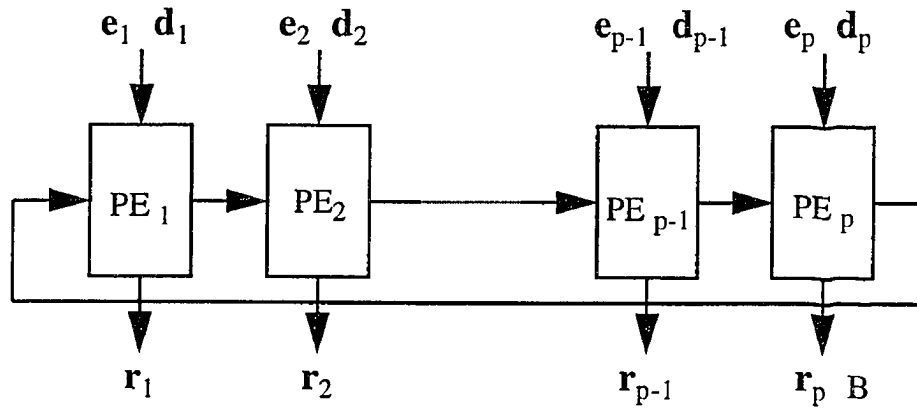


Figure 5.6: QR decomposition Module1

2. PE_2 starts to update column 2 as soon as the first one of the multiplication pairs comes from PE_1 . After it has used all the multiplication pairs from PE_1 , it starts to annihilate the subdiagonal elements of column 2. PE_2 also passes its multiplication pairs to PE_3 and takes row 6 as the new assignment.
3. This process continues until all the columns have been processed.

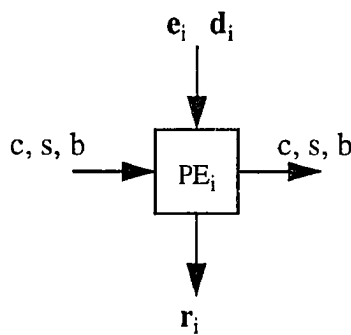


Figure 5.7: A processing element in QRM₁

Our architecture partitions the input data into column blocks and assigns them one by one to each PE. A PE performs all the computations required for the column and generates the resulting column of the output matrix. Passing the multiplication pairs are the only data movements between PEs for the matrix triangularization algorithm, and it is local and uni-directional. Therefore, this approach improves the ratio of floating-point operations to the data movement by reducing unnecessary data movements. Also, the design of the interconnection network is simplified since the data movements are uni-directional. The PE also computes $Q^T D (=B)$ while it annihilates the subdiagonal elements.

The PE in the QR decomposition Module1 (QRM₁) is shown in Fig. 5.7. PE_{*i*} performs as follows :

1. Receives column *i* of *E* and row *i* of *D* from NORTH
2. Updates column *i*

For $q = 1$ to $i - 1$

For $p = q$ to *col_size*

receives multiplication pairs from WEST

$etmp = e_{qi}$

$e_{qi} = -s_{pq} * etmp + c_{pq} * e_{pi}$

$e_{pi} = c_{pq} * etmp + s_{pq} * e_{pi}$

passes multiplication pairs to EAST

end For

end For

3. Annihilates the subdiagonal elements

For $p = i + 1$ to col_size

$$r_{pi} = \sqrt{e_{ii}^2 + e_{pi}^2}$$

$$c_{pi} = e_{ii}/r_{pi}$$

$$s_{pi} = e_{pi}/r_{pi}$$

sends multiplication pairs to EAST

end For

4. Updates $B (= Q^T D)$

For $q = 1$ to $i - 1$

For $t = 1$ to num_eigen

receive intermediate results of b_{qt} from WEST

$$btmp = b_{qt}$$

$$b_{qt} = -s_{iq} * btmp + c_{iq} * b_{it}$$

$$b_{it} = c_{iq} * btmp + s_{iq} * b_{it}$$

if i is not a last column

sends updated b_{qt} to EAST

end For

end For

if i is not a last column

For $t = 1$ to num_eigen

sends updated b_{it} to EAST

end For

5. Sends the results (column i of R) to SOUTH

Here, we can see that the ratio of floating-point operations to data movements is increased. For example, when a $M \times M$ matrix is processed, the computations required in a PE for the 2nd column are :

- column elements are updated $M - 1$ times.
 - 4 multiplications and 2 additions for each update.
- $M - 2$ column elements are annihilated.
 - 2 multiplications, 1 addition, 1 square root, 2 divisions for each annihilation.
- $2K$ elements of B are updated.
 - 2 multiplications and 1 addition for each update.

The total number of operations required to complete all the computations for the 2nd column in a PE is $(6M + 4K - 8)$ multiplications, $(3M + 4K - 4)$ additions, $(M - 2)$ square roots, and $(2M - 4)$ divisions. We calculate the number of clock cycles required to complete the computations for the 2nd column. The number of clock cycles is computed based on the programmable TMS320C40 digital signal processing (DSP) processor from Texas Instruments [21] (see Table 5.2). The computation of a square root takes 11 clock cycles using TMS320C40.

From Table 5.2, the total clock cycles required to complete all the computations for the 2nd column in a PE is $45M + 16K - 78$. The total number of data movements to complete all the computations for the 2nd column in a PE is :

- input
 - column 2 from IM (M).
 - multiplication pairs from PE_{i-1} ($(M - 1) \times 2$).
 - intermediate B from PE_{i-1} (K).
- output
 - multiplication pairs to PE_{i+1} ($((M - 1) + (M - 2)) \times 2$).
 - intermediate B to PE_{i+1} ($2K$).
 - results to OM (column 2 of R) (2).

	operations	clock cycles
ADDF	floating-point addition	2
ADDI	integer addition	2
ASSIGN	assignment or =	1
BR_NT	branch not taken	2
BR_T	branch taken	4
DIVI	integer division	4
DIVF	floating-point division	8
LOGICAL	logical expression evaluation	2
MOD	modulus operator	40
MULI	integer multiplication	2
MULF	floating-point multiplication	2
READ	reading from a communication channel	1
WRITE	writing to a communication channel	1

Table 5.2: The number of clock cycles required for a floating-point operation, based on the TMS320C40

The total number of data movements is $7M + 3K - 6$.

Based on these results, we compute the ratio of the total computations to complete the required computations in a PE versus the total data communications required in a PE. Table 5.3 shows the rate of computational clock cycles to data movements when the number of partial eigenvalues to be solved (K) is 4.

matrix size	8×8	16×16	32×32	64×64
rate	5.5806	5.9831	6.2000	6.3128

Table 5.3: The rate of computational clock cycles to data movements

From Table 5.3, the rate of floating-point operations to data movements becomes high as the block size becomes large (in this application, the block size is the number of elements in a column). Also, the rate of computational clock cycles to data movements is much higher than the 1.5 in the systolic array implementation [7]. This is an advantage for high throughput applications.

Table 5.4 shows the simulation results for QRM_1 with 2 PEs in the module. In the simulations, we use 64×64 matrices for E and 64×4 matrices for the initial trial matrix D . The same matrices are solved using 4 PEs in Table 5.5. Table 5.6, Table 5.7, Table 5.8, Table 5.9, and Table 5.10 show the simulation results when 6 PEs, 8 PEs, 10 PEs, 16 PEs, and 22 PEs are used respectively to solve the same problems.

The matrix dimension is 68×64 , in the Tables. This is because the QRM_1 not only computes the upper triangular matrix R from E (64×64) but also updates the trial matrix D (64×4 , see equation (5.8)). Therefore, a PE receives not only a column of E (64×1 vector) but also a row of D (4×1 vector).

algorithm	QRM ₁
matrix dimensions	68 × 64
number of matrices	50
number of PEs	2
run time	149526925 cycles
real-time rate	0.00145526
avg. utilization	92.2662 %

Table 5.4: Simulation result for QRM₁ with 2 PEs

algorithm	QRM ₁
matrix dimensions	68 × 64
number of matrices	50
number of PEs	4
run time	76518859 cycles
real-time rate	0.00284374
avg. utilization	90.1495 %

Table 5.5: Simulation result for QRM₁ with 4 PEs

algorithm	QRM ₁
matrix dimensions	68 × 64
number of matrices	50
number of PEs	6
run time	51598841 cycles
real-time rate	0.00421715
avg. utilization	89.1253 %

Table 5.6: Simulation result for QRM₁ with 6 PEs

algorithm	QRM ₁
matrix dimensions	68 × 64
number of matrices	50
number of PEs	8
run time	39971479 cycles
real-time rate	0.00544388
avg. utilization	86.2883 %

Table 5.7: Simulation result for QRM₁ with 8 PEs

algorithm	QRM ₁
matrix dimensions	68 × 64
number of matrices	50
number of PEs	10
run time	32208966 cycles
real-time rate	0.00675588
avg. utilization	85.6674 %

Table 5.8: Simulation result for QRM₁ with 10 PEs

algorithm	QRM ₁
matrix dimensions	68 × 64
number of matrices	50
number of PEs	16
run time	21562664 cycles
real-time rate	0.0100915
avg. utilization	79.9780 %

Table 5.9: Simulation result for QRM₁ with 16 PEs

algorithm	QRM ₁
matrix dimensions	68 × 64
number of matrices	50
number of PEs	22
run time	15311814 cycles
real-time rate	0.0142112
avg. utilization	78.9114 %

Table 5.10: Simulation result for QRM₁ with 22 PEs

The number of matrices refers to the number of consecutive covariance matrices (say $E_1, E_2, \dots, E_{49}, E_{50}$) processed in this module. The run time shows the total clock cycles required to complete the computations for these matrices. The real-time rate represents the highest possible input rate for which the data can be processed in real-time.

$$\text{real-time rate} = (\text{total number of input data})/(\text{run time}) \quad (5.23)$$

A real-time rate 0.1 represents that the input data can be processed in real-time when an element of data is input to this module every 10 clock cycles.

Table 5.4 shows that the real-time rate for 2 PEs is 0.00145526 even though it has high utilization (92.2662 %) of the processing elements. The real-time rate can be improved by increasing the number of PEs or by pipelining the internal operations of the PEs. Table 5.11 shows the real-time rate and speedup as the number of PEs is increased. It shows that as the number of PEs increases, the real-time rate also increases.

In Table 5.11, N represents the number of PEs, and $T(N)$ represents the run time. It shows that as the number of PEs increases, the run time decreases and the real-time

algorithm		QRM ₁		
matrix size		68 × 64		
number of matrices		50		
N	T(N)	real-time rate	speedup	avg. utilization
	cycles			%
4	76518859	0.00284374	3.9082	90.1495
6	51598841	0.00421715	5.7957	89.1253
8	39971479	0.00544388	7.4793	86.2883
10	32208966	0.00675588	9.2848	85.6674
16	21562664	0.01009150	13.8690	79.9780
22	15311814	0.01421120	19.5308	78.9114

Table 5.11: Simulation results for QRM₁ with different number of processing elements

rate and the speedup increases almost linearly. The speedup was computed by

$$\text{speedup} = T(1)/T(N) \quad (5.24)$$

where, $T(1)$ is a run time to execute all the processes using 1 PE, while $T(N)$ is a run time to execute all the processes using N PEs. In the Table, $T(1)$ is assumed to be $2 \times T(2)$. Fig. 5.8 shows the speedup of the architecture as the number of PEs is increased.

In many digital signal applications, the number of eigenvalues required for the partial solution as well as the size of the matrix may vary with time. These properties make the use of the systolic array infeasible. This is because the systolic array uses a fixed number of processing elements for a fixed sized matrix. In our parallel architecture, the number of processing elements does not depend on the size of the matrix to be processed (*i.e.*, the same number of PEs can be used for different sizes of matrices and/or for the solution of a different number of eigenvalues). Table 5.12 shows the average utilization of the processing elements when the matrix size changes.

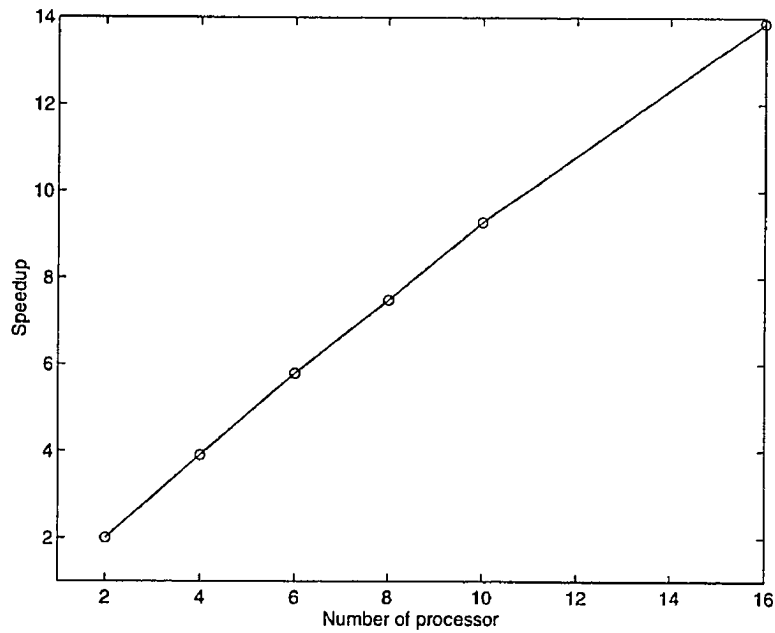


Figure 5.8: The speedup of QRM₁

matrix size	50 × 50	62 × 62	63 × 63	64 × 64	65 × 65	70 × 70
avg. utilization (%)	90.0842	90.4069	90.7073	90.1495	90.3061	90.6712

Table 5.12: The utilization comparison for different matrix sizes. The number of eigenvalues to be solved is assumed to be 4. Four PEs are used.

Table 5.12 shows that the size of the matrix may be an even number, or an odd number without changing the number of PEs (the number of PEs is 4 in this application) in QRM₁. This is because the PEs can continuously receive the columns of the matrix modulo the number of PEs (i.e., if the last column of E_1 is assigned to PE₂, then the first column of E_2 is assigned to PE₃) so, it can continuously compute its tasks without interruption.

The number of eigenvalues and the corresponding eigenvectors to be solved varies with time in most of the signal processing applications. For example, the number of

signals can be 1 or may be 6 for adaptive beamforming. Our parallel architecture still can be used to solve this problem without having to change the number of PEs in the module. Table 5.13 shows the simulation results when the number of eigenvalues to be solved varies. It shows that, the same number of PEs can still be used with high

number of signals	1	2	3	4	5	6
avg. utilization (%)	89.8986	89.9872	90.0708	90.1495	90.1971	90.2693

Table 5.13: The utilization comparison with varying of the number of eigenvalues to be solved. E is a 64×64 matrix and B and D are the $64 \times K$ matrices where K is the number of eigenvalues to be solved. Four PEs are used.

utilization for a different number of eigenvalues to be solved. Therefore, this parallel architecture is more flexible than the systolic array.

5.4 Back-Substitution Module

The back-substitution module (BSM) computes U from the equation

$$RU = B \quad (5.25)$$

where, R and B are obtained from the QRM₁. Since R is a $M \times M$ matrix and U and B are $M \times K$ matrices, this module solves K elements of U simultaneously. For example, when M is equal to 4 and K is equal to 2, we have

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ & r_{22} & r_{23} & r_{24} \\ & & r_{33} & r_{34} \\ & & & r_{44} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix} \quad (5.26)$$

From equation (5.26), the linear equations are solved in reverse order. Therefore, the fourth row of U is computed first.

$$\begin{aligned} u_{41} &= \frac{b_{41}}{r_{44}} \\ u_{42} &= \frac{b_{42}}{r_{44}} \end{aligned} \quad (5.27)$$

Then, the third row of U is computed.

$$\begin{aligned} u_{31} &= \frac{b_{31} - r_{34}u_{41}}{r_{33}} \\ u_{32} &= \frac{b_{32} - r_{34}u_{42}}{r_{33}} \end{aligned} \quad (5.28)$$

These procedures are repeated until the first row of U is obtained.

Fig. 5.9 shows the block diagram of the BSM.

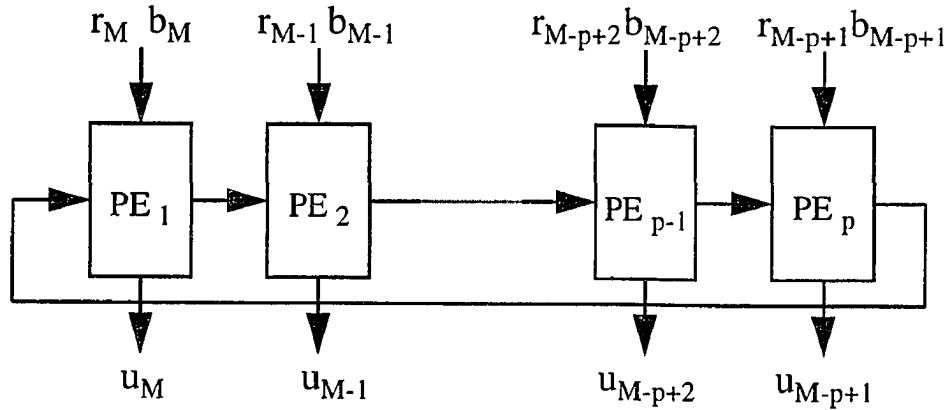


Figure 5.9: Back-Substitution Module

The PEs in the BSM operate similarly to the PEs in the QRM_1 . Each PE receives the assigned row of R and the assigned row of B from the register module1 (RM_1).

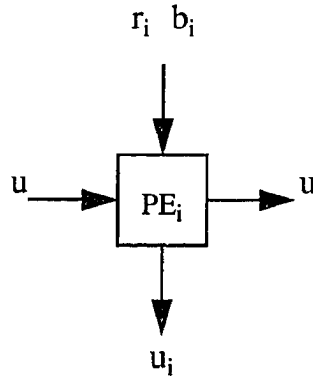


Figure 5.10: A processing element in BSM

Then, it updates the row of B using the row elements of R and the row of U which is received from the previous PE. It computes the row of U using the updated row of R . Finally, it sends the results (row of U) to the next PE for updating the next row of B and sends the results to RM_2 for the next module (QRM_2). The block diagram of the processing element of the BSM is shown in Fig. 5.10.

The internal operations of the PE_i in the BSM are as follows :

1. Receive a row $M - i$ of R and a row $M - i$ of B from NORTH
2. Update a row $M - i$ of B

For $k = M$ down to $M - i + 1$

receive u_k from WEST

$$\mathbf{b}_{M-i} = \mathbf{b}_{M-i} - r_{M-i,k} \mathbf{u}_k$$

send u_k to EAST

end For

3. Compute a row $M - i$ of U

$$\mathbf{u}_{M-i} = \mathbf{b}_{M-i} / r_{M-i, M-i}$$

send \mathbf{b}_k ($k = M, M - 1, \dots, i$) to EAST

4. Send \mathbf{b}_{M-i} to SOUTH

Passing the rows of U (\mathbf{u}_k s) is the only interprocessor communications.

Table 5.14 shows the simulation results for the BSM. In the simulation, R is a 64×64 upper triangular matrix and B is a 64×4 matrix. We solved for a 64×4 matrix U using the back-substitution method. Each PE computes a row of R (in this application, 4 elements) simultaneously. We set the number of PEs to 2 in the simulation.

algorithm	BSM
matrix dimensions	68×64
Number of matrices	50
number of PEs	2
run time	13978626 cycles
real-time rate	0.0155666
avg. utilization	93.9478 %

Table 5.14: Simulation result for BSM with 2 PEs

In Table 5.14, the matrix dimension (68×64) means that a PE receives a row of R (64×1) and a row of B (4×1) as an input data block. In the simulation, 50 consecutive matrices (50 R s and 50 B s) were input and processed. Table. 5.14 shows that the BSM module has good utilization (93.9478 %) of the processing elements with a real-time rate of 0.0155666. Also, we can increase the real-time rate by increasing the number of PEs (see Table 5.15).

algorithm		BSM		
matrix size		68 × 64		
number of matrices		50		
N	T(N)	real-time rate	speedup	avg. utilization
	cycles			%
4	7123441	0.0305470	3.9247	92.1792
6	4710354	0.0461961	5.9354	91.9351
8	3697932	0.0588437	7.5603	88.7846
10	2977015	0.0730933	9.3918	88.2282
16	1985493	0.1095950	14.0808	82.6809

Table 5.15: Simulation results for BSM with different number of processing elements

Table 5.15 shows that the real-time rate is increased to 0.1095950 as the number of PEs is increased to 16. Fig. 5.11 shows the speedup of the BSM. It shows that the speedup is almost linear with increasing the number of PEs. This module can

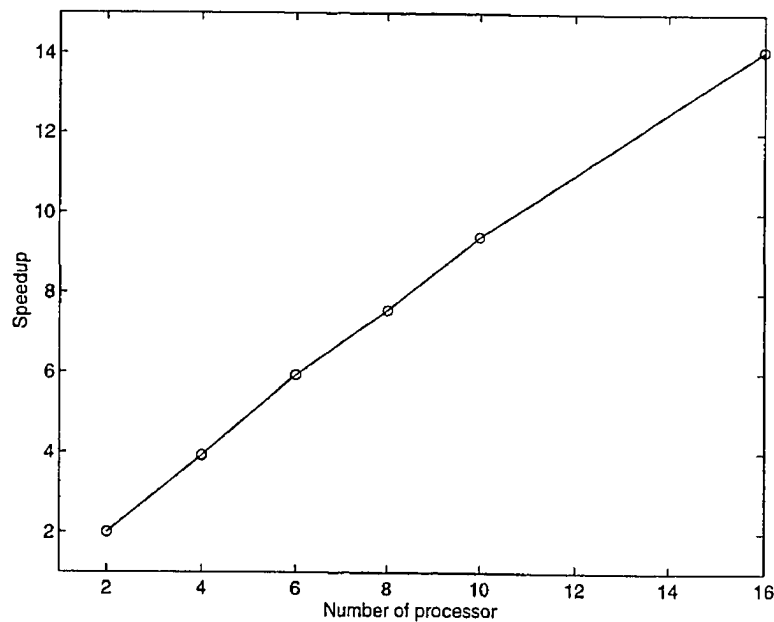


Figure 5.11: The speedup of BSM

work very efficiently with changes in the input matrix size and/or the number of

eigenvalues to be solved. Table 5.16 shows the varying of the average utilization of PEs with varying of the input matrix size. Our results show that the utilization is

matrix size	50×50	62×62	63×63	64×64	65×65	70×70
avg. utilization (%)	93.2548	93.4789	93.8255	92.1792	92.8446	93.5957

Table 5.16: The utilization comparison for different matrix sizes. The number of eigenvalues to be solved is assumed to be 4. Four PEs are used.

always high and does not vary much as a function of the size of the input matrix.

Table 5.17 shows the simulation results when the number of eigenvalues to be solved varies. In the simulation, we assume that the number of eigenvalues required for the partial solution varies from 1 to 6 for the 64×64 matrix. Table 5.17 shows

number of signals	1	2	3	4	5	6
avg. utilization (%)	94.0818	93.1716	92.5320	92.1792	91.8030	91.5734

Table 5.17: The utilization comparison with varying of the number of eigenvalues to be solved. R is a 64×64 matrix and B and U are the $64 \times K$ matrices where K is a number of eigenvalues to be solved. Four PEs are used.

that the average utilization of PEs is not affected much by the varying of the number of eigenvalues required for the partial solution. This is because the changing of the number of eigenvalues to be solved affects the matrix size to be processed and changing the matrix size does not significantly affect the utilization of the system. These simulation results show that the same number of PEs can be used to solve for a different number of eigenvalues with different sizes for the input matrices with high efficiency.

5.5 QR Decomposition Module2

This module computes the QR decomposition of U ($M \times K$). However, only Q is needed in the next computation. The QR decomposition using Given's method creates multiplication pairs and transfers these pairs to the next PE whenever it annihilates a subdiagonal element to obtain an upper triangular matrix R of U ($= QR$). After R is obtained, the Q is computed by the $O(M^2)$ matrix-matrix multiplications (see equation (5.19) in section 5.3 (QRM₁)).

$$Q = Q_{n,n-1}Q_{n,n-2} \cdots Q_{3,1}Q_{2,1} \quad (5.29)$$

This procedure requires lots of computations and data transfers, because it unnecessarily computes matrix R to get the matrix Q . Therefore, the Given's method may not be suitable to find only Q for U , even though this method has good parallelism for finding R for U .

We used the modified Gram-Schmidt method to compute only Q for U . This method computes Q directly without creating the c, s pairs. The procedures are :

For $i = 1$ to K

$$\mathbf{q}_i = \mathbf{u}_i / \text{norm}(\mathbf{u}_i)$$

For $j = i + 1$ to K

$$\mathbf{u}_j = \mathbf{u}_j - \mathbf{q}_i(\mathbf{q}_i^T \mathbf{u}_j)$$

end For

end For

where, \mathbf{u}_i is an i th column of U , and \mathbf{q}_i is an i th column of Q which is the result of

the QR decomposition of $U = QR$. The $\text{norm}(\mathbf{u}_i)$ is defined by

$$\text{norm}(\mathbf{u}_i) = (u_{1i}^2 + u_{2i}^2 + \dots + u_{Mi}^2)^{\frac{1}{2}} = (\mathbf{u}^T \mathbf{u})^{\frac{1}{2}} \quad (5.30)$$

where, u_{ji} is an element of \mathbf{u}_i .

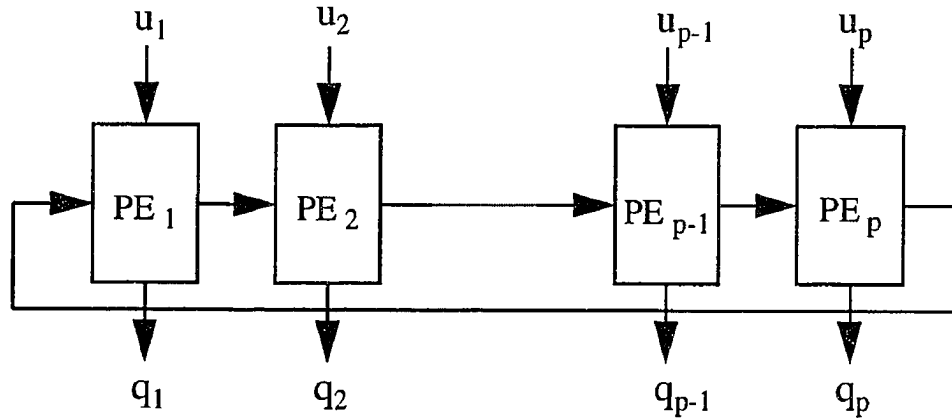


Figure 5.12: QR decomposition Module2

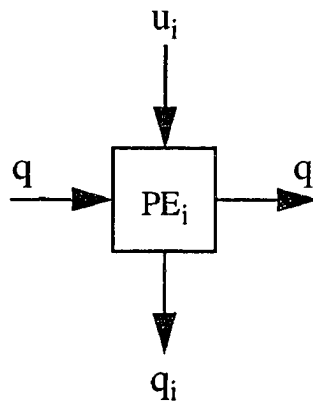


Figure 5.13: A processing element in QRM_2

Fig. 5.12 shows the block diagram for the QR decomposition module2 (QRM₂). In Fig. 5.12, each PE receives a column of U from the register module and the resulting columns of Q from the previous PE. Then, the PE updates the received column of U and computes the corresponding column of Q using the updated column of U . Therefore, the column of Q is the only data transferred in the module. The procedure for the PE is as follows :

1. Receive column i of U (\mathbf{u}_i) from NORTH

2. Update column i

For $j = 1$ to $i - 1$

Receive \mathbf{q}_j from WEST

$$tmp1 = \mathbf{q}_j^T * \mathbf{u}_i$$

end For

$$\mathbf{u}_i = \mathbf{u}_i - \mathbf{q}_j * tmp1$$

3. Compute \mathbf{q}_i

$$tmp2 = \text{norm}(\mathbf{u}_i)$$

$$\mathbf{q}_i = \mathbf{u}_i / tmp2$$

4. Send \mathbf{q}_i to EAST and NORTH

Fig. 5.13 shows a processing element in the QRM₂.

Table 5.18 shows the simulation results for the QRM₂. In the simulation, U was a 64×4 matrix and 4 PEs were used. Each PE received a column of U and computed a column of Q of the QR decomposition of U using the modified Gram-Schmidt method.

Fifty consecutive matrices of U were processed.

algorithm		QRM2		
matrix size		64 × 4		
number of matrices		50		
N	T(N)	real-time rate	speedup	avg. utilization
	cycles			%
2	1001735	0.0127778		84.0039
4	629450	0.0203352	3.1828	83.5192

Table 5.18: Simulation results for QRM₂

Table 5.18 shows that even though the average utilization (83.5192 %) of this method is lower than that for the QR Given's method, it has a good real-time rate (0.0203352) compare to the QRM₁ (0.00284374). This is due to reducing the unnecessary computations and unnecessary data movements.

5.6 Matrix-Matrix Multiplication Module

The matrix-matrix multiplication module (MM) performs the matrix-matrix multiplications. Considering a matrix-matrix multiplication as repeated matrix-vector multiplications is one of the more popular methods.

$$\begin{aligned}
 T &= EQ & (5.31) \\
 &= (Eq_1, \dots, Eq_K) \\
 &= \left(\sum_{i=1}^M q_{i1} \mathbf{e}_i, \dots, \sum_{i=1}^M q_{iK} \mathbf{e}_i \right) \\
 &= (\mathbf{t}_1, \dots, \mathbf{t}_K)
 \end{aligned}$$

Where, E is a $M \times M$ matrix, Q is a $M \times K$ matrix, T is a $M \times K$ matrix, \mathbf{e}_i is a column i of E , \mathbf{q}_i is a column i of Q , \mathbf{t}_i is a column i of T , and q_{ij} is a (i, j) element of Q . The first column of T (\mathbf{t}_1) is computed by

$$\mathbf{t}_1 = q_{11}\mathbf{e}_1 + q_{21}\mathbf{e}_2 + \cdots + q_{M1}\mathbf{e}_M \quad (5.32)$$

In the parallel architecture, an element of Q (q_{ij}) and a column of E (\mathbf{e}_i) are assigned to each PE (PE_i). Therefore, PE_1 computes $tmp = q_{11}\mathbf{e}_1$ and sends tmp to PE_2 , PE_2 computes $tmp = tmp + q_{21}\mathbf{e}_2$ and sends the results to the next processing element and so on until all the columns have been processed. Computing of \mathbf{t}_2 is the same except the elements $q_{11}, q_{21}, \cdots, q_{M1}$ are replaced by the elements $q_{12}, q_{22}, \cdots, q_{M2}$, respectively. Therefore, for the matrix-matrix multiplication (EQ), each PE (PE_i) receives a column of E (\mathbf{e}_i) and a row of Q (\mathbf{q}_i) from the NORTH channel and receives intermediate results of EQ (tmp) from the previous PE (PE_{i-1}). The PE_i computes $tmp = tmp + \mathbf{e}_i * \mathbf{q}_i$ and sends the results to the next PE until all the columns of E and all the rows of Q have been processed. The total data movements for a PE is $2MK+M+K$ while the total number of floating-point operations is MK multiplications and MK additions. Therefore, the rate of floating-point operations to data movements is low in this case.

Since the above algorithm has a low ratio of floating-point operations to data movements, we use the block algorithm [15] [17] to solve this problem. In the block algorithm, the matrix is partitioned into submatrices as follows :

$$T = \begin{bmatrix} E_{11} & \cdots & E_{1L} \\ \vdots & & \vdots \\ E_{L1} & \cdots & E_{LL} \end{bmatrix} \begin{bmatrix} Q_{11} & \cdots & Q_{1P} \\ \vdots & & \vdots \\ Q_{L1} & \cdots & Q_{LP} \end{bmatrix} \quad (5.33)$$

Here, the size of a submatrix is $B \times B$. In the parallel architecture (see Fig. 5.14), E_{11} and Q_{11} are assigned to a PE_1 .

The PE_1 computes $E_{11}Q_{11}$ and sends it to PE_2 and it receives the next assigned submatrix for the next multiplications. As soon as PE_2 receives E_{12} and Q_{21} , it computes $E_{12}Q_{21}$ and adds these to $E_{11}Q_{11}$ and sends the results to the next PE. Therefore, each PE processes B^3 multiplications and $B^2(B - 1) + B^2$ additions while the number of data movements is $4B^2$. Therefore, the rate of floating-point operations to data movements is $2B^3/4B^2$ while the rate for the conventional method is $2MK/2MK + M + K$. The ratio for this rate is

$$ratio = \frac{2B^3/4B^2}{2MK/(2MK + M + K)} = \frac{(2MK + M + K)B}{4MK} \quad (5.34)$$

Equation (5.34) shows that the ratio increase linearly as B increase. The block diagram for the matrix-matrix multiplication module with L PEs is shown in Fig. 5.14.

Table 5.19 shows the simulation results, when a 64×64 matrix is multiplied by a 64×4 matrix with 2 PEs. We set the submatrix size to 4×4 . The simulation results

algorithm	MM
matrix dimensions	64×64 by 64×4
number of matrices	50
number of PEs	2
run time	26151250 cycles
real-time rate	0.0156627
avg. utilization	97.5839 %

Table 5.19: Simulation result for MM with 2 PEs

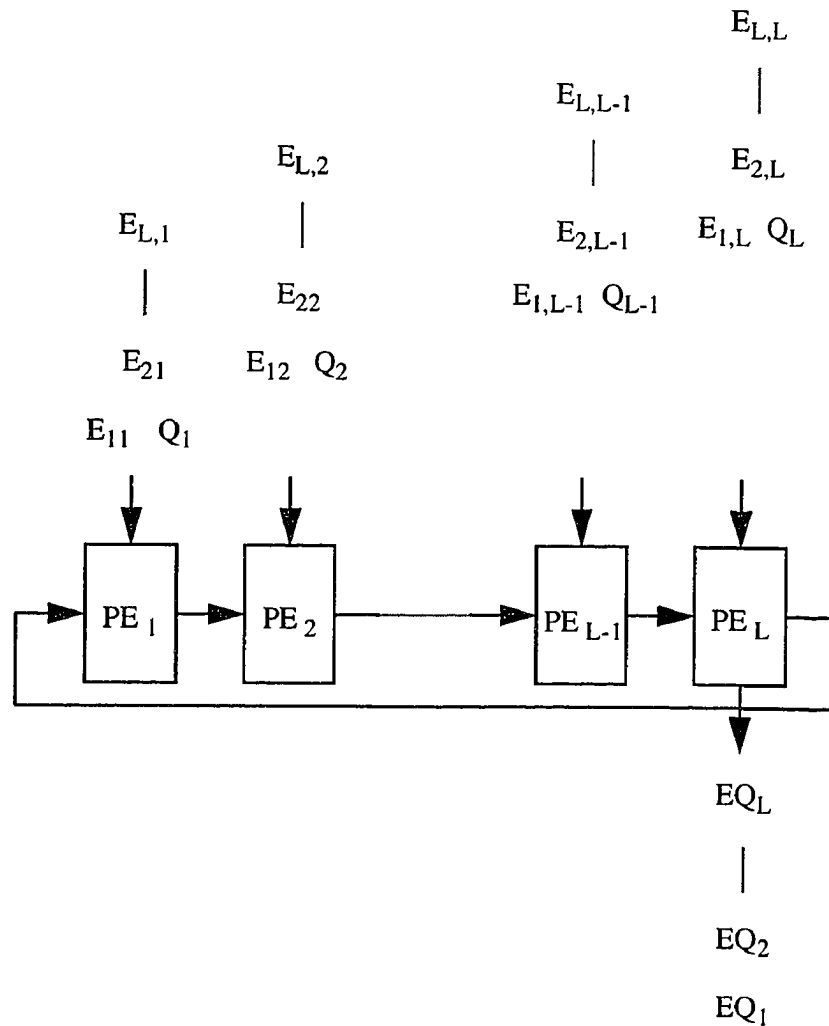


Figure 5.14: Matrix-matrix Multiplication Module

shows that it has good utilization (97.5839 %) of the PEs when 2 PEs are used. Table 5.20 shows the simulation results for MM with different numbers of PEs. With the increasing of the number of PEs (16), this architecture still has a good utilization (96.8444 %, see Table 5.20) which makes the system speedup almost linear (see Fig. 5.15).

algorithm		MM		
matrix size		64 × 64 by 64 × 4		
number of matrices		50		
N	T(N)	real-time rate	speedup	avg. utilization
	cycles			%
4	13113883	0.0312341	3.9883	97.3264
6	8759750	0.0467593	5.9708	97.3975
8	6610683	0.0619603	7.9118	97.1892
10	5310716	0.0771271	9.8485	97.2095
16	3359083	0.1219380	15.5705	96.8444

Table 5.20: Simulation results for MM with different number of processing elements

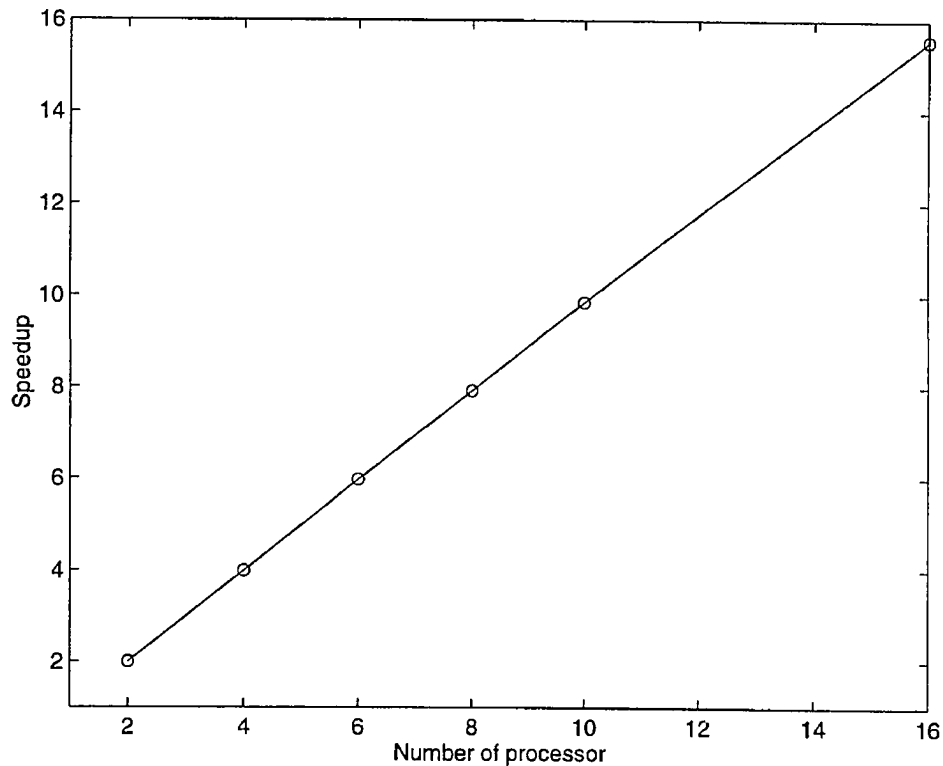


Figure 5.15: The speedup of MM

5.7 $K \times K$ Eigenvalue Solution Module

The eigenvalue solution module (ESM) computes the full eigenvalue solution for the $K \times K$ symmetric matrix H required in the sixth step of the partial eigenvalue solution algorithm. In this applications, K is the number of eigenvalues required for the partial solution. Usually, K is much smaller than M . For example, if we want to compute 4 signal eigenvectors from a 64×64 matrix, then H is a 4×4 matrix. Since the matrix size of H is small, a PE in the ESM performs all the computations required in one iteration.

Several methods exist to compute the eigenvalues and the corresponding eigenvectors of a symmetric matrix [4] [9] [34] [42] [55]. We use the QR-inverse iteration method to compute all of the eigenvalues and the corresponding eigenvectors. We make this choice because the QR decomposition is stable due to using orthogonal transformations, and its convergence rate can be improved by using the origin-shift method [19].

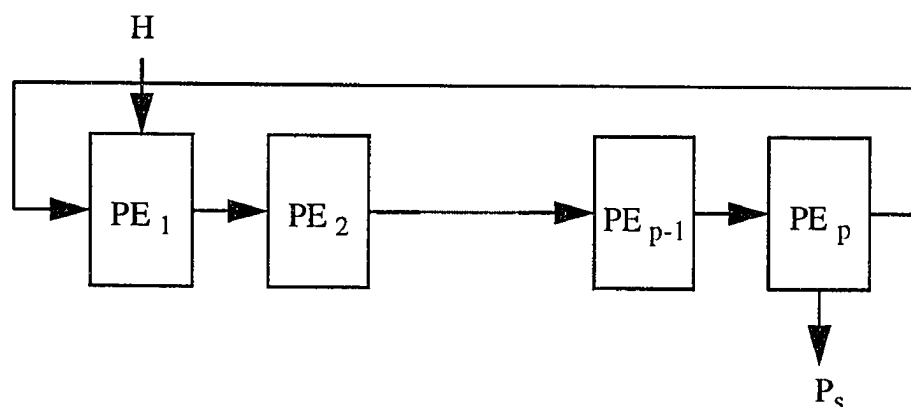


Figure 5.16: Eigenvalue Solution Module

Fig. 5.15 shows the block diagram for the ESM. In the architecture, the first processing element performs the matrix tridiagonalization to reduce the computational tasks for the following PEs. The second through the last PEs perform QR decomposition of the tridiagonal matrix to find the eigenvalues and the corresponding eigenvectors. Only the last PE has the convergence checker. It computes the quantities $conver_chk_i$ as

For $i = 1$ to K

$$conver_chk_i = \frac{|H_{i,i}^i - H_{i,i}^o|}{H_{i,i}^i}$$

where, $H_{i,i}^i$ and $H_{i,i}^o$ represent the input diagonal element of H and the resulting diagonal element of H , respectively. It compare the quantities of $conver_chk_i$ to the allowable tolerance. If it satisfies the convergence condition (*i.e.*, $conver_chk_i$ is smaller than the allowable tolerance), then it sends the output to RM₆ for the next computation. If not, only the last PE performs the computations for the another iteration step. Since this module compute a small matrix compared to the other modules, it can afford the time required for this computation.

Table 5.21 shows the simulation results for the ESM. In the simulation, 4×4 matrices were processed using 8 PEs. The table shows that the ESM has good utilization (94.4041 %) of the PEs with a good real-time rate.

5.8 The Real-Time Rate for the Overall System

In this subsection, we show the real-time rate for the overall system. The real-time rate may be changed by the size of the input matrix and/or the number of eigenvalues to be solved. Table 5.22 shows the real-time rate versus the number of PEs in each

algorithm	ESM
matrix dimensions	4×4
number of matrices	50
number of PEs	8
real-time rate	0.0231528
run time	276425 cycles
avg. utilization	94.4041 %

Table 5.21: Simulation result for ESM with 8 PEs

module when the modules are processed to extract the 4 smallest eigenvalues and the corresponding eigenvectors from a 64×64 matrix.

module	number of PEs	real-time rate
QRM ₁	22	0.0142112
BSM	2	0.0155666
QRM ₂	4	0.0203352
MM ₁	2	0.0156627
MM ₂	2	0.0156627
ESM	2	0.0061663
MM ₃	2	0.0156627

Table 5.22: Real-time rate to extract the 4 smallest eigenvalues and the corresponding eigenvectors from a 64×64 matrix

In the overall system, QRM₁ receives the block of input data and sends the resulting block to the buffers in register module1 (RM₁). It then receives the next input data and performs the computations for the next block of data. The BSM receives data from the buffer in RM₁ and performs the back-substitution. Therefore, the BSM has to work faster than QRM₁ to avoid becoming a bottleneck for the whole system. Since the data transfer between modules is performed asynchronously, the only re-

quirement on the speed of the modules is that each of the modules must perform at a higher rate than QRM_1 .

From Table 5.22, the real-time rate for ESM is 0.0061663 which means the average clock cycles required to perform all the processing for an element is 162.2 ($= 1/0.0061663$) cycles. The input matrix size of ESM is 4×4 in this application. Therefore, the total clock cycles to perform all of the computations for this 4×4 matrix is 2659 (16×162.2) cycles. In the QRM_1 , the average number of clock cycles required to perform all of the processing for an element is 70.4 ($= 1/0.0142112$) cycles. The QRM_1 has to do the computations for all of the elements for a 68×64 matrix and it takes 306240 ($68 \times 64 \times 70.4$) clock cycles to perform these computations. Therefore, the ESM can work faster than the QRM_1 even though the real-time rate for the ESM is smaller than that for the QRM_1 . In this applications, the ESM can perform the iterations up to 115 times without stopping the system.

Table 5.22 shows that the overall system can processes a 64×64 matrix (number of eigenvalues to be solved is 4) with the real-time rate of 0.0142112 using 36 PEs. Since it takes average 70.4 ($= 1/0.0142112$) clock cycles to process one element of a 68×64 matrix, the total clock cycles required to process a whole matrix is 306240 ($= 68 \times 64 \times 70.4$) clock cycles. Therefore, this system computes this application (matrix size is 64×64 , and 4 eigenvalues are extracted) approximately 163 times ($= 50 \times 10^6 / 306240$) per a second using a 50 MHz processors (such as TMS320C40) after filling the pipeline of the modules.

6

Summary and Future Research

6.1 Summary

In this dissertation, we propose a modified eigenvector method (MEVM) which not only uses the best weighting for the eigenvectors of the covariance matrix but also enhances the estimate of the covariance matrix to obtain the best spectral estimate for the direction-of-arrival problem. We evaluated the performance of the MEVM and compared the results with the conventional EVM method. By comparing the perturbation angle of the noise-subspace, we show that the spectral estimate obtained using the proposed method is less distorted than the spectral estimate obtained using the conventional covariance matrix method (EVM) under the same noise conditions. Furthermore, the MEVM has better resolution and accuracy than the conventional method for sources with a small difference in the direction-of-arrival. The superiority of the MEVM was shown by our simulation results.

We implemented a partial eigenvalue solution algorithm using the BDPA to prevent this from becoming the bottleneck for the proposed system. We showed that the parallel architecture for the partial eigenvalue solution module was very flexible in the sense that many high performance DSP algorithms can efficiently be mapped

onto it with promising results. Our simulation results for QRM_1 , BSM, QRM_2 , MM, and EVM have all shown that the architecture performs well with varying the size of the input matrix and the number of eigenvalues to be solved.

The results obtained in chapter 5 show that our approach increases the ratio of floating-point operations to data movements significantly over that for the systolic array implementations to balance the computations and data transfers. Also, the architecture is programmable and can perform the operations with as few as two processing elements (if real-time output is not a concern), and with more processing elements for faster performance with the speedup directly related to the number of processing elements used.

As far as we know, this parallel architecture is the first to implement the partial eigenvalue solution algorithm with the number of eigenvalues required for the partial solution varying with time.

6.2 Future Research

Many signal processing and matrix operation algorithms need to be mapped onto this architecture to truly show its computational power. Also, the performance of this architecture on the mapped algorithms needs to be compared to comparable architectures such as the hypercube, systolic array, and wavefront array.

Next, since the architecture is very flexible and has programmable processing elements, a library for each module can be developed. The modules can be used to implement digital signal processing algorithms and the matrix operations on this architecture. Also, the high efficiency and linear speedup obtained from this archi-

tecture along with its versatility in handling multiple sized problems will provide the necessary supporting justification for this development.

Bibliography

- [1] T. Agrewala and Arvind. Data flow systems. *IEEE Trans. on Computer*, C-36:10–13, 1982.
- [2] S. T. Alexandre, W. E. Alexander, and D. S. Reeves. An architecture and a simulator for parallel signal processing. In *IEEE Southeastcon. '93 Proceedings*, 1993.
- [3] K. J. Bathe and E. L. Wilson. Solution methods for eigenvalue problems in structural mechanics. *International Journal for Numerical Methods in Engineering*, 6:213–226, 1973.
- [4] H. Bowdler, R. S. Martin, C. Reinsch, and J. H. Wilkinson. The QR and QL Algorithms for Symmetric Matrices. *Numer. Math.*, 11:293–306, 1968.
- [5] R. P. Brent and F. T. Luk. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. STAT. COMPUT*, 6(1):69–84, Jan. 1985.
- [6] C. L. Byrne and A. K. Steele. Stable nonlinear methods for sensor array processing. *IEEE Journal of Oceanic Engineering*, OE-10(3):255–259, July 1985.
- [7] M. J. Chen and K. Yao. Systolic Kalman filtering based on QR decomposition. *Proc. SPIE, Advanced Algorithms and Architectures for Signal Processing*, 826:25–32, 1987.
- [8] C. E. Davila and H. Chiang. An algorithm for pole-zero system model order estimation. *IEEE Trans. on Signal Processing*, 43(4):1013–1017, Apr. 1995.
- [9] T. J. Dekker and J. F. Traub. The Shifted QR Algorithm for Hermitian matrices. *Lin. Alg. and Its Applic.*, 4:137–154, 1971.

- [10] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computer*. SIAM, Philadelphia, PA, 1993.
- [11] W. Du and R. L. Kirlin. Enhancement of Covariance Matrix for Array Processing. *IEEE Trans. on Signal Processing*, 40(10):2602–2606, Oct. 1992.
- [12] D. E. Dudgeon and R. M. Mersereau. *Multidimensional digital signal processing*. Prentice Hall, Englewood Cliffs, NJ, 1984.
- [13] M. J. Flynn. Very high-speed computing system. *Proc. of IEEE*, 54:1901–1909, 1966.
- [14] Z. Fu and E. M. Dowling. Conjugate Gradient Eigenstructure Tracking for Adaptive Spectral Estimation. *IEEE Trans. on Signal Processing*, 43(5):1151–1160, May 1995.
- [15] G. Golub and J. M. Ortega. *Scientific computing an introduction with parallel computing*. Academic Press Inc, San Diego, CA, 1993.
- [16] G. H. Golub. Numerical methods for solving linear least squares problems. *Numer. Math.*, 7:206–216, 1965.
- [17] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, MD, 1993.
- [18] S. Haykin, J. H. Justice, N. L. Owsley, J. L. Yen, and A. C. Kak. *Array signal processing*. Prentice Hall, Englewood Cliffs, NJ, 1985.
- [19] C. P. Huang. On the Convergence of the QR Algorithm with Origin Shifts for Normal Matrices. *IMA J. Num. Anal.*, 1:127–133, 1981.
- [20] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing*. McGraw Hill, New York, 1978.
- [21] Texas Instruments. *TMS320C4x User's Guide*. Texas Instruments, May 1991.
- [22] A. Jennings. *Matrix Computation for Engineers and Scientists*. John Wiley and Sons Ltd., 1977.
- [23] A. Jennings and W. J. Stewart. Simultaneous iteration for partial eigensolution for real matrices. *J. Inst. Maths. Applics.*, 15:351–361, 1975.

- [24] D. H. Johnson and D. E. Dudgeon. *Array signal processing*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [25] Don H. Johnson and S. R. DeGraaf. Improving the resolution of bearing in passive sonar array by eigenvalue analysis. *IEEE Trans. on Antennas and Propagation*, ASSP-30(4):638–647, Aug. 1982.
- [26] H. T. Kung. Why systolic architectures ? *IEEE, Computer*, 15, Jan. 1982.
- [27] H. T. Kung and W. M. Gentleman. Matrix triangularization by systolic arrays. *Proc. SPIE, Real Time Signal Processing.*, 298:19–26, 1983.
- [28] S. Y. Kung. *Wavefront array processors*. Maecel Dekker Inc., New York, 1987.
- [29] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, NJ, 1988.
- [30] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer, and D. V. B. Rao. Wavefront array processor: Language, architecture, and applications. *IEEE Trans. on Computer*, C-31:1054–1066, 1982.
- [31] S. Y. Kung and R. J. Gal-Ezer. Synchronous vs. asynchronous computation in VLSI array processors. In *In Proc. SPIE Conference*, Arlington, VA, 1982.
- [32] C. S. Lee and R. J. Evans. A new eigenvector weighting method for stable high resolution array processing. *IEEE Trans. on Signal Processing*, 40(4):999–1004, 1992.
- [33] B. Lint and T. Agerwala. Communication issues in the design and analysis of parallel algorithms. *IEEE Trans. Software Eng.*, SE-7:174–188, 1981.
- [34] S. Lo, B. Philippe, and A. Sameh. A Multiprocessor Algorithm for the Symmetric Tridiagonal Eigenvalue Problem. *SIAM J. Sci. and Stat. Comp.*, 8:s155–s165, 1987.
- [35] M. Lu. A Toeplitz-Induced Mapping Technique in Sensor Array Processing. *IEEE Trans. on Signal Processing*, 43(5):1128–1139, May 1995.
- [36] W. Lu, H. Wang, and A. Antoniou. Design of Two-Dimensional FIR Digital Filters by Using the Singular-Value Decomposition. *IEEE Trans. on Circuits and Systems*, 37(1):35–45, Jan. 1990.
- [37] F. T. Luk. A triangular processor array for computing the singular value decomposition. *Lin. Alg. Appl.*, 77:259–273, 1986.

- [38] C. L. Nikias M. İ. Gürelli. An Eigenvector-Based Algorithm for Multichannel Blind Deconvolution of Input Colored Signals. *IEEE Trans. on Signal Processing*, 43(1):134–149, Jan. 1995.
- [39] A. Moghaddamjoo. Application of Spatial Filters to DOA Estimation of Coherent Sources. *IEEE Trans. on Signal Processing*, 39(1):221–225, Jan. 1991.
- [40] M. Moonen and J. Vandewalle. A systolic array for recursive least squares computations. *IEEE Trans. on Signal Processing*, 41(2):906–911, Feb. 1993.
- [41] C. T. Pan and R. J. Plemmons. Least squares modifications with inverse factorizations: parallel implications. *J. Computational Appl. Math*, 27(1-2):109–127, 1989.
- [42] B. N. Parlett. *The symmetric eigenvalue problem*. Prentice Hall, Englewood Cliffs, NJ, 1980.
- [43] W. Phillips and W. Robertson. A systolic architecture for the symmetric tridiagonal eigenvalue problem. In *In Proc. Int. Conf. Systolic Arrays*, pages 145–150, 1988.
- [44] W. K. Pratt. *Digital Image Processing*. Wiley-Interscience, New York, 1978.
- [45] K. J. Raghunath and V. U. Reddy. Finite data performance analysis of MVDR beamformer with and without spatial smoothing. *IEEE Trans. on Signal Processing*, 40(11):2726–2736, Nov. 1992.
- [46] S. Rajendran and M. V. Narasimhan. An accelerated subspace iteration method. *International Journal for Numerical Methods in Engineering*, 37:141–153, 1994.
- [47] B. D. Rao and K. V. S. Hari. Weighted Subspace Methods and Spatial Smoothing : Analysis and Comparison. *IEEE Trans. on Signal Processing*, 41(2):788–803, Feb. 1993.
- [48] W. Robertson and W. J. Phillips. A System of Systolic Modules for the MUSIC Algorithm. *IEEE Trans. on Signal Processing*, 39(11):2524–2534, 1991.
- [49] H. Rutishauser. *Handbook series linear algebra: Simultaneous iteration method for symmetric matrices*. Springer-Verlag, 1970.
- [50] R. O. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation*. PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, Nov. 1981.

- [51] R. O. Schmidt and R. E. Franks. Multiple source DF signal processing : An experimental system. *IEEE Trans. on Antennas and Propagation*, AP-34(3):281–290, 1986.
- [52] J. Speiser and H. Whitehouse. Architectures for real time matrix operations. In *Proc. of 1980 Government Microcircuit Application Conference*, pages 19–21, 1980.
- [53] A. O. Steinhardt. The PDF of adaptive beamforming weights. *IEEE Trans. on Signal Processing*, 39(5):1232–1235, May 1991.
- [54] S. W. Su and A. K. Thakore. Matrix operations on multicomputer system with switchable memory modules and dynamic control. *IEEE Trans. on Computer*, C-36:1467–1484, 1987.
- [55] D. W. Tufts and C. D. Melissinos. Simple, effective computation of principal eigenvectors and their eigenvalues and application to high resolution estimation of frequencies. *IEEE Trans. on Acoust., Speech, Signal Processing*, 5(34), Oct. 1986.
- [56] B. D. Van Veen. Minimum variance beamforming with soft response constraints. *IEEE Trans. on Signal Processing*, 39(9):1964–1972, Sep. 1991.
- [57] Mati Wax. Detection and localization of multiple sources via the stochastic signals model. *IEEE Trans. on Signal Processing*, 39(11):2450–2456, Nov. 1991.
- [58] H. Xu. *BDFA-A Block Data Flow Architecture for Real-Time Signal Processing and Matrix Operations*. PhD thesis, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1991.
- [59] Q. Zhang, K. M. Wong, P. C. Yip, and J. P. Reilly. Statistical analysis of the performance of information theoretic criteria in the detection of the number of signals in array processing. *IEEE Trans. on Acoust., Speech, Signal Processing*, 37(10):1557–1567, Oct. 1989.