

**Development & Implementation of Visual Approach and Parallel
Distributed Architecture for 2-D DFT & UMRT computation**

A THESIS

Submitted by

BHADRAN V.

for the award of the degree of

DOCTOR OF PHILOSOPHY

Under the guidance of

R. GOPIKAKUMARI



DIVISION OF ELECTRONICS ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE & TECHNOLOGY
KOCHI – 682 022, INDIA

NOVEMBER 2009

CERTIFICATE

This is to certify that the thesis entitled “**Development & Implementation of Visual Approach and Parallel Distributed Architecture for 2-D DFT & UMRT computation**” is a bonafide record of the research work carried out by *Bhadran V.* under my supervision and guidance in the *Division of Electronics Engineering, School of Engineering, Cochin University of Science and Technology* and that no part thereof has been presented for the award of any other degree.

Dr. R. Gopikakumari
(*Supervising Guide*)

Head

Division of Electronics Engineering
School of Engineering

Cochin University of Science and Technology

Kochi
10 November 2009

DECLARATION

I hereby declare that the work presented in the thesis entitled **“Development & Implementation of Visual Approach and Parallel Distributed Architecture for 2-D DFT & UMRT computation”** is based on original research work carried out by me under the supervision and guidance of Dr. R. Gopikakumari in the *Division of Electronics Engineering, School of Engineering, Cochin University of Science and Technology* and that no part thereof has been presented for the award of any other degree.

Kochi
10 November 2009

Bhadran V.

ABSTRACT

Transform theory plays a key role in signal/image processing, as well as in other areas, as it allows the processing simple and flexible. The Fourier transform is especially prevalent, partially since it is the eigen function representation of Linear Time Invariant (LTI) systems. Discrete Fourier Transform (DFT) becomes a powerful tool for frequency domain analysis of discrete time signals due to the increasing flexibility and reconfigurability of digital systems. But the conventional DFT computation is time consuming, especially for 2-D applications, as the processing is done in the complex domain. In this thesis, the 2-D DFT computation is visually represented using a set of primitive symbols based on 2×2 data. The computations are mostly real, except at the final stage. The size of the data matrix is assumed to be even. The DFT coefficients are classified. A basic set of DFT coefficients necessary and sufficient to represent the entire signal is identified using the redundancy analysis. An algorithm is developed for the computation of 2-D DFT by visual approach. Two approaches namely visual representation of DFT coefficients based on 2×2 DFT and 2×2 data are used to design and develop Parallel Distributed Architecture for the computation of 2-D DFT. Version I and version II Parallel Distributed Architectures for the computation of 8×8 point DFT are developed based on former approach while M spacing based 2-D DFT computation employs the latter. The comparison results show the powerful performance of the M spacing based method for 2-D DFT computation against the three existing methods namely conventional DFT computation, modified DFT and closed form method as well as the two other methods developed namely the visual approach and modified DFT using basic DFT in terms of speed. Four different algorithms are also designed for the computation of particular solution required for the M spacing based algorithm. Further derived redundancy present in the MRT coefficients is analyzed and eliminated to obtain 2-D UMRT, which require only the same memory space as required for the original image. A suitable placement scheme is developed to place the UMRT coefficients. Three approaches, one of which is a Parallel Distributed Architecture, are designed, developed and compared for the computation

of 2-D UMRT. Finally, the FPGA implementation of the three architectures developed for the computation of 8×8 point UMRT is compared in terms of area and speed. Different schemes for the M spacing based 2-D UMRT computation are simulated and synthesized in FPGA and their performance are also compared.

ACKNOWLEDGEMENTS

First and foremost I would like to express my profound gratitude to my research guide, Dr. R. Gopikakumari, Head, Division of Electronics Engineering, School of Engineering, Cochin University of Science and Technology, under whose supervision and guidance I have been able to complete my research. But for her able guidance, encouragement and inspiration, this would have never materialized. I wish to express my sincere gratitude to her daughter and mother for the patience in supporting Dr. R. Gopikakumari to spent long hours in the lab outside normal working hours.

I wish to thank the Vice Chancellor and Registrar, CUSAT for the opportunity to complete the research work and submit the thesis. I also thank the Principal, School of Engineering, and members of the Research Committee for guidance and help at various stages of the period of research. I thank the Doctoral Committee members Dr. P. Mythli and Dr. S. Mridula for the support and help throughout the span of research. I wish to express gratitude to the office staff of various sections of CUSAT for the help and assistance. Thanks are also due to the faculty, Division of Electronics Engineering, School of Engineering, and office staff at School of Engineering for all help and assistance provided in relation with the research work. I would like to thank the Director, IHRD for providing me study leave to complete the research work.

I would like to convey my appreciation to Mr. Renjith R., Technical Assistant, Department of Computer Science, his wife Mrs. Bindu, Mr. Rajesh, LBS center for Science and technology, Mr. Jyothish and Mr. Binesh for the timely help. Special thanks are due to Mrs. Deepa, Lecturer, College of Engineering, Thiruvananthapuram, Mr. Pradeep M., Mr. Manilal D., Mr. Shanavaz K. T. and Mr. Sunilkumar K. research scholars for the valuable suggestions.

I wish to acknowledge the blessings of all my teachers from elementary level. I acknowledge the valuable discussions with and support given by Mr. Rajesh Cherian Roy, research scholar at Division of Electronics Engineering, School of Engineering. Special thanks are due to Mrs. Rekha K. James, Dr. Sahana, Mr. Unni and Mr. Sasigopal for the valuable suggestions, encouragement and timely help, offered me during the period of the research work. I take the opportunity to disclose my sincere thanks to all my colleagues, research scholars, students and friends who helped me directly or indirectly in carrying out the research work to fruitful level.

I express my profound gratitude to my wife Mrs. M. Jayasree, for the timely help, patience and support. I wish to convey my appreciation to my two sons Master Vishnu and Master Nandu for the patience and support during the years when I was pursuing my studies. I wish to acknowledge my parents whose blessings have guided me throughout this period. I wish to thank my father in law, mother in law, sisters and brother in laws for the faith and patience during the phase of research.

I am indebted to one and all, who have enlightened me in this crucial period of life.

Bhadran V.

SYMBOLS AND DEFINITIONS

Z	-	Set of integers
$q \in Z$	-	q is an element of set Z
$a, b, h, i, j, l, n, q, r, s, t, v, z, \alpha$	-	Integer variables
n_1	-	Signal spatial index in the vertical direction
n_2	-	Signal spatial index in the horizontal direction
N	-	Integer, $N \in Z$, mostly used to indicate size of input signal
M	=	$N/2$
x_{n_1, n_2}	-	2-D signal sample at n_1, n_2
k_1	-	Frequency index in the vertical direction
k_2	-	Frequency index in the horizontal direction
p	-	Phase index
Y_{k_1, k_2}^p	-	2-D MRT coefficient at k_1, k_2, p .
W_N	=	$e^{-j2\pi/N}$, Twiddle factor
Y_{k_1, k_2}	-	2-D DFT coefficient at k_1, k_2 .
$\text{gcd}(a, b)$	-	Greatest Common Divisor (GCD) of integers a and b
dm	=	$\text{gcd}(k_1, k_2, M)$, is a divisor of M
$a b$	-	a divides b
$a \nmid b$	-	a does not divide b
$a^\alpha b$	-	a^α is the highest power of a dividing b
$((a))_b$	-	Remainder of a/b (Modulo operation)
$\varphi(N)$	-	Euler Totient function of N
$\forall q$	-	For all values of q
\Rightarrow	-	Such that
$ A $	-	Cardinality of the set A
m_c	-	Number of complex multiplications for a DFT coefficient
a_c	-	Number of complex additions for a DFT coefficient
a_r	-	Number of real additions for a DFT coefficient
n_p	-	Number of Y_{k_1, k_2}^p corresponding to a DFT coefficient
nb	-	Number of basic DFT coefficients for N
nb_{dm}	-	Number of basic DFT coefficients where $\text{gcd}(k_1, k_2, M) = dm$

nd_{dm}	-	Number of DFT coefficients that could be derived from a basic DFT coefficient where $\gcd(k_1, k_2, M) = dm$
npt_{dm}	-	Total number of DFT coefficients where $\gcd(k_1, k_2, M) = dm$
dm_o	-	$\gcd(k_1, k_2, M)$ when $\gcd(dm_o, dm) = dm$ and $dm_o > dm$
d_{dm}	-	Divisors of dm other than dm
n_r	-	Number of redundant MRT coefficients for each dm
dm_e	-	$\gcd(k_1, k_2, M)$ where derived redundancy exist in MRT coefficients
nu	-	Number of UMRT coefficients in a basic DFT coefficient
p_i	-	Number of odd prime divisors of M/dm
n	-	Number of odd prime divisors of N
β_i	-	Power of odd prime divisors p_i in the prime factorization of M/dm
α	-	Power of 2 in the prime factorization of M/dm
$nmrt_{dm}$	-	Total number of MRT coefficients corresponding to the basic DFT coefficients where $\gcd(k_1, k_2, M) = dm$, except for $dm = M$
$nmrt_M$	-	Total number of MRT coefficients corresponding to the basic DFT coefficients where $\gcd(k_1, k_2, M) = M$
$nmrt_{nb}$	-	Total number of MRT coefficients for the entire basic DFT coefficients
Tnu	-	Total number of UMRT coefficients corresponding to all the basic DFT coefficients where $\gcd(k_1, k_2, M) = dm$
$A_{R(UMRT)}$	-	Total number of real additions to compute 2-D UMRT, for N

LIST OF TABLES

Table 3.1: Influence of ‘ dm ’ when $N = 16$	51
Table 3.2: Index relation for $N = 20$	55
Table 3.3: nd_{dm} that could be derived from the basic DFT coefficients for different N	56
Table 3.4: nb when $N/2$ is prime	60
Table 3.5: nb when N is a power of 2	60
Table 4.1: Mnemonics used to represent the DFT coefficients for $N = 8$	71
Table 4.2: Number of data points involved in the computation of Y_{k_1, k_2}^p for $N = 4, 6$ & 8 ...	90
Table 4.3: Execution time of different methods for particular solution	110
Table 5.1: Index relation in derived redundancy - $N = 6$	114
Table 5.2: Index relation in derived redundancy - $N/2$ prime	115
Table 5.3: Derived redundancy when $N = 6$	115
Table 5.4: Example of index relation in derived redundancy - $N = 12$	116
Table 5.5: Index relation in derived redundancy - $N = 12$	116
Table 5.6: Derived redundancy for $N = 12$	117
Table 5.7: Example of index relation in derived redundancy - $N = 18$	118
Table 5.8: Index relation in derived redundancy - $N = 18$	118
Table 5.9: Derived redundancy when $N = 18$	119
Table 5.10: Derived redundancy when $N = 8$	119
Table 5.11: nr corresponding to each dm for N , where $((N))_4 = 0$ & N not a power of 2 ..	120
Table 5.12: nr corresponding to each dm for N , where $((N))_4 = 2$ and $N/2$ not prime.	121
Table 5.13: Reduction in computation due to derived redundancy	122
Table 5.14: nb_{dm} and nu for different N	123
Table 5.15: nb for each dm when N power of 2	124
Table 5.16: nu to be computed by each group for different N	127
Table 6.1: Execution time (in sec.) of 2-D DFT computational schemes for N	132
Table 6.2: Comparison of computational complexity of 2-D DFT computation	135
Table 6.3: Execution time (in sec.) of 2-D UMRT computational schemes for N	137
Table 6.4: Computational complexity of 2-D DFT & UMRT for M spacing method	138
Table 6.5: Computation time for 8×8 point DFT by different methods	139
Table 6.6: Computation time for 8×8 point UMRT by different methods	140
Table 6.7: Synthesis results of fully parallel version of 2-D UMRT	141
Table 6.8: Synthesis results of fully parallel, 3 & 4 layer M spacing 8×8 point UMRT .	142
Table 6.9: MCPD for the computation of different combinations of UMRT coefficients	142
Table 6.10: Synthesis results of fully parallel, three layer M spacing 8×8 point UMRT	144
Table 6.11: Synthesis results of 8×8 UMRT chips shown in fig. 6.10 and 6.11.	145

TABLE OF FIGURES

Fig. 3.1: Primitive symbols for visual representation of DFT coefficients using 2×2 data...	46
Fig. 3.2: Visual representation of 2-D DFT coefficients for $N = 4$	46
Fig. 3.3: Visual representation of 2-D DFT coefficients for $N = 6$	46
Fig. 3.4: Visual representation of 2-D DFT coefficients for $N = 8$	47
Fig. 3.5: nb for different N	60
Fig. 3.6: Visuals of Y_{k_1, k_2}^p corresponding to the basic DFT coefficients for $N = 4$	64
Fig. 3.7: Visuals of Y_{k_1, k_2}^p corresponding to the basic DFT coefficients for $N = 6$	64
Fig. 3.8: Visuals of Y_{k_1, k_2}^p corresponding to the basic DFT coefficients for $N = 8$	64
Fig. 3.9: Flow chart depicting the computation of DFT using visual method.....	66
Fig. 4.1: Visual representation of 64 unique set of Y_{k_1, k_2}^p for $N = 8$	70
Fig. 4.2: Version I parallel distributed architecture for 8×8 point DFT computation.....	74
Fig. 4.3: Schematic diagram for version I architecture of 8×8 point DFT.....	76
Fig. 4.4: Version II parallel distributed architecture for 8×8 point DFT.....	85
Fig. 4.5: Patterns seen repeated in Y_{k_1, k_2}^p	91
Fig. 4.6: M spacing based five layer architecture for 8×8 DFT.....	95
Fig. 4.7: M spacing based four layer architecture for 8×8 DFT.....	98
Fig. 4.8: Four layer architecture for $N \times N$ DFT.....	102
Fig. 4.9: Comparison of execution time when employing different particular solution.....	109
Fig. 5.1: Example 1 for derived redundancy for $N = 6$	114
Fig. 5.2: Example 2 for derived redundancy for $N = 6$	114
Fig. 5.3: Example for derived redundancy for $N = 12$	115
Fig. 5.4: Example for derived redundancy for $N = 18$	117
Fig. 5.5: Placement details of 8×8 MRT.....	125
Fig. 5.6: M spacing parallel distributed architecture for $N \times N$ UMRT.....	128
Fig. 6.1: Comparison of execution time of different 2-D DFT computational schemes.....	132
Fig. 6.2: M_R of different 2-D DFT computational scheme for N	134
Fig. 6.3: A_R of different 2-D DFT computational scheme for N	136
Fig. 6.4: Comparison of execution time for different 2-D UMRT computational schemes..	136
Fig. 6.5: Fully parallel 8×8 UMRT.....	140
Fig. 6.6: 8×8 UMRT chip with one data in (parallel implementation).....	143
Fig. 6.7: 8×8 UMRT chip with two data in (parallel implementation).....	143
Fig. 6.8: 8×8 UMRT chip with four data in (parallel implementation).....	143
Fig. 6.9: Block diagram of the 8×8 UMRT chip with four data in.....	144
Fig. 6.10: 8×8 UMRT chip with four Data in (sequential implementation).....	144
Fig. 6.11: 8×8 UMRT chip with two Data in and single UMRT out serially.....	145
Fig. A.1: List of primitive symbols based on 2×2 DFT.....	153
Fig. A.2: Visual representation based on 2×2 DFT for $N = 8$	154
Fig. A.3: Matrix showing the grouping of DFT coefficients for $N = 8$	155

CONTENTS

ABSTRACT	i
ACKNOWLEDGEMENTS	iii
SYMBOLS AND DEFINITIONS	v
LIST OF TABLES	vii
TABLE OF FIGURES	ix
TABLE OF CONTENTS	xi

CHAPTER 1..... 1

INTRODUCTION 1

1.1	DIGITAL SIGNAL PROCESSING	1
1.2	TRANSFORMS	4
1.2.1	Laplace Transform	4
1.2.2	Z Transform	5
1.2.3	Fourier Transform	5
1.2.3.1	One Dimensional DFT (1-D DFT)	6
1.2.3.2	Two Dimensional DFT (2-D DFT)	6
1.2.3.3	Algorithms to implement 2-D DFT	7
1.2.3.3.1	Direct computation	7
1.2.3.3.2	Row-column decomposition	7
1.2.3.3.3	Vector-radix Fast Fourier Transform (FFT)	7
1.2.3.4	Modified DFT computation	8
1.2.4	M-Dimensional Real Transform (MRT)	9
1.2.5	Unique MRT (UMRT) for N power of 2	10
1.2.6	Discrete Cosine Transform (DCT)	10
1.2.7	Wavelet Transform	11
1.2.7.1	Haar Transform	12
1.2.8	Hadamard Transform	13
1.3	IMPLEMENTATIONS	14
1.4	VISUALIZATION	15
1.5	MOTIVATION FOR THE PRESENT WORK	16
1.6	BRIEF SKETCH OF THE PRESENT WORK	18

CHAPTER 2..... 21

REVIEW OF PAST WORK..... 21

2.1	1-D TRANSFORMS	22
2.1.1	DFT	22
2.1.2	DCT	25
2.1.3	Hadamard Transform	25
2.2	TWO-DIMENSIONAL TRANSFORM	26

2.2.1	DFT	26
2.2.2	Modified DFT	27
2.2.3	DCT.....	28
2.2.4	Hadamard Transform	28
2.2.5	Wavelet Transform	29
2.2.6	Haar Transform.....	29
2.2.7	MRT.....	30
2.3	IMPLEMENTATIONS	30
2.3.1	DFT	30
2.3.2	Modified DFT	35
2.3.3	DCT.....	36
2.3.4	Wavelet Transform	37
2.3.5	Haar Transform.....	38
2.3.6	Hadamard Transform	39
2.4	CONCLUSION.....	40
CHAPTER 3.....		41
VISUAL REPRESENTATION AND COMPUTATION OF 2-D DFT		41
3.1	VISUAL REPRESENTATION BASED ON 2×2 DFT	42
3.2	VISUAL REPRESENTATION BASED ON 2×2 DATA	42
3.2.1	Primitive symbols	42
3.2.2	Visual representation	45
3.2.3	Analysis of the visual representation	45
3.2.3.1	Classification of DFT coefficients based on the appearance.....	45
3.2.3.2	Classification of DFT coefficients based on the existence of Y_{k_1, k_2}^p	49
3.2.3.3	Classification of visual representation based on N	51
3.2.4	Redundancy.....	52
3.2.5	Calculation of number of basic DFT coefficients.....	58
3.2.6	Algorithm for computing the index of all basic DFT coefficients.....	61
3.2.7	Patterns in basic DFT coefficients	63
3.3	DFT COMPUTATION USING VISUAL METHOD.....	66
3.4	CONCLUSION.....	68
CHAPTER 4.....		69
PARALLEL DISTRIBUTED ARCHITECTURE FOR $N \times N$ DFT		69
4.1	DEVELOPMENT FOR 8×8 DFT BASED ON 2×2 DFT	70
4.1.1	Hierarchical computation scheme.....	71
4.1.2	Development of Version I architecture.....	72
4.1.2.1	Algorithm.....	75
4.1.2.2	Sample computation.....	80
4.1.3	Version II architecture	84
4.1.3.1	Algorithm.....	84
4.1.4	Comparison of version I & II models with the model for $((N))_4 = 2$	89
4.2	DEVELOPMENT OF M SPACING BASED ARCHITECTURE FOR $N \times N$ DFT	89
4.2.1	Patterns in Y_{k_1, k_2}^p of basic DFT coefficients.....	89
4.2.2	M spacing based data availability	90
4.2.3	Five layer architecture for 8×8 DFT	95
4.2.4	Four layer architecture for 8×8 DFT.....	98
4.2.4.1	Sample computation.....	100

4.2.5	Proposed architecture for $N \times N$ DFT	102
4.2.5.1	Number of additions for each Y_{k_1, k_2}^p in layer 3	103
4.2.5.2	Development of algorithm for layer 3 computation	104
4.2.5.3	Particular solution	105
4.2.5.3.1	Trial and error method	106
4.2.5.3.2	Using extended Euclidean algorithm	106
4.2.5.3.3	Combination of visual approach and other methods	108
4.2.5.3.4	Modified trial and error method	108
4.2.5.3.5	Simulation results	109
4.2.5.4	M spacing based algorithm for any even N	110
4.3	CONCLUSION	112
CHAPTER 5		113
2-D UMRT		113
5.1	DERIVED REDUNDANCY IN MRT	113
5.1.1	Analysis of the MRT coefficients for $N = 6$ ($N/2$ prime)	114
5.1.2	Analysis for $N = 12$ where $((N))_4 = 0$ and N not a power of 2	115
5.1.3	Analysis for $N = 18$ where $((N))_4 = 2$ and $N/2$ not prime	117
5.1.4	Analysis of MRT coefficients when N is a power of 2	119
5.2	COMPUTATION OF REDUNDANT MRT COEFFICIENTS	119
5.3	MRT COEFFICIENTS IN BASIC DFT COEFFICIENTS FOR N , POWER OF 2	123
5.4	SELECTION OF UMRT COEFFICIENTS	124
5.5	PLACEMENT/MATRIX REPRESENTATION OF UMRT COEFFICIENTS	125
5.5.1	Algorithm for placement of a UMRT coefficient	126
5.6	DEVELOPMENT OF ALGORITHMS FOR THE COMPUTATION OF 2-D UMRT	126
5.6.1	Three layer M spacing method	126
5.6.1.1	Algorithm	128
5.6.2	Visual method	130
5.6.3	Modified direct method for UMRT computation	130
5.7	CONCLUSION	130
CHAPTER 6		131
IMPLEMENTATION OF PARALLEL DISTRIBUTED ARCHITECTURE FOR THE COMPUTATION OF 2-D DFT & UMRT		131
6.1	MATLAB SIMULATION	131
6.1.1	2-D DFT	131
6.1.1.1	Computational complexity	133
6.1.2	2-D UMRT	136
6.1.2.1	Computational complexity	137
6.1.3	Parallel distributed architectures and other methods for 8×8 point DFT	139
6.1.4	Parallel distributed architectures and other methods for 8×8 point UMRT	139
6.2	FPGA IMPLEMENTATION OF THE ARCHITECTURES FOR 2-D UMRT	140
6.2.1	Fully parallel implementation of 2-D UMRT	140
6.2.2	Different schemes of M spacing based architecture for 8×8 point UMRT	141
6.2.2.1	Fully parallel implementation of four layer architecture	141
6.2.2.2	Semi parallel implementation	142
6.2.2.3	Parallel distributed architecture with data in/out serially	142
6.2.2.4	Sequential implementation	144
6.2.2	Comparison of different FPGA implementations	145

6.3	CONCLUSION.....	146
CHAPTER 7.....		147
DISCUSSIONS AND CONCLUSIONS.....		147
7.1	VISUAL REPRESENTATION.....	147
7.1.1	The software.....	147
7.1.2	Computation.....	147
7.1.3	Exploitation of redundancy.....	148
7.2	COMPUTATION OF 2-D DFT.....	149
7.3	ARCHITECTURE.....	149
7.4	2-D UMRT.....	150
7.5	FPGA IMPLEMENTATION.....	150
7.6	SUGGESTIONS FOR FURTHER WORK IN THE FIELD.....	151
7.6.1	Visual representation.....	151
7.6.2	Algorithm.....	151
7.6.3	Architecture.....	152
APPENDIX A.....		153
VISUAL REPRESENTATION OF DFT COEFFICIENTS BASED ON 2×2 DFT		153
A.1	PRIMITIVE SYMBOLS AND ITS MNEMONICS.....	153
A.2	VISUAL REPRESENTATION OF 8×8 DFT BASED ON 2×2 DFT.....	154
A.3	MATRIX SHOWING THE GROUPING OF DFT COEFFICIENTS FOR $N = 8$	155
APPENDIX B.....		157
B.1	GREATEST COMMON DIVISOR (GCD).....	157
B.2	LINEAR DIOPHANTINE EQUATION.....	157
B.3	BEZOUT'S LEMMA:.....	157
B.4	THEOREMS ON LINEAR CONGRUENCE.....	157
B.5	PRINCIPLE OF INCLUSION EXCLUSION.....	158
B.6	EUCLIDEAN ALGORITHM, EXTENDED EUCLIDEAN ALGORITHM.....	158
B.7	CO-PRIME INTEGER.....	158
B.8	EULER TOTIENT FUNCTION.....	158
APPENDIX C.....		159
C.1	LAYER 3 COMPUTATIONS FOR $N = 4$	159
C.2	LAYER 3 COMPUTATIONS FOR $N = 6$	159
C.3	LAYER 3 COMPUTATIONS FOR $N = 10$	160
C.4	LAYER 3 COMPUTATIONS FOR $N = 12$	163
REFERENCES.....		173
LIST OF PUBLICATIONS.....		185

An idea that is developed and put into action is more important than an idea that exists only as an idea.

---Buddha

CHAPTER 1

INTRODUCTION

1.1 Digital Signal processing

Digital Signal Processing (DSP) is the processing of signals by digital means. A signal in this context can mean a stream of information representing number of different things. The origin of signal processing is in electrical engineering, and a signal here means an electrical signal carried by a wire or telephone line, or perhaps by a radio wave. Signals contain information about a variety of things and activities in physical world. They can be represented in time domain and frequency domain. In time domain representation, a signal is a time varying quantity. In frequency domain, a signal is represented by its frequency spectrum. Generally, a signal is anything from stock prices to data from a remote-sensing satellite. A digital signal can be processed by performing numerical calculations.

In many cases, the signal of interest is initially in the form of an analog electrical voltage or current, produced for example by a microphone or some other type of transducer. In some situations, such as the output from the readout system of a compact disc (CD) player, the data is already in digital form. An analog signal must be converted into digital form before DSP techniques can be applied. An analog electrical signal can be digitized using an analog-to-digital converter (ADC) [151]. This generates a digital output as a stream of binary numbers whose values represent the electrical input to the device at each sampling instant.

Signals commonly need to be processed in a variety of ways. E.g., the output signal from a transducer may well be contaminated with unwanted electrical "noise" [151]. The electrodes attached to a patient's chest, when an ECG is taken, measure tiny electrical voltage changes due to the activity of the heart and other muscles. The signal is often strongly affected by "mains pickup" due to electrical interference from the mains supply. Processing the signal using a filter circuit can remove or at least reduce the unwanted part of the signal. Increasingly nowadays, the filtering of signals to improve signal quality or to extract important information is done by DSP techniques [165] rather than by analog electronics.

DSP becomes more powerful due to the increasing flexibility and re-configurability of digital systems. DSP technology is nowadays commonplace in such devices as mobile phones, multimedia

computers, video recorders, CD players, hard disc drive controllers and modems, and will soon replace analog circuitry in TV sets and telephones. An important application of DSP is in signal compression and decompression [119]. Signal compression is used in digital cellular phones to allow a greater number of calls to be handled simultaneously within each local "cell". DSP signal compression technology allows people not only to talk to one another but also to see one another on their computer screens, using small video cameras mounted on the computer monitors, with only a conventional telephone line linking them together. In audio CD systems, DSP technology is used to perform complex error detection and correction on the raw data as it is read from the CD.

General-purpose microprocessors [18] such as the Intel x86 family are not ideally suited to the numerically intensive requirements of DSP. The DSP chip [115], a specialized microprocessor with architectures designed specifically for the types of operations required in DSP can carry out such operations incredibly fast, processing hundreds of millions of samples every second, to provide real-time performance. Like a general-purpose microprocessor, programmable digital signal processor (PDSP) is a programmable device, with its own native instruction code. DSP chips are capable of carrying out millions of floating point operations per second, and like their better-known general-purpose cousins, faster and more powerful versions are continually being introduced. DSP chips can also be embedded within complex "system-on-chip" (SoC) [136] devices, often containing both analog and digital circuitry. It has the ability to process a signal "live" as it is sampled and then output the processed signal, for example to a loudspeaker or video display. All of the practical examples of DSP applications mentioned earlier, such as hard disc drives and mobile phones, demand real-time operation.

The major electronics manufacturers have invested heavily in DSP technology. Because they now find application in mass-market products, DSP chips account for a significant proportion of worldwide semiconductor sales, amounting to billions of dollars annually. This trend seems, likely to continue to increase rapidly.

Application specific integrated circuits (ASICs), programmable and field programmable gate arrays (FPGAs) platforms [84] are now mostly used to manufacture DSP chips. ASIC includes logic cells that are customized and all mask layers that are customized. Customizing all of the IC's features in this way allows designers to include analog circuits, optimized memory cells, or mechanical structures on an IC. An FPGA is a semiconductor device containing programmable logic components called "logic blocks", and programmable interconnects. The FPGA platform creates a programmable design chip as against the ASIC platform, where the programme is fixed and cannot be changed once the chip is developed.

FPGAs are a part of Programmable Logic Devices (PLDs) and they facilitate lower time and cost in designing. PLDs are becoming popular while ASICs are becoming more expensive. Deciding between ASICs and FPGAs designers require to answer tough questions concerning costs, tool availability and effectiveness etc. These could include cost (including nonrecurring engineering

charges), die size, time-to-market, tools, performance, intellectual property requirements and reconfigurability. Time-to-market is often at the top of the list. Some large ASICs can take a year or more to design. An option is to do a "rapid ASIC" using preformed ASIC blocks, which saves time and lowers NRE costs. A good way to shorten development time is to make prototypes using FPGAs and then switch to an ASIC. The development cost of chips on an ASIC platform starts from \$5 million and it requires a long time as against for an FPGA platform starts from \$20,000, and can be developed faster than an ASIC. The tools are free on the Web for the smaller FPGAs, while you'll have to pay for a license file for the ones with high gate counts. But there are no NRE charges. Modern FPGAs [183] are packed with features that were not previously available. Today's FPGAs usually come with phase-locked loops, low-voltage differential signal, clock data recovery, more internal routing, high speed (most tools measure timing in picoseconds) hardware multipliers for DSPs [110], memory, programmable I/O, IP cores and microprocessor cores. You can integrate all your digital functions into one part and really have a system on a chip (SoC). A wide range of electronics devices such as mobile phones, digital cameras, base station equipments etc. need SoC. There is no doubt that SoC will drive various applications. Digital devices like cameras use FPGA chips. FPGA-based SoCs have a bright future. Major companies are working on developing the Intellectual Property Cores and these are predefined blocks of various functions. IP cores also perform custom logic operations. Since FPGAs are re-programmable, customers can program the chips to their specifications. Reconfiguration at speeds fast enough to permit run time reconfiguration (RTR) without intolerable overheads is possible [84]. This enables FPGAs to be used as general purpose parallel computing devices [183]. Perhaps this is the major reason as to why FPGAs are not only driving the business, but are also overtaking the ASIC market in costs, start from a couple of dollars to several hundred or more depending on the features, recent times.

Another trend in the DSP hardware design world is the migration from graphical design entries to hardware description language (HDL). Although many DSP algorithms can be described with signal flow graphs, it has been found that code re-use is much higher with HDL-based entries than with graphical design entries. Two HDL languages are popular namely Very High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog® HDL and both seem to be well suited for FPGA. Nowadays there are many tools available in the market for designing FPGA. A set of tools, that were jointly developed by RTI and Virginia Tech, that semi-automates the process of constructing VHDL performance models for DSP applications was described by F.G. Gray [86]. Use of these tools allows rapid evaluation of larger design spaces than was previously feasible. M. Haldar [102] presented the MATCH compiler that takes MATLAB as input and produces hardware in RTL VHDL, which can be mapped to an FPGA using commercial CAD tools. Even though the design time reduces from days to minutes, the generated hardware is slower than the manually designed hardware. P. Banerjee [133] described a behavioral synthesis tool called Accel FPGA, which reads in high-level descriptions of DSP applications written in MATLAB, and automatically generates synthesizable

RTL models and simulation test benches in VHDL or Verilog. The RTL models can be synthesized using commercial logic synthesis tools and place and route tools onto FPGAs. S. Balakrishnan [173] discussed the challenges and requirements of creating portable algorithmic IP for FPGAs and ASICs and illustrates how an ESL synthesis methodology using Synplicity's Synplify DSP tool can significantly reduce the time and effort to implement either technology. The Synplify DSP tool automatically creates optimized logic implementations for both FPGAs and ASICs.

1.2 Transforms

In the field of signal/image processing as well as in other areas, transform theory plays a central role. Transforms are used to find an alternative domain where processing of the task at hand is easier or advantageous to perform. E.g. the convolution in the time domain is equivalent to multiplication in the frequency domain. Transformation of signals also helps in identifying distinct information, which might otherwise be hidden in the original signal. Transforms come in many forms. Linear transforms, especially Fourier and Laplace transforms are widely used to solve problems in science and engineering. Depending on the application, the transformation technique is chosen, and each technique has its advantages and disadvantages.

1.2.1 Laplace Transform

Laplace Transform (LT) is a mathematical tool [123], which provides broader characterization of signals and systems compared to Fourier transform (FT). In some cases LT can be used where FT cannot be used. LT can be used for the analysis of unstable systems where as FT has several limitations. There are several signals for which FT does not converge but the LT converges. An important difference between FT and LT is that FT uses a summation of waves of positive and negative frequencies whereas the LT employs damped waves through the use of an additional factor $e^{-\sigma}$ where, σ a positive number. Both FT and LT convert time domain function $x(t)$ to the frequency domain function $X(e^{j\omega})$ and $X(s)$ respectively. Also the LT provides the total solution to the differential equation and the corresponding initial and final value problems.

For periodic or non periodic time function $x(t)$ which is zero for $t \leq 0$ and deferred for $t > 0$, the LT of $x(t)$ denoted as $L[x(t)]$ may be defined as

$$L[x(t)] = X(s) = \int_0^{\infty} x(t)e^{-st}.dt$$

$$\text{where } s = \sigma + j\omega.$$

The inverse Laplace transform is expressed as

$$X(t) = L^{-1}[x(t)] = \frac{1}{2\pi j} \int_{\sigma_0 - j\omega}^{\sigma_0 + j\omega} X(s)e^{st}.ds.$$

LT uses a transform variable in the complex plane. The transform variable in FT is a pure imaginary number restricted to $s = j\omega$ as can be seen soon.

1.2.2 Z Transform

Z transform [78] plays the same role in the analysis of discrete time signals and LTI systems as the LT plays in the analysis of Continuous time signals and LTI systems i.e., the Z transform is the discrete time counter part of the LT. Z transform of a discrete time signal $x(n)$ may be expressed as

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n},$$

where z is a complex variable.

Inverse Z transform is expressed as

$$x(n) = \frac{1}{2\pi j} \oint_c F(z)z^{n-1}.dz.$$

1.2.3 Fourier Transform

The Fourier transform is used in almost every area of Science and Engineering. When we analyze continuous-time signals with the help of Fourier series and Fourier transforms, the Fourier series and Fourier transforms are called continuous-time Fourier series and continuous time Fourier transform [179] respectively or simply Fourier series and Fourier transform. But when we analyze discrete time signals with the help of Fourier series and Fourier transform, then the Fourier series and Fourier transforms are called discrete-time Fourier series and discrete time Fourier transform respectively.

Fourier series is used to get frequency spectrum of a time domain signal, when the signal is a periodic function of time. With the help of Fourier series, a given periodic function of time may be expressed as the sum of an infinite number of sinusoids whose frequencies are harmonically related. The frequency spectrum of a periodic signal is discrete. Fourier transform is used to get frequency spectrum of a time domain signal when the signal waveform is a non periodic function of time. The Fourier transform provides a continuous frequency spectrum of an arbitrary time domain signal waveform. The continuous time Fourier series exists only when the function $x(t)$ satisfies Dirichlet's condition.

Like continuous Fourier transform discrete time Fourier transform (DTFT) is used for the analysis of discrete time aperiodic signals. DTFT is periodic with period 2π . So any interval of length 2π is sufficient for the complete specification of the spectrum. Generally, the spectrum is drawn in the fundamental interval $(-\pi, \pi)$. Like continuous time FT, the frequency spectrum in DTFT is also continuous in nature. But the frequency spectrum is not periodic in CTFT where as in DTFT the spectrum is periodic with period 2π .

The frequency analysis of discrete time signals is usually performed on a digital computer. Since the representation of the above signal is not a computationally convenient one, we go one step further by sampling its continuous spectrum. This type of frequency domain representation of a signal is known as Discrete Fourier Transform (DFT). The DFT plays a very crucial role in DSP ever since its inception. It is a very powerful tool for frequency analysis of discrete time signals.

DFT is itself a sequence rather than a function of a continuous variable and it correspond to equally spaced frequency samples of DTFT of a signal. Also, FS representation of the periodic sequence corresponds to the DFT of the finite length sequence i.e., by using DFT the discrete time sequence $x(n)$ is transferred into corresponding discrete frequency sequence. DFT offer frequency domain analysis of signals and systems and allows time domain signal processing operations to be performed equivalently in the frequency domain.

1.2.3.1 One Dimensional DFT (1-D DFT)

Let $x(n)$ be a finite duration sequence of length N samples so that $x(n) = 0$ outside the range $0 \leq n \leq N - 1$. Then

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn}, \quad 0 \leq k \leq N - 1 \\ &= 0, \quad \text{otherwise.} \end{aligned} \quad (1.1)$$

where $W_N = e^{-\frac{j2\pi}{N}}$ is called the twiddle factor. Then the corresponding Inverse DFT is expressed as

$$\begin{aligned} x(n) &= \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn}, \quad 0 \leq n \leq N-1 \\ &= 0, \quad \text{otherwise.} \end{aligned} \quad (1.2)$$

1.2.3.2 Two Dimensional DFT (2-D DFT)

There are myriads of image processing and two-dimensional applications where the DFT representation of two-dimensional sequences is of considerable computational importance [63].

Consider a finite duration sequence $x(n_1, n_2)$ of size $N_1 \times N_2$ samples so that $x(n_1, n_2) = 0$ outside the range $0 \leq n_1 \leq N_1 - 1, 0 \leq n_2 \leq N_2 - 1$. Then the Discrete Fourier Transform [21] relations are given by

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2}, \quad 0 \leq k_1 \leq N_1 - 1, 0 \leq k_2 \leq N_2 - 1 \quad (1.3)$$

$$x(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) W_{N_1}^{-n_1 k_1} W_{N_2}^{-n_2 k_2}, \quad 0 \leq n_1 \leq N_1 - 1, 0 \leq n_2 \leq N_2 - 1. \quad (1.4)$$

1.2.3.3 Algorithms to implement 2-D DFT

There are many algorithms for calculating 2-D DFT, which vary, considerably in their computational complexity [21]. However, we shall examine four algorithms in the following sections, to bring out the salient aspects of 2-D DFT computation.

1.2.3.3.1 Direct computation

The direct calculation of 2-D DFT is simply the evaluation of the double sum as in (1.3). If we assume that the complex exponential in equation (1.3) have been pre-computed and stored in a table, then the direct evaluation of one sample of $X(k_1, k_2)$ requires $N_1 N_2$ complex multiplications and a like number of complex additions. Since the entire DFT involves $N_1 N_2$ output samples, the total number of complex multiplications and complex additions needed to evaluate the DFT by direct calculation is $N_1^2 \cdot N_2^2$. Also, the computation time for multiplication will be much more than that of addition.

1.2.3.3.2 Row-column decomposition

The DFT relation can be rewritten as

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} x(n_1, n_2) W_{N_2}^{n_2 k_2} \right] W_{N_1}^{n_1 k_1}$$

If we write $G(n_1, k_2) = \sum_{n_2=0}^{N_2-1} x(n_1, n_2) W_{N_2}^{n_2 k_2}$ then,

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} G(n_1, k_2) \cdot W_{N_1}^{n_1 k_1} .$$

Each column of G is the 1-D DFT of the corresponding column of x . Each row of X is the 1-D DFT of the corresponding row of G . Thus we can compute a 2-D DFT by decomposing it into row and column DFTs. We first compute the DFT of each column of x , put the results into an intermediate array, then compute the DFT of each row of the intermediate array. Alternatively we could do the row DFTs first and the column DFTs second.

If a direct calculation is used to compute the 1-D DFTs in a row-column decomposition, then the evaluation of a 2-D DFT requires, $C_{r/c\ direct} = N_1 \cdot N_2 (N_1 + N_2)$ complex multiplications and additions. If each of N is a power of 2, so that 1-D Fast Fourier Transforms (FFT) can be used, the complex multiplications are further reduced to $C_{r/c\ FFT} = N_1 \cdot N_2 (\log N_1 \cdot N_2) / 2$. The number of complex additions needed is twice this number.

1.2.3.3.3 Vector-radix Fast Fourier Transform (FFT)

The 1-D FFT algorithm achieves its computational efficiency through a 'divide and conquer' strategy. If the DFT length is, for example, a power of 2, the DFT can be expressed in turn as a combination of two quarter-length DFTs and so on [32]. The 2-D vector-radix FFT algorithm is philosophically identical [21]. A 2-D DFT is broken down into successively smaller 2-D DFTs until, ultimately, only trivial 2-D DFTs need be evaluated.

We can derive the decimation-in-time version of the algorithm by expressing an $(N \times N)$ -point DFT in terms of four $N/2 \times N/2$ DFTs (if N is divisible by 2). The DFT summation can be decomposed into four summations: one over those samples of x for which n_1 and n_2 is even, and one for which both n_1 and n_2 are odd. This gives us,

$$X(k_1, k_2) = S_{00}(k_1, k_2) + S_{01}(k_1, k_2)W_N^{k_2} + S_{10}(k_1, k_2)W_N^{k_1} + S_{11}(k_1, k_2)W_N^{k_2+k_1},$$

$$\text{where } S_{00}(k_1, k_2) = \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1, 2m_2)W_N^{2m_1k_1+2m_2k_2}$$

$$S_{01}(k_1, k_2) = \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1, 2m_2+1)W_N^{2m_1k_1+2m_2k_2}$$

$$S_{10}(k_1, k_2) = \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1+1, 2m_2)W_N^{2m_1k_1+2m_2k_2}$$

$$S_{11}(k_1, k_2) = \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1+1, 2m_2+1)W_N^{2m_1k_1+2m_2k_2}.$$

The arrays $S_{00}, S_{01}, S_{10}, S_{11}$ are each periodic in (k_1, k_2) with horizontal and vertical periods $N/2$.

The above said equations expresses the samples of the $N \times N$ DFT, $X(k_1, k_2)$, in terms of four $N/2 \times N/2$ DFTs. By analogy with the corresponding equations from the 1-D case, the computation is represented by a butterfly, or more properly a radix (2×2) butterfly.

Each butterfly requires that three complex multiplications and eight complex additions be performed. To compute all samples of X from $S_{00}, S_{01}, S_{10}, S_{11}$ requires calculation of $N^2/4$ butterflies. This decimation process can be performed $\log_2 N$ times if N is a power of 2. The number of complex multiplications that need to be performed during the computation of a $(N \times N)$ point radix (2×2) FFT is

$$C_{vr(2 \times 2)} = \frac{3}{4} N^2 \log_2 N.$$

The foregoing discussion would have revealed that the 2-D DFT is computationally quite intensive. While research have been progressing on speeding up the 2-D DFT, the transform itself found many applications in image processing, which describes information in two dimensions. Some of the issues in digital image processing is worth considering to stress the diversity of the utility of 2-D DFT.

1.2.3.4 Modified DFT computation

In the modified 2-D DFT [88], the expression for the 2-D DFT computation was restructured by grouping data associated with a given twiddle factor in addition to the exploitation of periodicity and symmetry properties. Then the DFT coefficients Y_{k_1, k_2} , $0 \leq k_1, k_2 \leq N-1$ corresponding to the given data x_{n_1, n_2} , $0 \leq n_1, n_2 \leq N-1$, N even, can be expressed as

$$Y_{k_1, k_2} = \sum_{p=0}^{M-1} Y_{k_1, k_2}^p \cdot W_N^p, \quad 0 \leq k_1, k_2 \leq N-1 \quad (1.5)$$

$$\text{where } Y_{k_1, k_2}^p = \sum_{\forall (n_1, n_2) \Rightarrow z=p} x_{n_1, n_2} - \sum_{\forall (n_1, n_2) \Rightarrow z=p+M} x_{n_1, n_2} \quad (1.6)$$

$$z = ((n_1 k_1 + n_2 k_2))_N \quad (1.7)$$

$$M = \frac{N}{2}. \quad (1.8)$$

Thus the computational complexity is reduced from N^2 complex multiplications for each DFT coefficient required in direct DFT to that of $N/2$.

A pictorial representation for the computation of 2-D DFT in terms of 2×2 point DFT was developed in [88]. This simplifies the computation, as 2×2 DFT involves only real additions. The primitive symbols used for its representation are shown in fig. A.1 and the visual representation of 8×8 point DFT is shown in fig. A.2. All the operations are complex in direct DFT, whereas in visual representation, the computations are mostly real and the scaling by the twiddle factor is done only in the final stage. As a result, the 2-D DFT computation was reduced in terms of real additions and complex multiplication from N^2 to $N/2$ for each coefficient where N is the dimension of the Data matrix and is even.

A hierarchical neural network model was also developed in [88] to implement 2-D DFT for a particular order N such that $((N))_4 = 2$. The model used a parallel distributed scheme of computation in which the processing is in terms of real additions. The only complex operation involved is the scaling by the pre-computed twiddle factors done at the final layer.

1.2.4 M-Dimensional Real Transform (MRT)

In the modified 2-D DFT computation [88], 2-D DFT representation was modified in terms of real additions, which requires $N/2$ complex multiplication in the computation of each of the N^2 DFT coefficients. The complex multiplications can be avoided if the representation of the signal is done in terms of the signal components which would otherwise be multiplied with the exponential term in the DFT representation developed in [88]. A transform named MRT [140] represent 2-D signals in terms of real additions alone rather than using complex multiplications.

MRT coefficients of a data matrix $[x]$ of size $N \times N$ is expressed as

$$Y_{k_1, k_2}^p = \sum_{\forall (n_1, n_2) \Rightarrow z=p} x_{n_1, n_2} - \sum_{\forall (n_1, n_2) \Rightarrow z=p+M} x_{n_1, n_2} \quad (1.9)$$

$$z = ((n_1 \cdot k_1 + n_2 \cdot k_2))_N \quad (1.10)$$

$$M = N/2 \quad (1.11)$$

$$k_1 = 0, 1, \dots, N-1, k_2 = 0, 1, \dots, N-1, p = 0, 1, \dots, M-1.$$

This transform maps the data matrix into M matrices using real additions alone. MRT helps to do the frequency domain analysis of 2-D signals without any complex operations but in terms of real additions.

1.2.5 Unique MRT (UMRT) for N power of 2

The MRT representation has redundant elements, which makes it unsuitable for use in situations where memory usage needs to be minimized. A procedure to obtain a lean MRT or Unique MRT (UMRT) representation of an image is presented in [167] for image size of $N \times N$, where N power of 2. The UMRT coefficients are unique, numerically compact and require only the same memory space as required for the original image. Each MRT coefficient is formed by unique, linear, multiplication-less combinations of image data and thus has spatial significance.

1.2.6 Discrete Cosine Transform (DCT)

Ahmed et al [6] proposed Discrete Cosine Transform (DCT) and based on empirical evidence, conjecture that its performance is closer to the optimal Karhunen-Loeve Transform (KLT) than the other commonly used transforms.

A DCT expresses a sequence of finitely many points in terms of a sum of cosine functions oscillating at different frequencies. The use of cosine rather than sine function is critical in many applications like compression, numerical solutions of partial differential equations, etc. DCT is a Fourier related transform similar to DFT, but using only real numbers. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry. There are eight standard DCT variants, of which four are common namely DCT-I, DCT-II, DCT-III and DCT-IV. The most common is the type-II DCT, which is often called “the DCT” and its inverse, the type-III DCT is the inverse DCT. DCT of an N point signal can be computed using a $2N$ point DFT [6]. A method that employs an N point DFT of a reordered version of the signal (where N is assumed to be even) to compute an N point DCT is presented in [9], resulting in a saving of $\frac{1}{2}$ over the previous method.

The DCT of a list of N real numbers $S(x)$, $x = 0, 1, \dots, N - 1$ is the list of length N given by

$$S(u) = \sqrt{\frac{2}{N}} \cdot C(u) \cdot \sum_{x=0}^{N-1} s(x) \frac{\cos(2x+1) \cdot u \cdot \pi}{2 \cdot N}, \quad u = 0, 1, \dots, N - 1$$

$$\text{where, } C(u) = 2^{-1/2} \text{ for } u = 0$$

$$= 1 \text{ otherwise.}$$

Each element of the transformed list $S(u)$ is the dot product of the input list $S(x)$ and a basis vector. The constant factors are chosen so that the basis vectors are orthogonal and normalized.

DCT is a technique for converting signal into elementary frequency components. The DCT helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). It transforms a signal or image from the spatial domain to the frequency domain. With an input image A the coefficients for the output image B are:

$$B(k_1, k_2) = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} 4 \cdot A(i, j) \cdot \cos\left(\frac{\pi \cdot k_1}{2 \cdot N_1} (2 \cdot i + 1)\right) \cdot \cos\left(\frac{\pi \cdot k_2}{2 \cdot N_2} (2 \cdot j + 1)\right)$$

The input image is N_2 pixels wide by N_1 pixels high; $A(i, j)$ is the intensity of the pixel in row i and column j ; $B(k_1, k_2)$ is the DCT coefficient in row k_1 and column k_2 of the DCT matrix. All DCT multiplications are real. This lowers the number of required multiplications, as compared to the discrete Fourier transform. For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT. The lower right values represent higher frequencies, and are often small enough to be neglected with little visible distortion.

The DCT is by far the most popular transform used for image compression applications [45], [52]. Reasons for its popularity include not only its good performance in terms of energy compaction for typical images but also the availability of several fast algorithms [89]. It has found a wide spectrum of applications in image and video processing and several other signal processing application domains [181]. Joint Photographic Experts Group (JPEG) for compression of still images [45], Moving Picture Experts Group (MPEG) for compression of motion video [44] and International Telegraph and Telephone Consultative Committee (CCITT H.261, also known as $P \times 64$) for compression of video telephony and teleconferencing employ DCT. Its application to image compression was pioneered by Chen and Pratt [19].

1.2.7 Wavelet Transform

The Fourier transform, with its wide range of applications, like many other mathematical tools, has its limitations. For example, this transformation cannot be applied to non-stationary signals. These signals, e.g. speech and image, have different characteristics at different time or space. Although the modified version of the Fourier transform, referred to as short-time (or time-variable) Fourier transform (STFT) can resolve some of the problems associated with non-stationary signals, but does not address all issues of concern. There is only a minor difference between STFT and FT. In STFT, the signal is divided into small enough segments, where these segments of the signal can be assumed to be stationary. For this purpose, a window function 'w' is chosen. The width of this window must be equal to the segment of the signal where its stationarity is valid. In STFT, the window is of finite length, and we no longer have perfect frequency resolution. The STFT is extensively used in speech signal processing but rarely, if ever, used in image processing.

Wavelets were developed independently in the field of mathematics, quantum physics, electrical engineering, and seismic geology [149]. Interchanges between these fields during the last ten years have led to many new wavelet applications such as image compression, turbulence, human vision, radar, and earthquake prediction. The wavelet transform, which was developed independently on different fronts, is gradually substituting the Fourier transform in some essential signal processing applications. Multi resolution signal processing used in computer vision; subband coding, developed for speech and image compression; and wavelet series expansions developed in applied mathematics, have been recognized as different views of a single theory.

Wavelet transform applies to both continuous and discrete signals [156]. This transformation provides a general technique that is applicable to many tasks in signal processing. The wavelet transform is successfully applied to non-stationary signals for analysis & processing and provides an alternative to STFT. In contrast to STFT, which uses a single analysis window, the wavelet transform uses short windows at high frequencies and long windows at low frequencies. This flexibility is introduced in the spirit of so-called constant Q, or constant relative bandwidth frequency analysis. For some applications it is desirable to obtain the wavelet transform as signal decomposition onto a set of basis functions, referred to as wavelets. These basis functions are obtained from a single prototype wavelet by dilations and contractions (scaling) as well as shifts. Recent surge in application of wavelet transform in various areas of signal processing resulted from the effectiveness of this mathematical tool for analysis and synthesis of signals. But as the transforms evolved so did the complexities involved with them. And hence simpler approaches were always welcomed.

1.2.7.1 Haar Transform

The Haar transform [130] is the simplest of the wavelet transform. This transform cross-multiplies a function against a wavelet called Haar wavelet with various shifts and stretches, like the Fourier transform cross-multiplies a function against a sine wave with two phases and many stretches. The Haar transform is derived from the Haar matrix.

The Haar function $h_k(x)$ are defined on a continuous interval, $x \in [0, 1]$, and for $k = 0, 1, \dots, N - 1$, where $N = 2^n$. The integer k can be uniquely decomposed as

$$k = 2^p + q - 1$$

where $0 \leq p \leq n - 1$; $q = 0, 1$ for $p = 0$ and $1 \leq q \leq 2^p$ for $p \neq 0$.

Representing k by (p, q) , the Haar functions are defined as

$$h_0(x) = h_{0,0}(x) = \frac{1}{\sqrt{N}}, \quad x \in [0, 1].$$

$$h_k(x) = h_{p,q}(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2}, & \frac{q-1}{2^p} \leq x < \frac{q-\frac{1}{2}}{2^p} \\ -2^{p/2}, & \frac{q-\frac{1}{2}}{2^p} \leq x < \frac{q}{2^p} \\ 0, & \text{otherwise for } x \in [0, 1] \end{cases}$$

The Haar transform is real and orthogonal. It is a very fast transform and the basis vectors are sequency ordered. The energy compaction for images is poor.

The Haar Transform Matrix

The Haar transform is obtained by letting x take discrete values at m/N , $m = 0, 1, \dots, N - 1$. For example, when $N = 4$, we have

$$H_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

Higher order Haar matrices follow the same structure as the 4×4 matrix. When $N = 8$,

$$H_8 = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}.$$

The Haar transform can be envisioned as a sampling process in which rows of the transform matrix act as samples of finer and finer resolution. The Haar transform provides a transform domain in which energy is concentrated in localized regions.

1.2.8 Hadamard Transform

The elements of the basis vectors of the Hadamard transform [130] take only ± 1 and are therefore well suited for digital signal processing. The Hadamard matrices, H_n , are $N \times N$ matrices, where $N = 2^n$, $n = 1, 2, 3$.

Recursively, the 1×1 Hadamard transform H_0 can be defined as $H_0 = 1$, and then define H_N for $N > 0$ by

$$H_N = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{N-1} & H_{N-1} \\ H_{N-1} & -H_{N-1} \end{bmatrix},$$

where the $1/\sqrt{2}$ is a normalization factor that is sometimes omitted. Thus other than this normalization factor, the Hadamard matrices is made up entirely of 1 and -1. Some examples of the Hadamard matrices are

$$H_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \text{ and}$$

$$H_3 = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Sampling a class of functions called the Walsh functions can also generate the basis vectors of the Hadamard transform. These functions also take only the bipolar values ± 1 and form a complete orthonormal basis for square integrable functions. Due to this, the above transform is also called the Walsh-Hadamard transform. The Hadamard transform is real, symmetric and orthogonal. It is a fast transform and has good energy compaction for highly correlated images.

1.3 Implementations

In the field of signal processing, the development of algorithms and the progress of implementation are closely tied to each other. Programmable processors such as microprocessors or Digital signal processors implement a limited and fixed set of arithmetic and control operations that can be organized and sequenced to implement any transforms. The speed of such processors has steadily increased to match the needs of emerging applications. However the fundamental physical limitation imposed by the speed of light makes it impossible to achieve further improvements in the speed of such processors. Speed of computation, important in real time applications can be improved by parallel processing techniques. In cases where the native processor operations are not well suited to the task at hand or in cases where massive amount of parallelism can be exploited, these processors are inefficient and deliver poor performance [84].

Several implementations of transforms on different parallel computer architectures such as vector processors, parallel processors etc. are available in the literature. Even these approaches will not meet design requirements, in many applications. Either they are too expensive in terms of power and size, or their performance may still be insufficient. The algorithm for such implementations are designed keeping in mind the underlying architecture and hence is not portable. Such implementations cannot be used in mobile equipments [53]. In such cases a custom hardware can often resolve the problem. For real-time very high speed signal processing, a dedicated special purpose hardware processor is often required. Nowadays, ASIC and FPGA platforms are mostly used to manufacture such processors [84].

Selection of proper architecture and implementation styles can strongly influence the performance of dedicated VLSI DSP circuit [53]. E.g., J. Pihl [74] has shown that bit-serial and bit parallel architectures are comparable by area-time measure, but indicate that bit-serial design for high performances are not as efficient from a power consumption point of view. One of several architectures may be chosen, depending on the requirements such as speed, cost, power etc. E.g., in space borne application, efficient implementations are of paramount importance because of power and size constraints. The circuit architecture for high speed applications will continue to evolve in the direction of exploiting parallelism in algorithms using multiple processing elements. Furthermore, keeping architecture and implementation in mind, we can design inherently new signal processing algorithms which are possessed with concurrency.

1.4 Visualization

There is still a huge gap between our ability to extract answers and our ability to present the information in meaningful ways. The explosive generation of massive data sets and our ability to extract data's inherent information has continued to spawn research in several significant areas. The aim of such research is to increase our understanding of the data, what useful information is latent within it, and how to detect portions of it that are of strong interest. The primary relevant fields to this endeavor are statistics, data visualization, databases, and the combinational fields of data mining, pattern recognition, machine learning, and artificial intelligence. In the early days, statistics is used for data analysis by applying various mathematical and numerical methods to determine the fit of mathematical models to the data. The majority of the work in statistics has focused on methods for verifying apriori hypothesis about the data.

"A Picture is worth a thousand words". Different people give different interpretation to a visual seen by them. Visualization technique is a very useful method in discovering patterns present in data sets [93]. The early set of techniques and technologies used to analyze and model data sets, revolved around the visualization of the data using graphs, charts and tables. Digital computers and data storage dramatically changed the picture. The data volumes ruled out traditional 'manual' approaches to analysis.

Visualization, well done, harnesses the perceptual capabilities of humans to provide visual insight into data. Early statistical methods provided reasonable visual support for data explorers. Similarly modern graphical techniques help to provide visual comprehension of the various computational approaches. Moreover, visualizations are also being used to display properties of data that have complex relations – possibly patterns not obtainable by current computation methods.

a) Purpose of data visualization

Human beings look for structure, featuring patterns, trends, anomalies and relationships in data. Visualization supports this by presenting the data in various forms with differing interactions. Visualization can i) provide a qualitative overview of large and complex data sets, ii) summarize data and iii) assist in identifying regions of interest and appropriate parameters for more focused quantitative analysis. In an ideal system, visualization harnesses the perceptual capabilities of the human visual system.

b) Classification of visualization techniques

Visualization can be classified in a number of ways, based on (i) the task at hand, (ii) the structure of the underlying data set, or (iii) the dimension of the display [101].

Visualization can be used i) to explore data, ii) to confirm a hypothesis, or iii) to manipulate a viewer. In an exploratory visualization, the user does not necessarily know what he is looking for. This creates a dynamic scenario in which the interaction is critical. The user is searching for a

structure or trends and is attempting to arrive at some hypothesis. In a confirmatory visualization, the user has a hypothesis that needs to be tested. This scenario is more stable and predictable. System parameters are often predetermined. Analytical tools are especially necessary to be able to confirm or refute the hypothesis. In productive visualization, the user has a validated hypothesis and so knows exactly what is to be presented and focuses on refining the visualization to optimize that presentation. This is the most stable and predictable visualization.

Visualizations can also be classified as to whether (1) the underlying data is spatial or non spatial or (2) the displayed data is to be 2-D or 3-D.

e) Visualization components

Data can be either stable or dynamic. A PET scan for example, is static, where as cloud vapor over time is dynamic. Visualization can be stationary, animated or interactive, respectively, for example, a set of MRI image, a simulation of a finite element analysis over time, or a real time representation of wind flow across an automobile. The processing of data in the visualization system can be batch or interactive; batch for the analysis of a set of images, interactive for pre-surgery analysis.

d) Visualization interaction

The user can interact with the data in a variety of ways. These include (i) browsing, to get the big picture; (ii) sampling, to reduce data size; (iii) directed, for adhoc querying; and (iv) associative, to access related data.

The user can also interact with the visualization system. E.g. the user can create, edit or manipulate the visualization networks; the user can specify data files to be retrieved, data fields to be displayed, specify visualization pipeline parameters, create or manipulate the output (for further queries) or display further available information about the data.

1.5 Motivation for the present work

Transform theory has a prime task in signal processing, as well as in other areas. The transforms find an alternate domain, so that it is easier and convenient to process the task. The predominance of Fourier transform is partially because it represents the eigen function of Linear Time Invariant (LTI) systems [123]. Due to the increasing flexibility and re-configurability of digital systems, DSP is becoming more and more powerful. The DFT is widely used in DSP in various forms so as to exploit the advantages of digital system, especially when J. W. Cooley and J. W. Tukey popularized the idea of FFT by a publication [1]. It is used for frequency domain analysis of signals by mapping the digital data into the frequency domain. Furthermore the convolution theorem also holds in DFT. There are many fast algorithms for computing the DFT, which made it quite popular. But most of them suffer from the complex operations in implementation phase, i.e., in conventional DFT computation algorithms, the data will also be converted to complex form and the computations are carried out in complex form. This increases the computation time and memory requirement, especially in 2-D applications. But most of the world problems have real data. One complex multiplication requires 4

real multiplications and two real additions. Two memory locations will be required to store one complex data. Also, the computation time will be more for multiplication than addition. Thus the speed of computation can be improved by reducing the number of complex multiplications.

In [88], 2-D DFT computation was modified in terms of real additions rather than complex multiplications by grouping data associated with a given twiddle factor in addition to the periodicity and symmetry properties. This modification of the 2-D DFT computation enabled to reduce the computational complexity from N^4 complex multiplications of conventional DFT to $N^3/2$ in the modified DFT. By exploring the features of DFT computation in the real domain, the computational complexity can further be reduced.

From the beginning of science, visual observation has played a major role. Pictorial information provides much more details, than what you explain. Visualization techniques are of increasing importance in exploring and analyzing large amounts of multidimensional data, as the personal computers are powerful enough to process it. A major advantage of visualization is that it allows a direct interaction with the user and provides an immediate feed back, which is difficult to achieve in most non-visual approaches. A visual representation for the computation of 2-D DFT in terms of 2×2 point DFT was developed in [88]. However, the visual representation based on 2×2 DFT does not give a direct relationship between the data and the DFT coefficients. A visual representation based on data may give a better insight into the relationship between time domain data and the frequency domain representation. Analysis of the visual representation can be used to derive simple and efficient computational schemes.

Although purely from a computational efficiency point of view, the DFT is more expensive than the FFT, from a VLSI implementation point of view there are significant reasons why the DFT may be preferable to the FFT (particularly with a small number of coefficients). The FFT's complex data routing limits overall speed and is expensive in terms of chip area [28], [17]. Majority of the available VLSI implementation of 2-D DFT are based on the popular row-column approach, where the 2-D transform is performed in three steps: (i) 1-D transformation of the input rows, followed by (ii) intermediate results transposition (usually implemented with a transposition memory) and (iii) 1-D transformation of the intermediate result columns. A hierarchical neural network model was developed [71], [72] to implement 2-D DFT for a particular order N such that $((N))_4 = 2$. The scheme can be used to compute 2-D DFT with few real multiplications and real additions. The speed of computation will be very high due to the parallel distributed nature, at the same time reducing the memory requirement due to the hierarchical structure. Repetitive nature of the simple modules shows a potential in easy and cost effective VLSI implementation of the 2-D DFT. Fast hardware implementation of the parallel distributed computation of 2-D DFT, for any even N , could be developed which enables 2-D signal processing easy.

MRT [140], [155], as explained in section 1.2.4, represent 2-D signals in terms of real additions alone rather than using complex multiplications. This transform maps the data matrix into M matrices

using real additions alone. MRT, in the raw form contains significant redundancy. In [167] it was shown that the UMRT coefficients are unique, numerically compact and require only the same memory as required for the original image, when N is a power of 2. The visual representation can be analyzed to obtain unique MRT for any even N . Further a fast hardware implementation of a parallel distributed computation of 2-D UMRT, for any even N , could be developed which enables 2-D signal processing easy.

1.6 Brief sketch of the present work

The scheme of the work presented in this thesis is given below:

A review of the important research work done in the field of signal transforms aimed at improving the performance viz. a viz. speed are presented in chapter 2. The emphasis is given to the fast implementations so as to expose the significance of the present work.

The visual representation of 2-D DFT computation using a set of primitive symbols based on 2×2 data are described in chapter 3. Analysis of visual representation is extensively done in section 3.2.3. The classification of DFT coefficients is also explained in section 3.2.3. Different levels of redundancy present in the visual representation of DFT coefficients are analyzed in section 3.2.4. A basic set of DFT coefficients necessary and sufficient to represent the entire signal, identified using the redundancy analysis, is narrated in section 3.2.6. The chapter concludes with the development of an algorithm for the computation of an $N \times N$ point DFT, using visual approach, for any even N .

Chapter 4 gives details of the development of Parallel Distributed Architectures for the computation of 2-D DFT. Version I and II Parallel Distributed Architectures for the computation of 8×8 point DFT, designed by the analysis of the visual representation of DFT coefficients based on 2×2 DFT, are explained in section 4.1.2 and 4.1.3 respectively. M spacing based DFT computation developed, by analyzing the visual representation of DFT coefficients in terms of 2×2 data, is outlined in section 4.2. Four different algorithms for the computation of particular solution required for the M spacing based algorithm are also explained in section 4.2.5.3.

Analysis of derived redundancy present in the MRT coefficients and its elimination to obtain 2-D UMRT is outlined in chapter 5. Analysis of derived redundancy is explained in section 5.1. Section 5.2 narrates the computation of number of MRT coefficients, which are redundant in a basic DFT coefficient. Selection of UMRT coefficients is outlined in section 5.4. A suitable placement scheme developed to place the UMRT coefficients is described in section 5.5. Three approaches, one of which is a Parallel Distributed Architecture, designed and developed for the computation of 2-D UMRT are presented in section 5.6.

The simulation results & comparison of the various algorithms and the parallel distributed architectures developed in chapter 3, 4 & 5 are discussed in chapter 6. Results of the Matlab® simulation of 2-D DFT & UMRT algorithms and their comparisons are summarized in section 6.1. The simulation results of the three architectures for the computation of 8×8 point DFT and UMRT

using Matlab are compared in section 6.1.3 and 6.1.4 respectively. Further, the details of the FPGA implementation of the three architectures for the computation of 8×8 point UMRT is described in section 6.2. Different schemes for the M spacing based 2-D UMRT computation, which are simulated and synthesized in FPGA, are outlined in section 6.2.2 and their performance is also compared in this chapter.

The discussions and conclusions based on the results available from the implementation of various algorithms and architectures are dealt with in chapter 7. Important features of visual representation of DFT coefficients in terms of 2×2 data, presented in chapter 3, are discussed in section 7.1. Section 7.2 discusses the aspects of the 2-D DFT computation using visual approach. Results based on the development of different architectures for 2-D DFT computation are discussed in section 7.3. Salient features of 2-D UMRT algorithms and architectures are discussed in section 7.4. The synthesis results of the FPGA implementations corresponding to different architectures for 2-D UMRT computations are discussed in section 7.5. The chapter also discusses the scope for future research work.

Appendix A provides the primitive symbols used for the visual representation of 2-D DFT coefficients based on 2×2 DFT and the visual representation of 8×8 point DFT. The grouping of the DFT coefficients is shown in fig A.3. Appendix B presented the elements of number theory used for the analysis of visual representation and derivations of the algorithms. Appendix C deals with relevant information in the derivation of the M spacing based algorithm.

CHAPTER 2

REVIEW OF PAST WORK

The Fourier transform provides a continuous frequency spectrum of an arbitrary time domain signal waveform and hence used for frequency domain analysis of the signal ever since its inception in 1807. Due to the increase in flexibility and other numerous advantages of digital circuits over analog circuits, digital signal processing has developed very rapidly. Also, for processing very low frequency signals like seismic signals, EEG signals etc., analog circuits require inductance and capacitance of a very large size whereas digital processing is more suited for such type of applications [150]. The first electronic digital computers were completed in the late 1940's. People started using digital computers extensively for scientific computing. Hence the frequency domain representation of a signal using DFT became popular. Subsequent development of the digital computer over the years has been synonymous with developments in semiconductor technology. From the invention of the transistor at BELL telephone Labs in 1947, the rate of progress has been ever quickening. In 1968, Jack Kilby, then working at Texas Instruments, produced the first 'integrated circuit', which contained just a couple of transistors on a chip of about 1 cm square. According to Moore's law the number of transistors in an IC doubles every 18 months. Today's chips are having more than 2 billion transistors. This pace of development seems to show no sign of halting. But the DFT of an N point sequence require N^2 complex multiplications and $N(N - 1)$ number of complex additions which make it unsuitable for most of the applications. FFT became popular after J. W. Cooley of IBM and John. W. Tukey of Princeton [1] published a paper in 1965 reinventing the algorithm and describing how to perform it conveniently on a computer. Even though FFT is quite efficient for 1-D signals, due to the large number of complex multiplications required, it is not of much use in 2-D signal processing applications. The time domain processing and the use of real transforms in image processing therefore prevailed. E.g., JPEG for compression of still images [45] and MPEG for compression of motion video [44] employ DCT, whereas JPEG 2000 [118] is based on Discrete Wavelet Transform (DWT).

Within the mean time computational aspects of the Fourier transform were further developed to speedup the computation that is demanded for real time applications. To achieve computational speeds for such applications, a hardware implementation is often necessary. An examination of the history of signal processing shows that new algorithms in signal processing result from the need to improve the efficiency and reduce the cost of implementation [80]. New design and implementation techniques are often triggered by signal processing algorithms that tend to be computation intensive. In the field of signal processing, the development of algorithms and the progress of implementation

are closely tied to each other. The literature review comprising of the important work done in the above fields are presented below.

2.1 1-D Transforms

2.1.1 DFT

Various methods are available in the literature to compute the 1-D DFT. Few of them are listed below.

Two methods for computing DFT by polynomial transforms are proposed by H. J. Nussbaumer et al. [11] and shown that these techniques are particularly well adapted to multidimensional DFT's as well as to some 1-D DFT's and yield algorithms that are, in many instances, more efficient than FFT or the Winograd Fourier Transform (WFTA).

A systematic method of sparse matrix factorization developed by Z. Wang [22] for all four versions of the discrete W transform, the DCT, and the discrete sine transform, as well as for the DFT in which only real arithmetic is involved. A scheme for reducing multiplications and a convenient index system are introduced.

C. X. Fan et al. [25] introduced an FFT algorithm using Hadamard transform (HAT), which is called Hadamard Fourier Transform (HFT). In the proposed algorithm, a HAT is used as mid-transform and the redundant calculation in the original FFT algorithm is reduced by double transformation. The results of theoretical analysis show that the number of multiplications and additions of HFT are both decreased by 60% compared with that of traditional FFT and the executed result shows the computing speed of HFT is 1.6 to 1.7 times faster than FFT.

O. K. Ersoy [29] developed a two-stage representation in terms of preprocessing and Post-processing of DFT by vector transformation of sines and cosines into basis functions using Mobius inversion of number theory. The preprocessing matrix, with elements 1, -1, and 0, is obtained by replacing $\cos 2\pi n/N$ and $\sin 2\pi n/N$ by $\mu(n/N + 1/4)$ and $\mu(n/N)$, respectively, where $\mu(\cdot)$ is the bipolar rectangular wave function. The post-processing matrix is block diagonal where each block is a circular correlation and consists of the above basis functions. They claim that the two-stage representation is useful for parallel implementation of DFT.

Basis-vector-decomposition based two-stage computational algorithms for DFT and DHT were proposed by J. L. Wu et al. [54]. The computations of DFT are divided into two stages: an add/subtract preprocessing and a block-diagonal post-processing. Both stages can be computed effectively. They claim that the computational complexity of the proposed DFT algorithm is identical to that of the most popular split radix FFT yet requires real number arithmetic only. Generation and storage of the real multiplicative coefficients are easier than that in conventional FFT's.

Decimation-in-time-frequency (DITF) FFT algorithm is obtained by combining the decimation-in-time (DIT) and the decimation-in-frequency (DIF) FFT algorithms as proposed by A. Saidi [61] which reduces the number of real multiplications and additions. The above algorithm reduces the arithmetic complexity while using the same computational structure as the conventional

Cooley-Tukey (CT) FFT algorithm. The algorithm is extended to radix-R FFT as well as the multidimensional FFT algorithm using the vector-radix FFT.

A 1D-to-1D mapping was designed to get an iterative structure of Winograd FFT algorithm (WFTA) by X. Qingbin et al. [66]. With this representation, the WFTA has the features of in-place computation and unified computational structure as same as Cooley-Tukey algorithm.

Q. H. Liu et al. [87] proposed an accurate algorithm for the non uniform forward FFT (NUFFT) based on a new class of matrices, the regular Fourier matrices for a non-uniformly sampled data. For the non-uniform inverse FFT (NU-IFFT) algorithm, the conjugate-gradient method and the regular FFT algorithm are combined to speed up a matrix inversion.

Y. Jiang et al. [112] presented an FFT algorithm to reduce the frequency of memory access as well as multiplication operations. For an N-point FFT, they designed the FFT with two distinct sections: (1) the first section of the FFT structure computes the butterflies involving twiddle factors W_n^j ($j \neq 0$) through a computation/partitioning scheme similar to the Hoffman coding. In this section, all the butterflies sharing the same twiddle factor will be clustered and computed together. In this way, redundant memory access to load twiddle factors is avoided. (2) In the second section, the remaining (N-1) butterflies involving the twiddle factor W_N^0 are computed with a register-based breadth-first tree traversal algorithm.

G. X. Fan et al. [132] presented a fast algorithm for the evaluation of the Fourier transform of piecewise smooth functions with uniformly or non uniformly sampled data by using a double interpolation procedure combined with the FFT algorithm. This is a discontinuous FFT algorithm. The method also provides a non-uniform FFT algorithm for continuous functions.

Zhong Cui-xiang et al. [148] introduced a general method to deduce FFT algorithms and then transforms the deduced second radix-2 decimation-in-time FFT algorithm into another parallelizable sequential form. Finally the latter algorithm is transformed into a parallel FFT algorithm, reducing the time complexity of DFT to $O(N \log N/p)$ (where p is the number of processors). Using similar methods, other parallel 1-D and 2-D FFT algorithms can be designed.

S. Lee et al. proposed the [161] modified Single-path Delay Feedback (SDF) architecture for FFT implementation, which implements a mixed Decimation-in-Frequency (DIF) /Decimation-in-Time (DIT) FFT algorithm. Since final stage is computed as DIT FFT algorithm and other stages including input stage are computed as DIF FFT algorithm, both input and output data occur in normal order and additional clocks for reordering input or output is not required.

There are many algorithms with different radix, most of which are refined since its development. In terms of implementation, it should be noted that all of these radix algorithms can be performed 'in-place'; that is, at each stage of the algorithm the output data may overwrite the input data and so no storage is required beyond the size of the original data. Furthermore, each algorithm is amenable to simple recursive programming techniques.

An extended split-radix FFT algorithm that has the same asymptotic arithmetic complexity as the conventional split-radix FFT algorithm was proposed by D. Takahashi [104]. This algorithm has the advantage of fewer loads and stores than either the conventional split-radix FFT algorithm or the radix-4 FFT algorithm.

S. Bouguezel et al. [131] proposed an improved radix-16 DIF FFT algorithm by introducing new indices for some of the output sub-sequences resulting from the conventional radix-16 DIF decomposition of the DFT. This improved radix-16 DIF FFT algorithm achieves savings of more than 46% in the number of twiddle factor evaluations or accesses to the lookup table and address generations compared to the conventional radix-16 DIF FFT algorithm.

An alternate method to derive higher radix FFT algorithms by using a recursive approach and by appropriately combining the twiddle factors without increasing the structural complexity was proposed by S. Bouguezel et al. [164]. They have designed efficient radix-8 and radix-16 FFT algorithms and their arithmetic complexities shown to be slightly less than those of the corresponding existing Cooley-Tukey FFT algorithms.

Y.J. Moon et al. [153] developed a Mixed-Radix 4-2 Butterfly Structure with simple bit reversing output sequences derived by index decomposition technique, which was used in the Radix- 2^i algorithm. Compared with the Radix- 2^3 algorithm and the Split-Radix 2/4/8 algorithm, the proposed algorithm has the same number of multipliers and the less number of stages and butterflies than the Radix- 2^3 and the Split-Radix 2/4/8 algorithm. Moreover, the proposed algorithm makes an offer, the simple bit reversing for ordering the output sequences, which is only supported by a fixed-radix FFT algorithm.

S. Bouguezel et al. [174] proposed a general class of split-radix FFT algorithms for computing the length- 2^m DFT by introducing a recursive approach coupled with an efficient method for combining the twiddle factors. This enables the development of higher split-radix FFT algorithms from lower split-radix FFT algorithms without any increase in the arithmetic complexity.

Many algorithms were developed for implementation in different parallel computers and DSP chips.

D. Rodriguez [43] presented an algorithm for computing the DFT and implemented on DSP 96002. This FFT algorithm is obtained through decomposition of the Fourier matrix representing the DFT operator into a product of sparse matrices not all square matrices. The algorithm is based on additive properties of the input and output indexing sets of the Fourier transformation.

High-performance parallel 1-D FFT algorithms, for distributed-memory parallel computers with vector symmetric multiprocessor (SMP) nodes, were proposed by D. Takahashi [95]. In the parallel FFT algorithms implemented in four-step and five-step methods, since cyclic distribution is used, all-to-all communication takes place only once. Moreover, the input data and output data are both in natural order.

The grouped scheme was applied by C. P. Fan et al. [166] to compute the FFT when the portions of transformed outputs are calculated selectively. The grouped FFT algorithm applies the scheme of the grouped frequency indices to accelerate the computation of selected DFT outputs. The advantage of the grouped FFT algorithm is that it is more cost-effective than the conventional FFT algorithms when we need to compute parts of the transformed outputs, not all outputs. By sharing coefficients of the twiddle factors in the same frequency group, the grouped FFT can be implemented with hardware sharing VLSI architectures.

2.1.2 DCT

A discrete cosine transform (DCT) is defined and an algorithm to compute it using the FFT was developed by N. Ahmed et al. [6]. It is shown that the DCT can be used in the area of digital processing for the purposes of pattern recognition and Wiener filtering. Its performance is compared with that of a class of orthogonal transforms and is found to compare closely to that of the KLT, which is known to be optimal. The performances of the KLT and DCT are also found to compare closely with respect to the rate-distortion criterion.

M. J. Narasimha et al. [9] used an N -point DFT algorithm to evaluate a DCT by a simple rearrangement of the input data. This method is about two times faster compared to the conventional method, which uses a $2N$ -point DFT.

S. C. Chan et al. [47] presented efficient methods for mapping odd-length type-II, type-III, and type-IV DCT's to a real-valued DFT. It is found that odd-length type II and type III DCT's can be transformed, by means of an index mapping, to a real-valued DFT of the same length using permutations and sign changes only. The real-valued DFT can then be computed by efficient real-valued FFT algorithms such as the prime factor algorithm. Similar mapping is introduced to convert a type-IV DCT to a real-valued DFT up to a scaling factor and some additions. Methods for computing DCT's with even lengths are also discussed.

2.1.3 Hadamard Transform

Hadamard transforms (HAT) can save additional computer time since only real-number operations are involved. However, the power spectra represent groups of frequencies rather than individual frequencies, and in general, there is loss of phase information.

An algorithm for computing Hadamard transforms was presented by D. Coppersmith et al. [59]. If N is a power of four, then the algorithm uses $7/8 N \log_2 N$ multiply/adds to compute a Hadamard transform of length N .

B. J. Falkowski et al. [70] defined a family of Unified Complex Hadamard Transforms derived from Walsh functions. Different types of Complex Hadamard matrices can be generated from the developed direct matrix operator. Sparse matrix factorization or matrix partitioning of the Complex Hadamard matrices leads to the fast algorithms with complexity $N \log_2 N$ suitable for hardware implementation.

A Generalized Hadamard Transform for multiphase or multilevel signals was introduced by K. J. Horadam [145], which includes the Fourier, Generalized, Discrete Fourier, Walsh-Hadamard and Reverse Jacket Transforms. The jacket construction is formalized and shown to admit tensor product decomposition.

A family of fast Walsh Hadamard transform algorithms that have an identical and iterative stage factorization was presented by P. M. Puig [160]. The transform factorization is in terms of identical sparse matrices that implement the stages of general radix-R factorization, where R is a power of 2.

2.2 Two-dimensional Transform

2.2.1 DFT

G. Bongiovanni et al. [7] showed that if a one-dimensional vector A is fractured into a two-dimensional matrix E, a one-dimensional generalized discrete Fourier transforms (GFT) on A and a two-dimensional GFT on E give the same result and require the same number of operations to be computed. The result holds also for the DFT, as it is a particular case of the GFT.

C. Caraiscos et al. [14] has combined the well-known decimation-in-time and decimation-in-frequency FFT algorithms to give a 2-D DFT algorithm, the Mixed Simultaneous Decimation FFT algorithm.

R. D. Preuss [15] developed and presented a radix-2 FFT algorithm that reduces the number of multiplications to two-thirds of that required by most radix-2 algorithms. Its structure is particularly appealing when transforming pure real or imaginary sequences and/or symmetric or anti symmetric sequences and that the memory requirements other than those for storing the input data do not grow with the size of the transform.

A. Guessoum et al. [26] generalized the prime factor algorithm for the evaluation of a 1-D DFT to the evaluation of M-D DFTs defined on arbitrary periodic sampling lattices. It is shown that such an algorithm is equivalent in computational complexity to the evaluation of a rectangular DFT.

A fast discrete Radon transform (FDRT) algorithm for computing 2-D DFTs, which has the advantage of having the lowest number of multiplications and is more suitable for parallel implementations compared with other related algorithms was presented by D. YANG [36].

D. Yang [38] presented a decomposition in which the 2-D DFT can be converted into a series of the odd DFT using the discrete Radon transform (DRT). Further a Fast DRT based 2-D DFT algorithm is presented which has the advantage of greater regularity suitable for parallel architecture.

Use of Radon transform algorithm for reducing the number of 1-D DFTs to compute a 2-D DFT, are offset by a costly increase in the number of additions required to perform the Radon transform. A common factor 2-D FFT algorithm to compute the 2-D DFT is $O(N^2 \log_2 N)$ computationally, whereas L. M. Napolitano et al. [46] in their approach reduced the number of FFT additions and multiplications by 25%, but adds $O(N^3)$ additions.

The split-radix approach for computing the DFT is extended for the vector-radix FFT to two and higher dimensions by S. C. Chan et al. [51]. It is obtained by further splitting the $(N/2 \times N/2)$ transforms with twiddle factors in the radix- (2×2) FFT algorithm.

S. Bouguezal et al. [91] derived an algorithm for computing multidimensional Cooley-Tukey FFT's that is suitable for implementation on a variety of multiprocessor architectures from a Cooley decimation-in-time algorithm by using an appropriate indexing process and the tensor product properties. It is seen that the number of multiplications necessary to compute the algorithm is significantly reduced while the number of additions remains almost identical to that of conventional Multidimensional FFT's (MFFT).

An original multidimensional FFT algorithm was proposed by R. Bernardini [97], where the computation is first organized into multiplier-free butterflies and then completed by 1-D FFT's. The properties of well-known 1-D FFT algorithms blend in quite nicely with those of the proposed multidimensional FFT scheme, extending their computational and structural characteristics to it.

S. Bouguezal et al. [135] presented an efficient algorithm for pruning the output samples of the radix- (2×2) 2-D DIT algorithm. This is done by grouping, in the radix- (2×2) 2-D DIT FFT algorithm, all the stages that involve unnecessary operations into a single stage and introducing a recursive technique for computing the resulting stage. Due to this grouping and the efficient indexing process, the implementation of the proposed algorithm requires a minimum number of stages there by reduces the overall control and structural complexities.

S. C. Pei et al. [134] presented an efficient split vector-radix-2/8 FFT algorithm which saves 14% real multiplications and has much lower arithmetic complexity than the split vector-radix-2/4 FFT algorithm. The algorithm also reduces 25% data loads and stores compared with the split vector-radix-2/4 FFT algorithm.

2.2.2 Modified DFT

A pictorial Representation for the computation of 6×6 point DFT in terms of 2×2 point DFT was developed by R. Gopikakumari et al. [69]. All the operations are complex in direct DFT, whereas in visual representation, the computations are mostly real and the scaling by the twiddle factor is done only in the final stage. As a result, the 2-D DFT computation was reduced in terms of real additions and complex multiplication from N^2 to $N/2$ for each coefficient where N is the dimension of the Data matrix and is even. The DFT computation and analysis of 2-D signals can be simplified using this visual representation.

R. Gopikakumari et al. [73] proposed a visualization technique for the analysis of 2-D Data based on DFT.

In [83] a visual manipulation of symbols to implement 2-D DFT was developed by R. Gopikakumari et al.

R. Gopikakumari [88] modified 2-D DFT representation in terms of real additions, which requires $N/2$ complex multiplication in the computation of each of the N^2 DFT coefficients. The above restructuring was made possible by grouping data associated with a given twiddle factor in addition to the exploitation of periodicity and symmetry properties.

A fast approach for visual representation of selected DFT for $((N))_4 = 2$ was proposed in [126] by R. Gopikakumari et al. using direct and indirect method. The direct method involves selection of appropriate symbols for each 2×2 DFT from the symbol table after computation, whereas the indirect method was derived after analyzing the properties of visual representation. This method is suitable when a few DFT coefficients need be computed which will help in visualizing the influence of data at different time / spatial index over a particular frequency.

In [125] R. Gopikakumari et al. presented a semantic rule based visual representation of 2-D DFT for $N = 6$. This facilitates the construction of the complete set of DFT coefficients from a very few coefficients, reducing the computational complexity and normal memory requirement of visual representation.

2.2.3 DCT

J. Makhoul [12] showed that the DCT of an N -point real signal may be obtained using only an N -point DFT of a reordered version of the original signal, with a resulting saving of $1/2$. The method is then extended to two dimensions, with a saving of $1/4$ over the traditional method that uses the DFT.

M. Vetterli [22] presented a fast radix-2 2-D DCT by first improving the mapping of a real signal into a 2-D DFT followed by an usual polynomial transform approach to map the 2-D DFT into a reduced size 2-D DFT and 1-D odd DFTs. Odd DFT algorithms is optimized for real signals. There is a reduction in the number of multiplications and additions in comparison to other algorithms.

A 2-D DCT algorithm based on a direct polynomial approach is proposed by P. Duhamel et al [35] which results in a reduction in the number of both multiplications and additions compared to the previous ones.

A fast algorithm for the 2-D DCT computation was derived based on index permutation by Y. M. Huang [81] which require only the computation of N 1-D DCT's of length N samples and some post additions for a data matrix of size N . The associated post addition stage possesses a more regular butterfly structure, which makes it more suitable for VLSI and parallel implementations.

Y. M. Huang et al. [89] presented an index permutation-based fast 2-D DCT algorithm and shown that the $N \times N$ 2-D DCT, where $N = 2^m$, can be computed using only N 1-D DCT's and some post additions.

2.2.4 Hadamard Transform

A modified factorization for the Hadamard matrix was developed by H. Y. L. Mar et al. [5] for obtaining the fast Hadamard transform, which may be interpreted as operations on an H diagram.

A four level Hadamard transform which maintains the row orthogonality as the binary Hadamard transform but requires the use of complex numbers is presented by J. J. Komo et al. [31]. The four level Hadamard transform is more general than the binary Hadamard transform.

A column (row)-wise algorithm for computing the 2-D DHAT of size $2^r \times 2^r$, $r > 1$, by using the paired algorithm of the 1-D DHAT, was presented by A. M. Grigoryan et al. [98]. The discrete paired transforms, which split the 2^r -point DHAT, $r > 1$, into a set of smaller 2^{r-i} point transforms, $i = 1: n$, is introduced.

The Hadamard transform was generalized to the case of lapped transform and many methods have been proposed to construct lapped Hadamard matrices by S. E. Phoong et al. [121].

W. Ouyang et al. [188] proposed a fast algorithm for Walsh Hadamard Transform on sliding windows, which can be used to implement pattern matching efficiently. The computational requirement of the proposed algorithm is about 1.5 additions per projection vector for each sample.

2.2.5 Wavelet Transform

N. Ahmed et al. [4] showed that the Haar transform can be computed using a Cooley-Tukey-type algorithm that is implemented in $2(N-1)$ additions/subtractions. This algorithm is derived by relating the Haar transform to the modified Walsh-Hadamard transform using a simple bit-reversal scheme.

P. R. Roeser et al. [13] presented the implementation of Fast Haar transform by performing the transform in place and limiting the amount of data movement there by attaining greater memory efficiency and speed.

T. S. Lee [67] extended to two dimensions the frame criterion developed by Daubechies for 1-D wavelets, and it computes the frame bounds for the particular case of 2-D Gabor wavelets. He also derived the conditions under which a set of continuous 2-D Gabor wavelets will provide a complete representation of any image. He found the self-similar wavelet parameterizations, which allow stable reconstruction by summation as though the wavelets formed an orthonormal basis.

2.2.6 Haar Transform

A reformulation of the Haar transform algorithm is used to design systolic arrays for data compression by G. M. Megson [85]. First a triangular array is developed for the normalised 1-D transform and it is then extended to produce an inverse transformation. The 1-D designs are then incorporated into a 2-D design for image compression using row and column operations.

A transformation to localize the equations defining the successive levels of the Mallat pyramid for 2-D Haar wavelets was presented by P. Lenders et al. [92]. A methodology for implementing these wavelet transforms in parallel architectures like systolic arrays was also proposed by them and shown that there is a perfect match between the wavelet algorithms and the multiphase multirate array (MPRA) architectures.

An algorithm to compute the running discrete Haar wavelet transform of samples which reduces the computational complexity to $\log_2 N$ is proposed by J. A. R. Macias et al. [152].

2.2.7 MRT

A transform named MRT was proposed by R. C. Roy et al. [140], which represent 2-D signals in terms of real additions alone rather than using complex multiplications. This transform maps the data matrix into M matrices using real additions alone. Some of the properties of this transform are also presented.

R. C. Roy et al. [155] presented MRT as an alternative approach for frequency-domain representation of 2-D signals and a comparison is made with DFT. MRT helps to do the frequency domain analysis of two-dimensional signals without any complex operations but in terms of real additions.

M. S. Anish Kumar et al. [158] developed an image compression and decompression technique based on 8×8 MRT. The advantage of this method is the flexibility in determining the percentage compression at the expense of image quality by choosing appropriate threshold.

R. C. Roy [167] presented a procedure to obtain a lean MRT representation of an image when the image size is a power of two. The lean MRT coefficients are unique, numerically compact and require only the same memory space as required for the original image and that the proposed lean MRT representation can be used effectively to compress images.

A transform coder based on 4×4 MRT was proposed by M. S. Anish Kumar et al. [185] and its performance are analyzed for all types of gray scale images.

2.3 Implementations

2.3.1 DFT

A decimation-in-time radix-2 FFT algorithm was considered by L. N. Bhuyan et al. [16] for performance analyses in multiprocessors with shared bus, multistage interconnection network (MIN), and in mesh connected computers. It is shown that a computer with multistage interconnection network gives much higher speedup for processors greater than 16 and is more cost effective even with the high cost of switches when compared to a shared bus multiprocessor.

S. M. Said [20] demonstrates an FFT algorithm implemented on the 68000 microprocessor that can calculate a 256-point transform in less than 48 ms. The algorithm employs an interesting method of scaling data to overcome overflow.

In [24] five major DFT algorithms were evaluated on seven different computers by M. A. Mehalic et al. It is found that on the average, data transfers account for a greater percentage of the execution time than floating-point operations.

T. K. Truong et al. [30] used a conventional prime factor DFT algorithm of the Winograd type to realize a discrete Fourier-like transform on the finite field $GF(q^2)$, where q is a Mersenne prime. A pipeline structure is used to implement this prime factor DFT over $GF(q^n)$.

R. Polge et al. [34] compares a multiple radix Fast Fourier Number Theoretic Transform with the standard FFT algorithms in terms of performance and hardware cost.

A. Gupta et al. [39] analyzed the scalability of the parallel FFT algorithm on mesh and hypercube connected multicomputers. The paper also present experimental speedup results on a 1024-processor Neube/1 multicomputer which support the analytical results.

A VLSI architecture was suggested by I. S. Reed [48] for the simplified arithmetic Fourier transform (AFT) algorithm using a butterfly structure which reduces the number of additions by 25% of that used in the original AFT algorithm.

S. I. Sayegh [50] described a method for performing FFT's of various sizes, whose sizes are powers of the pipeline's radix, simultaneously in one pipeline processor. The processor consists of several stages of butterfly computational elements alternated with delay-switch-delay (DSD) modules that reorder the data between the butterfly stages. FFT's of radix 2, radix 4, and mixed 2 and 4 are considered.

A. Kumar et al. [57] proposed an FFT algorithm, which minimizes the number of cache misses, after analyzing the implementation of some existing FFT algorithms.

Parallel architectures for short time Fourier transform based on adaptive time-recursive processing was proposed for efficient VLSI implementation in [58] by K. J. Ray et al. Only $N - 1$ multipliers and $N + 1$ adders are required.

Variants of the Winograd FFT algorithm for prime transform size are derived, that offer options as to the operational counts and arithmetic balance by J. W. Cooley et al. [55]. The implementations on VAX, IBM 3090 VF, and IBM RS16000 are also discussed For processors that perform floating-point addition, floating-point multiplication, and floating-point "multiply-add" with the same time delay, variants of the FFT algorithm have been designed such that all floating-point multiplications can be overlapped by using "multiply-add." The use of a tensor product formulation, throughout, gives a means for producing variants of algorithms matching to computer architectures.

N. Shirazi et al. [64] implemented the 2-D FFT on a FPGA-based custom computer namely, Splash-2. The computation of a 2-D FFT requires $O(N^2 \log 2N)$ floating point arithmetic operations for an $N \times N$ image.

H. Park et al. [56] proposed modular and area efficient VLSI architectures for computing the arithmetic Fourier transform (AFT). By suitable design of PE's and I/O sequencing, nonuniform data dependencies in the AFT computation which require nonequidistant inputs and assignment of Mobius function values are resolved. The design achieves $O(N)$ speedup.

T. S. Wailes [60] described the use of VHDL in the specification, design, and development of a large-scale project that includes several custom ASICs, standard digital components, board-level

design, and bus interfacing. The common mechanism for the design was the use of VHDL in system testing, behavioral descriptions, structural descriptions, and synthesis. The use of VHDL allowed the project to be completed in one-fifth of the time used for previous methods.

A parallel FFT algorithm that removes the complex multiplier between the two pipeline stages is proposed in [65] by Y. T. Ma, which also enables each FFT processor at the pipelines to be integrated easily onto a single chip. The algorithm also simplifies the address generation of twiddle factors and reduces the number of twiddle factors to a minimum as it is now.

An approach for the systolic implementation of FFT algorithms was presented by H. Lim et al. [68] which is based on the fundamental principle that a 1-D DFT can be decomposed to a 2-D DFT (with or without twiddle factors) and the 2-D DFT can be computed efficiently on a 2-D systolic array. The essence of the proposed systolic array is to combine different types of semi-systolic arrays into one array.

In [76], two algorithms namely a stage by stage method and a multistage method parallel radix R FFT algorithms on a multiprocessor or multicomputer system with a global interconnection network was proposed by O. Taketa et al. The paper shows that the communication time is very sensitive to and affected by data exchange strategy. These algorithms are implemented on two commercial massively parallel computers (nCUBE/2 and CM5) and measured the communication time.

In [82] an adaptive FFT program that tunes the computation automatically for any particular hardware was presented by M. Frigo et al. The above program's self-optimizing approach usually yields significantly better performance than all other publicly available software.

T. Chen [90] presented an optimized column FFT architecture, which utilizes bit-serial arithmetic and dynamic reconfiguration to achieve a complete overlap between computation and communication.

In [96], a fixed-point FFT algorithm was presented by Jizhong I-fan et al., which can make designers easily, adjust the precision and execution time of FFT. It also analyzes the above algorithm from the viewpoint of round-off error analysis and presents the benchmarks on 'C620.

A complex parallel Radix-4 FFT algorithm was simulated, implemented and realized in hardware using VHDL and a Xilinx Virtex-E 1000 FPGA circuit by M. Nilsson [103]. The VHDL code was simulated and synthesized in Ease and Synplify Environment and provide speed improvements due to a parallel approach.

P. Rodriguez V [113] implemented a general radix -2 FFT algorithm for SIMD on the Intel Pentium and Motorola Power PC architecture for 1-D and 2-D.

J. Heikkinen et al. [116] designed an application specific instruction set processors (ASIP) for a 32 point DCT using the tools from the MOVE frame work, which is semi-automatic design methodology for designing processors that utilize the paradigm of transport triggered architecture. In ASIP design the hardware resources are tailored according to the requirement of the application.

In [122], a method to minimize memory references due to twiddle factors for implementing any existing FFT algorithms on DSP processors was presented by Y. Tang et al. The above method takes advantage of previously proposed twiddle factor reduction method (TFRM) and twiddle-factor-based butterfly grouping method (TFBBGM). DIT FIT implementation in TI TMS320C64x DSP, achieves an average reduction in the number of memory references by 79% for accessing the twiddle factors, and 17.5% reduction in the number of clock cycles.

Regular and nonmultiplicative mapping algorithms between different types of Generalized Discrete Fourier Transform (GDFT) were proposed by H. I. Saleh et al. [124]. Hardware realization of 16-point FFT, based on the proposed mapping algorithms, with real twiddle factor butterfly rather than complex twiddle factors in traditional FFT algorithms, is implemented in Xilinx XC4000 and Virtex series FPGA.

P. Coussy et al. presented [147] a methodology and a tool that permit the High-Level Synthesis of DSP applications, under both I/O timing and memory constraints. Based on formal models and a generic architecture, this tool helps the designer in finding a reasonable trade-off between the circuit's latency and its architectural complexity. The efficiency of the approach is demonstrated on the case study of a FFT algorithm.

An FFT array processing mapping algorithm was proposed in [142] by Z. Liu in which, arbitrary $2k$ butterfly units (BUs) could be scheduled to work in parallel on $n = 2s$ data ($k = 0, 1, \dots, s-1$). An 18-bit word-length 1024 point FFT architecture with 4 BUs is given to demonstrate this mapping algorithm. The design is implemented with TSMC 0.18 μ m CMOS technology. This processor could complete 1024 FFT calculation in 7.839 μ s.

To try to reconcile the dual requirements of high performance and ease of development, I.S. Uzun et al. [143] reported on the design and realization of high level frame work for the implementation of 1-D and 2-D FFTs for real-time applications. A wide range of FFT algorithms, including radix-2, radix-4, split-radix and fast Hartley transform (FHT) have been implemented under a common framework in order to enable the system designers to meet different system requirements.

G. Lakshminarayanan [144] developed a techniques for efficient implementation of FPGA based wave-pipelined (WP) multipliers, accumulators, and filters. WP multipliers of size 2×6 and the filters using them are found to be 11% faster and require lower power than those using pipelined multipliers. Filters with higher order WP multipliers also operate with lower power at the cost of speed. The delay-register products of such filters are found to be about 60% lower than those using the pipelined multipliers.

J. Y. Oh et al. [146] proposed the modified radix-2 and the radix-4 FFT algorithms and efficient pipeline FFT architectures based on those algorithms for Orthogonal Frequency Division Multiplexing (OFDM) systems. From the synthesis simulations of a standard 0.35 μ m CMOS SAMSUNG process, a proposed Canonical Signed Digit (CSD) constant complex multiplier achieved

more than 60% area and power efficiency when compared to the conventional programmable complex multiplier.

T. Lenart [162] presented architectures for supporting dynamic data scaling in pipeline FFTs, suitable when implementing large size FFTs in applications such as digital video broadcasting and digital holographic imaging. A 2048 point pipeline FFT has been fabricated in a standard CMOS process and a FPGA prototype integrating a 2-D FFT core in a larger design.

CORDIC based split-radix FFT/ IFFT processor dedicated to the computation of 2048/4096/8192 point DFTs was presented by T. Y. Sung [159]. The arithmetic unit of a butterfly processor and a twiddle factor generator are based on a CORDIC algorithm. The modified pipelining CORDIC arithmetic unit is employed for complex multiplication. A CORDIC twiddle factor generator is proposed and implemented for reducing the size of ROM required for storing the twiddle factors.

X. Li [163] compares the performance of a single processor implementation with two types of dual-processor implementations for a widely used Radix-2 N -point FFT algorithm in terms of processing speed and FPGA resource utilization. In the first dual-processor implementation, the partitioning is performed based on the computation complexity - $O(M\log(N))$ of the Radix-2 FFT algorithm. In the second implementation, the partitioning is based on a detailed profiling procedure applied to each line of the code in the single-processor implementation. This result shows that detailed profiling is crucial in identifying the bottlenecks of an algorithm (i.e., all the factors are taken into consideration) and consequently the algorithm can be efficiently mapped on a multiprocessor system based on the correct decision.

In [157] O. Atak et al. presented two Application-Specific Instruction-Set Processor (ASIP) design concepts for the Cached FFT algorithm (CFFT). A reduction in energy dissipation of up to 25% is achieved compared to an ASIP for the widely used Cooley-Tukey FFT algorithm, which was designed by using the same design methodology and technology. Further, a modified CFFT algorithm, which enables a better cache utilization, was also presented. This modification reduces the energy dissipation by up to 10% compared to the original CFFT implementation.

The modified Single-path Delay Feedback (SDF) architecture for FFT implementation, which implements a mixed Decimation-in-Frequency (DIF) /Decimation-in-Time (DIT) FFT algorithm, was proposed S. Lee et al. [170]. The proposed architecture has the same throughput as that of Radix-4 SDF and Radix-4 MDC architecture, and reduces the latency and hardware complexity with some tradeoff in hardware complexity increase compared to original SDF.

A prime length DFT can be reformulated into a $(N - 1)$ length complex cyclic convolution and then implemented by systolic array or distributed arithmetic. C. Cheng et al. [168] proposed a hardware efficient fast cyclic convolution algorithm which is combined with the symmetry properties of DFT to get a hardware efficient fast algorithm for small length DFT, and then WFTA is used to control the increase of the hardware cost when the transform length N is large. The proposed design

has much more choices for different applicable DFT transform lengths and the processing speed can be flexible and balanced with the hardware cost.

Memory reference reduction methods to minimize memory references due to twiddle factors for implementing various different FFT algorithms on DSP were presented by Y. Wang [169]. The proposed methods first group the butterflies with identical twiddle factors from different stages in the FFT diagrams and compute them before computing other butterflies with different twiddle factors, and then reduce the number of twiddle factor lookups by taking advantage of the properties of twiddle factors. Consequently, each twiddle factor is loaded only once and the number of memory references due to twiddle factors can be minimized. Performance gain is achieved for implementing radix-2 DIT FFT algorithms on TI TMS320C64x DSP using the above methods.

B. J. Mohd [175] examined the superscalar pipeline FFT algorithm and architecture. The algorithm presents a memory management scheme to prevent memory contention throughout the pipeline stages. The fundamental algorithm, a switch-based FFT pipeline architecture and an example 64-point FFT pipeline was presented. The pipeline consists of $\log_2 N$ stages, where N is the number of FFT points. Each stage can have M Processing Elements (PEs.) As a result, the architecture speed up is $M \times \log_2 N$. The pipeline algorithm is configurable to any $M > 1$.

S. K. Palaniappan [177] described the detail design of semi-custom ASIC CMOS FFT architecture for computing 16 point radix-4 FFT, and realized, utilizing 0.18 μm standard CMOS technology. Fixed point data format is preferred in comparison of floating point data format for a shorter dynamic range and reduced hardware utilization; thus, catering to the needs of portability. The computation results at particular stage are rounded to avoid overflow issue and to be stored in register.

M. Szmajda et al. [176] presented basic analog-to-digital conversion parameters, which are defined in the norms, the time duration analysis of DFT algorithms, considering described assumptions and comparing them to FFT. The software including the aforementioned codes was implemented in a low-cost power quality measurement system based on the TMS320c6713 floating point DSP processor from Texas Instruments.

In [178], an efficient algorithm with parallel and pipelining methods was proposed by N. Mahdavi et al. to implement high speed and high resolution FFT algorithm. Latency reduction is an important issue to implement the high speed FFT on FPGA. The Proposed FFT algorithm shows the latency of 5131 clock pulse when N refers to 1024 points. The design has the mean squared error (MSE) of 0.0001 which is preferable to Radix 2 FFT.

J. H. Bahn et al. [180] presented several parallel FFT algorithms with different degree of communication overhead for multiprocessors in Network-on-Chip (NoC) environment.

2.3.2 Modified DFT

R. Gopikakumari et al. [71] proposed a Modulo Arithmetic based Hierarchical Neural Network (MAHNN) model to implement $N \times N$ point DFT for $((N))_4 = 2$. In this model, there are 4 layers of

computational units, the first three layers are using only real additions and the last layer alone involves complex operation, which is also scalar multiplication of pre-computed twiddle factor values.

Performance evaluation of MAHNN model to implement $N \times N$ point DFT for $((N))_4 = 2$ was done in [72] by R. Gopikakumari et al. This scheme of computation has the advantage that it is based on parallel and distributed scheme in which each operation is a simple real addition except at the last layer, where it will be converted to the complex form.

R. Gopikakumari et al. [75] implemented a parallel distributed computation of 6×6 point DFT.

In [77] R. Gopikakumari et al. used the Parallel distributed computation of 6×6 point DFT for the determination of Ray Paths.

2.3.3 DCT

In [42] S. Uramoto et al. describes a 100-MHz 2-D DCT core processor, which is applicable to the real-time processing of HDTV signals. An excellent architecture utilizing a fast DCT algorithm and multiplier accumulators based on distributed arithmetic have contributed to reducing the hardware amount and to enhancing the speed performance. A layout scheme with a column-interleaved memory and a ROM circuit are introduced for the efficient implementation of memory-based signal processing circuits. The core integrates about 102K transistors, and occupies 21 mm^2 using $0.8 \text{ }\mu\text{m}$ double-metal CMOS technology.

D. Slawecki et al. [49] presented an 8×8 2-D DCT/IDCT processor chip that can be used for high data rate image and video coding. It is designed using the MOSIS $2 \text{ }\mu$ scalable CMOS technology. The chip is highly pipelined with a latency of 127 cycles and a maximum delay time of 18 ns between any two pipeline stages.

A fully parallel architecture for the computation of either the forward or the inverse 2-D DCT, based on row-column decomposition is presented by A. Aggoun [120]. It uses the same 1-D DCT unit for the row and column computations and $(N^2 + N)$ registers to perform the transposition. It can compute a $N \times N$ point DCT at a rate of one complete transform per N cycles after an appropriate initial delay.

G. A. Ruiz [141] presented a parallel-pipelined architecture of an 8×8 forward 2-D Integer Cosine Transform (ICT) processor for image encoding. The ICT kernel is integer-based, so computation only requires adding and shifting operations. A fully pipelined row-column decomposition method based on two one-dimensional (1-D) ICTs and a transpose buffer based on D-type flip-flops is used. The main characteristics of 1-D ICT architecture are high throughput, parallel processing, reduced internal storage, and 100% efficiency in computational elements.

A schedule for 2-D DCT computation to reduce the hardware cost based on the fast row/column decomposition algorithm was proposed in [171] by S. C. Hsia. With this approach, the

transposed memory can be simplified using shift-registers for the data transposition between two 1-D DCT units. A special shift cell with MOS circuit is designed using the energy transferring methodology. They claim that the maximum frequency of shift operation achieved is about 120 MHz, when implemented by 0.35 μm technology.

Z. Szadkowski [181] described an optimization of 16 point DCT algorithm using parallel architecture implemented into a FPGA.

S. An et al [186] computed the 2-D DCT by a simple procedure of the 1-D recursive calculations involving only cosine coefficients. The recursive kernel for the proposed algorithm contains a small number of operations. Architecture for the 2-D DCT designed by direct mapping from the computation structure of the proposed algorithm has been implemented in an FPGA board. Another architecture using one recursive computation block to perform different functions developed by them reduced the hardware by almost one-half.

A hardware implementation of a hybrid architecture to compute three 8 point transforms namely the DCT, the DFT, and the DWT on a single FPGA was proposed by K. Wahid et al. [187]. The architecture is based on an element-wise matrix factorization and row-permutation algorithm, where the forward basis transformation matrices are decomposed into multiple submatrices and the common units are shared among them. The hardware implementation is parallel, pipelined and multiplication-free.

L. V. Agostini [106] presented the architecture and VHDL design of a 2-D DCT for JPEG image compression. This architecture uses 4792 logic cells of one Altera Flex 10k E FPGA and reaches an operating frequency of 12.2 MHz. One input block with 8×8 elements of 8 bits each is processed in 25.2 μs and the pipeline latency is 160 clock cycles.

2.3.4 Wavelet Transform

A single chip implementation of DWT is described by G. Knowles [33].

A VLSI architecture suitable for 2-D orthogonal wavelet transforms was presented by A. S. Lewis [41], which for the Daubechies wavelet implements the forward and inverse transforms without multipliers. The four-coefficient Daubechies wavelet transform has excellent spatial and spectral locality, properties which make it very useful in image compression. A sample implementation is described.

In [107] the FPGA implementation of the wavelet transform with lattice filters and achieves the FFT with the Coordinate Rotational Digital Computation (CORDIC) was presented by W. Zhilu et al. Then, the emulation data of the Daubechies D4 & D6 wavelet transforms and the FFT with 16 points are given, and their performances are analyzed.

M. Nibouche et al. in [108] presented a framework for an FPGA-based DWT system. The approach helps the end-user to generate FPGA configuration for DWT at a higher level without any knowledge of the low-level design styles and architectures.

Z. Razak [114] described the development process of a DWT chip design comprising of simulation by MATLAB™, simulation and synthesis by SYNOPSIS™.

A methodology for building up filters for JPEG 2000 std that can be applied efficiently for both 1-D forward and inverse Wavelet transform was presented by G. Dimitroulakos et al. [118]. They are characterized by reduced memory access, area, power and the ability of progressive computation. The architecture can be embedded in systems, which are used in real time applications.

N. Aranki et al. [117] described an efficient hardware implementation of the DWT suitable for deployment on a reconfigurable FPGA based platform. This implementation is based on the lifting factorization of the wavelet filter banks that uses the overlap-state algorithm. They claim that it minimizes memory usage, computational complexity and communication overhead associated with parallel implementation.

A VHDL implementation of FDWT utilizing its lossless features and based on the JPEG 2000 was presented by S.M. Aziz et al.[128]. The architecture does not comprise any hardware multiplier unit and therefore suitable for development of high performance image processors.

In [129], the implementation of the 2-D DWT in VHDL is presented by R. Mateos et al. The proposed system has as advantages the segmentation of the wavelet algorithm in different processes and presents as novelty the control and addressing of input and output data.

An approach towards VLSI implementation of the DWT for image compression was presented by A. A. Muhit et al. [137]. The design follows the JPEG2000 standard and can be used for both lossy and lossless compression. In order to reduce complexities of the design, linear algebra view of DWT and IDWT has been used.

P. Salama et al. [154] presented a VHDL implementation of a decomposition unit based on Mallat's fast Wavelet Transform, which utilizes a two-channel c sub band coder. The units were simulated, synthesized, and optimized using Mentor® design tools.

Two different architectures for reversible integer to integer Wavelet Transforms was proposed in [127] by M. A. B. Ayed et al. One uses the lifting frame work as an architecture support and the other make advantage of the two finite impulse filter structure representing the wavelet transform function. The architectures are evaluated based on their computational complexity, latency time, hardware occupancy and the maximum operating clock frequency.

2.3.5 Haar Transform

Linear systolic array architecture for the implementation of Haar, Walsh, and the DFTs based upon matrix vector multiplication algorithms where the matrix elements can be computed from their row and column indexes was presented by G. E. Bridges et al. [27]. The method presented enables the n^2 matrix elements to be computed *in situ* directly from the $2n$ matrix indexes. A generalized method is

given for the development of recursively formed matrices and specifically the VLSI implementation of the Haar and Walsh transforms.

Three VLSI computing architectures namely systolic tree architecture, linear data flow array and sequential queue architecture were proposed by K. J. Ray et al. [40] for fast implementation of the Haar transform.

A processor chip programmable between $N = 8$ and $N = 1024$ for 1-D IFHT was presented by G. A. Ruiz et al. [79]. The processor uses a low latency data-flow with an architecture that minimizes the internal memory and an adder/subtractor as the only computing element. The control logic has a single and modular structure and can be easily extended to longer transforms. A prototype of the 1-D IFHT processor has been implemented using a standard-cell design methodology and a 1.0- μm CMOS process on a 11.7 mm² die. The maximum data rate is close to 60 MHz.

An FPGA implementation of a processor for the 1-D IFHT programmable for $N = 8$ up to $N = 1024$, with low latency data flow is presented by O. Martin et al. [172]. It enables on-line computing for both the normalized and the non-normalized IFHT to be performed, with very low *nrmse*.

2.3.6 Hadamard Transform

An algorithm of a simple systolic array processor for the HAT was presented by M. H. Lee et al. [37] that provides high pipelining rates. It is based on Hadamard coefficient generator, which makes the signs of the Hadamard matrix (HAM) elements and the execution of matrix vector multiplication.

An architecture for the Fast Hadamard Transform, using distributed arithmetic techniques, was proposed by A. Amira et al. [99]. The associated design using both a distributed arithmetic ROM and Accumulator structure and a sparse matrix factorization technique are also described. The above architecture is implemented on a Xilinx FPGA board by A. Amira et al. [105].

The FPGA implementation of two architectures namely systolic architecture and distributed arithmetic techniques for the computation of fast Walsh-Hadamard transform is presented in [109] by A. Amira et al. The first approach uses the Baugh-Wooley multiplication algorithm whereas the second approach is based on both a distributed arithmetic ROM and accumulator structure, and a sparse matrix factorization technique. The second method exhibits better performances when compared to the first one.

I. Amer et al. [139] presented a VLSI prototype for the 2×2 Hadamard transform that is applied to the DC coefficients of the four 4×4 blocks of each chroma component as described in the MPEG -4 Part 10 Advanced Video Coding (AVC) standard. The transform is computed using add operations only and the architecture satisfies the real-time constraints required by HDTV.

A hardware unit for producing binary Orthogonal Variable Spreading Factor (OVSF), Hadamard and Walsh codes for WCDMA/CDMA2000 systems was presented by T. Rintakoski et al. [138]. The generator uses a spreading factor, mode select, and the code index as the control input.

A fast algorithm for the sequency-ordered complex Hadamard transform (SCHAT) based on the decomposition method of decimation-in-sequency was proposed by G. Bi et al. [182]. To support high speed real time applications, a pipelined hardware structure is also proposed to deal with sequentially presented input/output data streams which requires only $\log_2 N$ complex adder/subtractors and $2(N - 1)$ complex data stores for an $-N$ point SCHAT.

Fully pipelined simple modular structures for the hardware realization of DHAT were proposed by P. H. Meher et al. [184]. Four different pipelined modular designs for transform length $N = 4$ were derived from the kernel matrix of HAT. It is shown further that the HAT of transform length $N = 8$ can be obtained from two 4-point HAT modules, and similarly, the HAT of transform length $N = 16$ can be obtained from four 4-point HAT modules. Long length transforms can be computed from these short length modules.

2.4 Conclusion

The literature reviewed here set the background to develop the main idea in the work reported. The chapters to follow shall demonstrate this aspect.

CHAPTER 3

VISUAL REPRESENTATION AND COMPUTATION OF 2-D DFT

“A Picture is worth a thousand words”. Hence for a long time, visual methods have been successfully applied to analyze data, in many application domains. Ordinary visualization of such data can lead to over crowded and cluttered displays and are therefore of limited use, especially when the data volume is large. Data abstraction can help to gain insight even into large data sets. Providing appropriate methods to facilitate analysis of data is a key issue. This is the point where analytical methods come into play. Integrating visual and analytical methods has become an increasingly important issue. The method adopted in this chapter is also an illustration of the usefulness of combining visual and analytical methods. The analytical method should i) communicate the fact that something interesting has been found, ii) emphasize interesting data among the rest of the data and iii) convey what makes the data interesting. When the data size is huge, the challenge of visualizing it in a comprehensible manner can be dealt with by analytical methods.

In the modified 2-D DFT, explained in section 1.2.3.4, computation of the N^2 DFT coefficients Y_{k_1, k_2} involves M complex multiplications each. But for a particular (k_1, k_2) , (1.6) involves computation of z for all values of (n_1, n_2) which is time consuming. A visual representation of $N \times N$ DFT was developed, in [88], in terms of 2×2 DFT. It represented Y_{k_1, k_2} visually using a set of primitive symbols. The visual representation developed in [88] provided a relation between $N \times N$ point DFT and 2×2 DFT.

A visual representation based on 2×2 data on the other hand will give direct relationship between time domain data and the frequency domain representation in terms of pictures. These visuals are the representatives of data points involved in the computation of DFT coefficients. By analyzing the visual representation, we can extract lot of information to derive simple and efficient schemes for DFT computation.

3.1 Visual representation based on 2×2 DFT

Let X be the 2×2 data matrix and Y be the corresponding 2×2 DFT matrix as shown.

$$X = \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} \quad Y = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$$

Then the relationships between these two matrices [88] can be expressed by symbols as shown below.

$$Y_{00} = X_{00} + X_{01} + X_{10} + X_{11} \quad \begin{array}{c} \square \\ \text{LAP} \end{array} \quad (3.1)$$

$$Y_{01} = (X_{00} + X_{10}) - (X_{01} + X_{11}) \quad \begin{array}{c} \square \\ \text{RAP} \end{array} \quad (3.2)$$

$$Y_{10} = (X_{00} + X_{01}) - (X_{10} + X_{11}) \quad \begin{array}{c} \square \\ \text{LBP} \end{array} \quad (3.3)$$

$$Y_{11} = (X_{00} + X_{11}) - (X_{10} + X_{01}) \quad \begin{array}{c} \square \\ \text{RBP} \end{array} \quad (3.4)$$

By combination of (3.1), (3.2), (3.3) and (3.4) other relationships can be derived. The 36 primitive symbols and its Mnemonics based on 2×2 DFT are shown in fig. A.1. These primitive symbols were derived from the relation between 2×2 data and 2×2 DFT coefficients. The visual representation of 8×8 point DFT based on these primitive symbols is also shown in fig. A.2. The computation of Y_{k_1, k_2}^p doesn't require any multiplications. The frequency domain analysis of 2-D signals is also made easy using these visuals.

3.2 Visual representation based on 2×2 data

Visual representation of 2-D DFT coefficients can also be developed based on 2×2 data instead of 2×2 DFT. Such visuals provide a direct relationship between the data and the DFT coefficients. These visuals can be used to compute the DFT coefficients rather than computing (1.6) and (1.7). The patterns in the visual representation can be analyzed, there by exploring different computational schemes for 2-D DFT.






3.2.1 Primitive symbols

The expression for the 2-D DFT computation was restructured in [88], and [140], by exploiting the periodicity and symmetry properties of twiddle factors, thereby reducing the computational






complexity from N^2 complex multiplications for each DFT coefficient required in direct DFT to that of $N/2$ in the present approach.

Time consuming computation in (1.6) is simplified by the visual representation developed in [88], [69] in terms of 2×2 DFT. The representation of $N \times N$ DFT using 2×2 data is obtained by replacing the primitive symbols corresponding to 2×2 DFT with the primitive symbol corresponding to data. This will help to compute the DFT coefficients, without computing (1.6) and (1.7), by using it as a look up table. The computation of Y_{k_1, k_2}^p using the visual representation involves only real additions.

The 37 primitive symbols based on data are shown in fig. 3.1. In the primitive symbols “◻” and “◼” denote that the data from the respective position is to be added and subtracted respectively. The meaning of a few primitive symbols and the corresponding mnemonics used in the visual representation are given below:

1. The symbol  called MPA (Matrix Positive Above) indicates that the two data on the top (i.e. data on the $(0, 0)^{\text{th}}$ and $(0, 1)^{\text{th}}$ position) are positive and that on the bottom (i.e. on the $(1, 0)^{\text{th}}$ and $(1, 1)^{\text{th}}$ position) of the 2×2 data matrix are sign reversed for computing Y_{k_1, k_2}^p .
2. The symbol  called DP (Diagonal Positive) indicates that the data on the $(0, 0)^{\text{th}}$ and $(1, 1)^{\text{th}}$ position are positive and the rest of the data in the 2×2 matrix are not involved in the computation.
3. Symbol  named LAP (Left Above Positive) indicates that the data on the $(0, 0)^{\text{th}}$ position of the 2×2 data matrix is taken with a positive sign and rest of the data are not considered. Similarly, RAP (right above positive), RBP (right below positive), LBP (left below positive) indicates consideration of data at positions $(0, 1)$, $(1, 1)$ and $(1, 0)$ respectively. It is to be noted that in the visual representation a hollow square symbol represents single positive data point.
4. Symbol  named LAN (Left Above Negative) indicates that the data on the $(0, 0)^{\text{th}}$ position of the 2×2 data matrix is taken with a Negative sign and rest of the data are not considered. Similarly, RAN (right above Negative), RBN (right below Negative), LBN (left below Negative) indicates consideration of data at positions $(0, 1)$, $(1, 1)$ and $(1, 0)$ respectively. It is to be noted that in the visual representation a filled square symbol represents single negative data point.
5. Symbol  named HAP (Horizontal above positive) indicates that the two data points on the upper row of the 2×2 data matrix i.e. at positions $(0, 0)$ and $(0, 1)$ are taken with a positive sign and the rest are not considered. Similar is the case for HBP (Horizontal below positive), VLP

(Vertical left positive) and VRP (Vertical right positive) each considering 2 data points with a positive sign. The precise data positions for HBP are (1, 0) and (1, 1), they are (0, 0) and (1, 0) for VLP and that for VRP are (0, 1) and (1, 1). It is to be noted that in the visual representation a thin line symbol represents a group of 2 adjacent positive data points.

6. Symbol  named HAN (Horizontal above negative) indicates that the two data points on the upper row of the 2×2 data matrix i.e. at positions (0, 0) and (0, 1) are taken with a negative sign and the rest are not considered. Similar is the case for HBN (Horizontal below negative), VLN (Vertical left negative) and VRN (Vertical right negative) each considering 2 data points with a positive sign. The precise data positions for HBN are (1, 0) and (1, 1), they are (0, 0) and (1, 0) for VLN and that for VRN are (0, 1) and (1, 1). It is to be noted that in the visual representation a thick line symbol represents a group of 2 adjacent negative data points.
7. Symbol  named DN (diagonal negative) indicates that the two data points on the diagonal of the 2×2 data matrix i.e. at positions (0, 0) and (1, 1) are taken with a negative sign and the rest are not considered. A thick diagonal line is shown in the visual representation. Similarly CN (cross-diagonal negative) considers points on the cross-diagonal i.e., at positions (0, 1) and (1, 0). A thick cross-diagonal line shows it.
8. Symbol  named MPL (Matrix Positive on the Left) indicates that the two data on the left (i.e. on the (0, 0)th and (1, 0)th positions) are taken with a positive sign and the data on the right (i.e., on the (0, 1)th and (1, 1)th positions) of the 2×2 data matrix are taken with a negative sign for computing Y_{k_1, k_2}^p . The other similar symbol is MPA.
9. The  symbol HAPR (horizontal above positive right) indicates that two data points are considered with right one i.e., (0, 1) with a positive sign and left one i.e., (0, 0) with a negative sign. It is shown in the visual representation as a rectangle, which is half filled from the left side denoting negative data point and is half empty from the right side denoting positive data point. Similar symbols are used for HAPL, HBPR, HBPL, VLPA, VRPA, VLPB, VRPB, DPA, DPB, CPA and CPB.
10. The  symbol MP (matrix positive) indicates that all the four data points of a 2×2 matrix are considered with a positive sign. It is shown in the visual representation by a big square covering almost the whole block constructed with a thin line.

11. A blank block \square denotes that none of the four data points are necessary for the DFT computation using Y_{k_1, k_2}^p .

3.2.2 Visual representation

The coefficient Y_{k_1, k_2}^p , for any (k_1, k_2) and $0 \leq p \leq M - 1$, is represented visually in terms of the primitive symbols in fig. 3.1. The visual representation of Y_{k_1, k_2}^p will have $M \times M$ cells with a primitive symbol in each cell. For each time index (n_1, n_2) , the value of $z = ((n_1 k_1 + n_2 k_2))_N$ is determined. If $z = p$, the data is to be added, else if $z = p + M$, the data is to be subtracted and otherwise the data is to be neglected. The primitive symbol to be used in each cell can be selected from fig. 3.1, based on the number of data used from that cell and its sign. Thus each cell represents a mapping of 2×2 data into primitive symbol based on the value of z in (1.7). E.g., for any even value of N , if $(k_1, k_2) = (0, 0)$ then $((n_1 k_1 + n_2 k_2))_N = 0$ for all values of (n_1, n_2) in the $M \times M$ cells. Thus all the four data in each cell will be involved with plus sign if $p = 0$ and the primitive symbol will be \square with mnemonic ‘MP’. The visual representation for $k_1 = k_2 = 0$ exists only for $p = 0$. Similarly, for $k_1 = 0$ and $k_2 = M$, $z = 0$ or M when n_2 is even or odd respectively. Thus in the columns of even n_2 , the data is to be added and for odd values of n_2 the data is to be subtracted. Equivalently, all the cells will have the symbol \square corresponding to the mnemonic ‘MPL’ for $p = 0$ alone. Similarly we can construct the visual representation for any (k_1, k_2) corresponding to any even value of N . The fig. 3.2, 3.3 and 3.4 show the visual representation for $N = 4, 6$ and 8 respectively.

3.2.3 Analysis of the visual representation

The visual representation of DFT is analyzed for $N = 4$ to 64 . The representation shows specific pattern depending on 1) the appearance, 2) existence of Y_{k_1, k_2}^p for selected p based on (k_1, k_2) and 3) order N .

3.2.3.1 Classification of DFT coefficients based on the appearance

In fig. 3.2, 3.3, and 3.4, $Y_{0,0}^p$, $Y_{0,M}^p$, $Y_{M,0}^p$, and $Y_{M,M}^p$ have representation for $p = 0$ only and each of them have identical cells. There are only 4 such real coefficients, for any even N , each of them can be represented by just one cell and is classified as group 1.

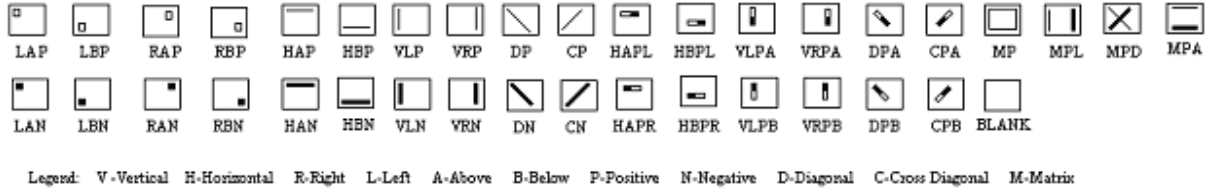


Fig. 3.1: Primitive symbols for visual representation of DFT coefficients using 2×2 data

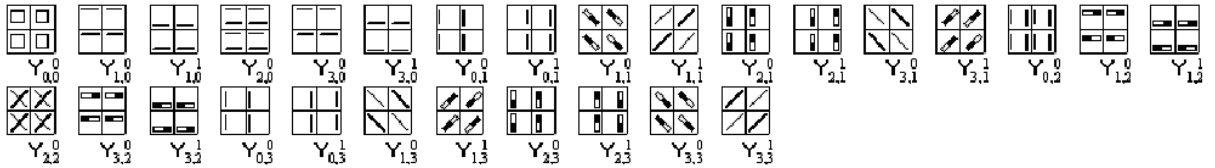


Fig. 3.2: Visual representation of 2-D DFT coefficients for $N = 4$

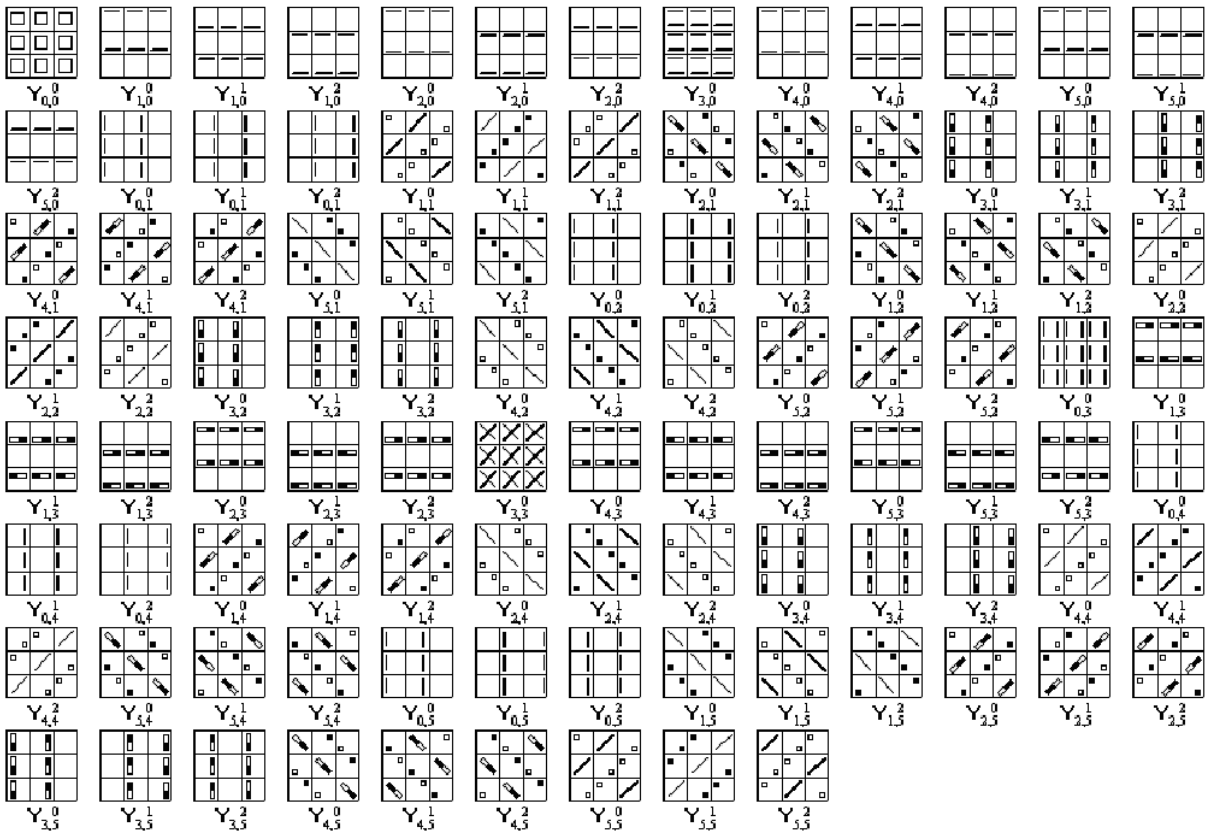


Fig. 3.3: Visual representation of 2-D DFT coefficients for $N = 6$

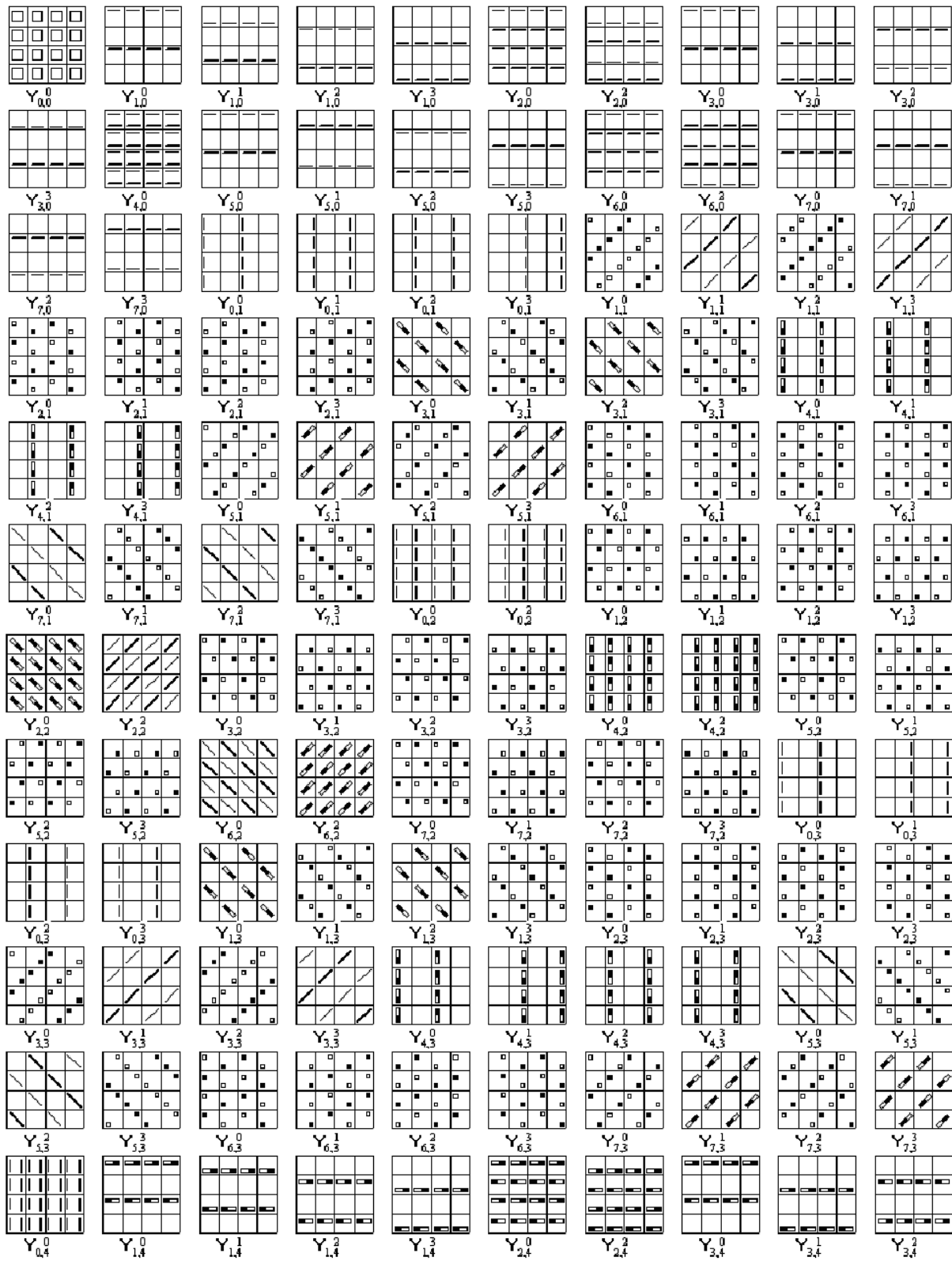


Fig. 3.4: Visual representation of 2-D DFT coefficients for $N = 8$

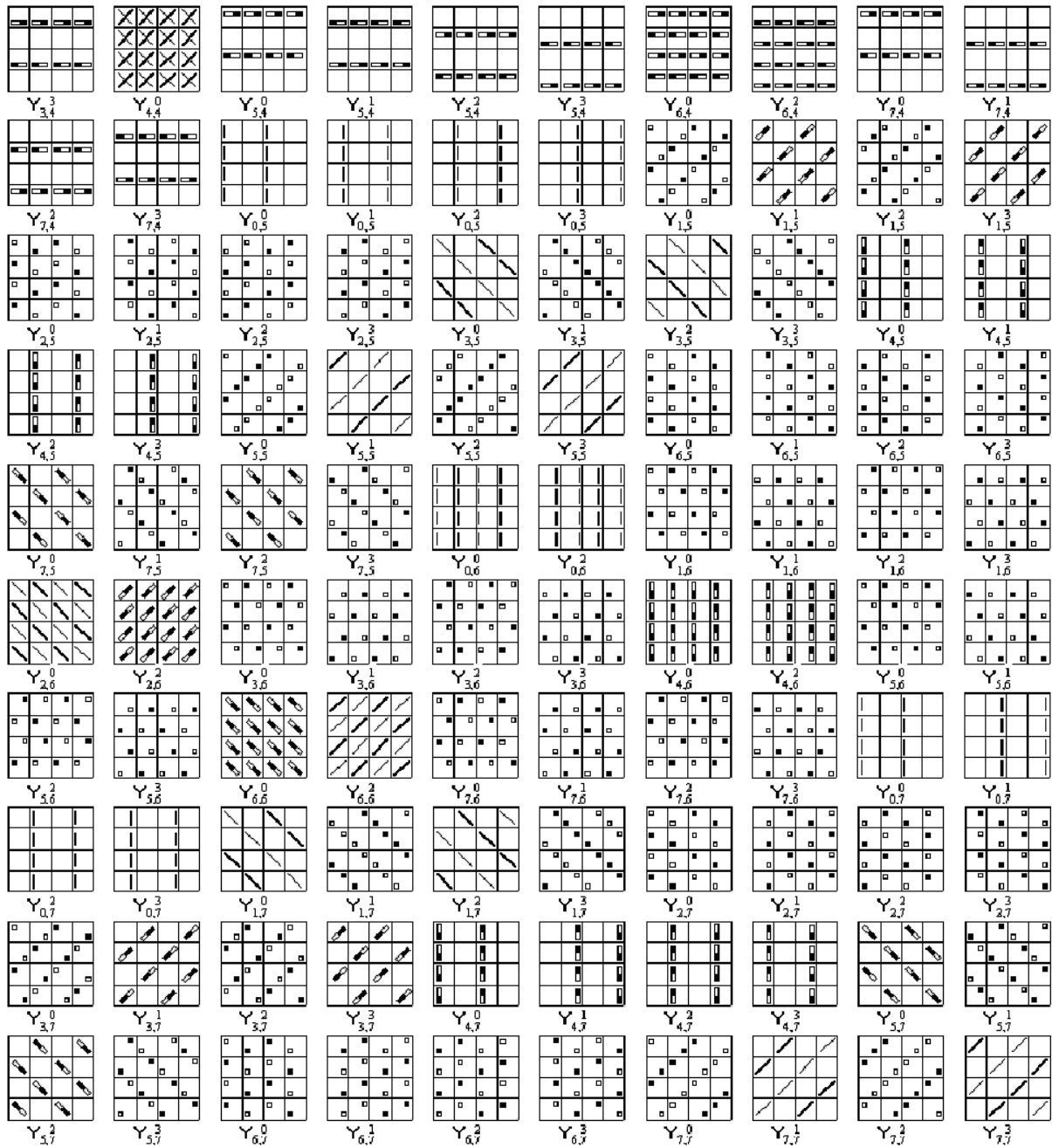


Fig. 3.4: Cont'd

In fig. 3.3 and 3.4 where, $N = 6$ and 8 respectively, $Y_{1,0}^0$, $Y_{1,0}^2$, $Y_{2,0}^0$, $Y_{2,0}^2$ etc. show a regular structure and use a pair of primitive symbols, where as $Y_{1,0}^1$, $Y_{1,0}^3$, $Y_{2,0}^1$, $Y_{2,0}^3$ etc. also show a regular structure but use another pair. When $N = 6$, as in fig. 3.3, $Y_{1,0}^0$, $Y_{1,0}^2$, $Y_{2,0}^2$ etc. use a pair of primitive symbols where as $Y_{1,0}^1$, $Y_{1,0}^3$, $Y_{2,0}^3$ etc. use another pair. Each column of these coefficients is identical

to the other columns of the coefficients. In $Y_{0,1}^0, Y_{0,1}^2, Y_{0,2}^2$ etc. each row is identical to the other rows. This type of characteristics is seen in the coefficients in the 0^{th} column, 0^{th} row, M^{th} column and M^{th} row other than in group 1, i.e., coefficients $Y_{k,0}, Y_{k,M}, Y_{0,k}, Y_{M,k}$ where $\text{gcd}(k, M) \neq M$. There are $4(N-2)$ such coefficients, classified as group 2.

The rest of the coefficients does not show any of the characteristics described above, but shows a specific pattern. E.g., $Y_{1,1}^0$ of $N=6$ in fig. 3.3, each column can be obtained from the other by a circular shift. The pattern also changes systematically depending on the change in the value of k_1, k_2 and p of Y_{k_1,k_2}^p . E.g., in fig. 3.4 where $N=8$, as p changes incrementally in $Y_{1,1}^{(p)}$, the pattern shift right horizontally in the same number of increments in a circular fashion. Similarly in $Y_{k_1,1}^0$, as k_1 changes incrementally, the pattern in each row shift left horizontally ' n ' times, where ' n ' is the row number of the data present in that $Y_{k_1,1}^0$. So also in Y_{1,k_2}^0 , as k_2 changes incrementally, the pattern in each column shift up vertically ' n ' times, where ' n ' is the column number. This is observed in $N^2 - 4(N-1)$ coefficients, for any even N , classified as group 3.

In brief, only one cell is enough to represent the coefficients in group 1 and one row/column for group 2 coefficients. In the case of group 3 coefficients, even though other rows/columns are not identical, they can be obtained by circular shift of one row/column. Again if the visual representation of Y_{k_1,k_2}^p for $p=0$ is available, the visual representation for other values of p (i.e. $p=1, 2$, etc.) could be obtained by circular shift of the pattern. Hence the visual representation can be obtained by visual manipulation rather than doing the computation in (1.6) and (1.7), which will simplify the representation.

3.2.3.2 Classification of DFT coefficients based on the existence of Y_{k_1,k_2}^p

Fig. 3.2, 3.3, and 3.4, show that for certain DFT coefficients, Y_{k_1,k_2}^p does not exist for all values of p . When $N=8$, $Y_{0,2}^{(p)}, Y_{2,0}^{(p)}, Y_{4,2}^{(p)}, Y_{6,4}^{(p)}$ etc. have $p=0$, and 2 only as shown in fig. 3.4. Here $\text{gcd}(k_1, k_2, M)=2$, $\text{gcd}(p, M)=2$ and $0 \leq p \leq M-1$, where 2 is a divisor of M [10]. Similarly when $N=12$, $Y_{0,3}^{(p)}, Y_{3,0}^{(p)}, Y_{6,3}^{(p)}, Y_{3,9}^{(p)}$ etc. have $p=0$, and 3 only, since 3 is a divisor of M . $Y_{4,4}^{(p)}, Y_{0,8}^{(p)}, Y_{8,4}^{(p)}$ etc. have $p=0, 2$, and 4 since $\text{gcd}(k_1, k_2, M)=2$, $\text{gcd}(p, M)=2$ and $0 \leq p \leq M-1$. When both k_1 and k_2 are 0, $Y_{0,0}^{(p)}$ exists for $p=0$ only. So the existence of Y_{k_1,k_2}^p depends on ' dm ', where ' dm ' is the divisor of M , which is illustrated in the following theorem.

Theorem 3.1

When ' dm ' is a divisor of M and $\text{gcd}(k_1, k_2, M) = dm$, then Y_{k_1,k_2}^p exists for $0 \leq p \leq M-1$, where $\text{gcd}(p, M) = dm$.

Proof

From (1.6) & (1.7), for Y_{k_1, k_2}^p to exist z should be equal to p or $p+M$ for at least one (n_1, n_2) ,

$0 \leq n_1, n_2 \leq N-1$. Let dm be a divisor of M , then $\frac{M}{dm} = r$.

Let $k_1 = s.dm$ and $k_2 = t.dm$, where $0 \leq s, t \leq \frac{N}{dm} - 1$.

Then from (1.7), $z = ((n_1.s.dm + n_2.t.dm))_N = (((n_1.s + n_2.t))_N ((dm))_N)_N$ (3.5)

For $0 \leq (n_1.s + n_2.t) \leq N-1$, (3.5) becomes

$$z = 0, dm, ((2.dm))_N, ((3.dm))_N, \dots, ((\alpha.dm))_N \dots (((\alpha+r).dm))_N \dots$$

$$\therefore z = ((q.dm))_N, 0 \leq q \leq \frac{N}{dm} - 1$$
 (3.6)

i.e., z is an integral multiple of dm .

Case 1: $k_1 = k_2 = 0$

Then $z = 0$ for all values of n_1 and n_2 . Hence Y_{k_1, k_2}^p exists for $p = 0$ only.

Case 2: $\gcd(k_1, k_2, M) = dm$

a) $p = \alpha.dm$, where $0 \leq \alpha \leq \frac{M}{dm} - 1$.

For $Y_{k_1, k_2}^{\alpha.dm}$ to exist z should be equal to $\alpha.dm$ or $(\alpha+r).dm$.

From (3.6), both α and $\alpha+r$ falls within the range of q

So $Y_{k_1, k_2}^{\alpha.dm}$ exist for $p = \alpha.dm$, $0 \leq \alpha \leq \frac{M}{dm} - 1$ and $k_1 = s.dm$ and $k_2 = t.dm$, $0 \leq s, t \leq \frac{N}{dm} - 1$ i.e., when

$\gcd(k_1, k_2, M) = dm$.

b) $p \neq \alpha.dm$, $0 \leq \alpha \leq \frac{M}{dm} - 1$

Since $p \neq \alpha.dm$ and $p+M \neq \alpha.dm + r.dm$, i.e., $p \neq \alpha.dm$ and $p+M \neq (\alpha+r).dm$ is not an integral multiple of dm . But, z should be an integral multiple of dm for Y_{k_1, k_2}^p to exist.

Hence Y_{k_1, k_2}^p does not exist for $p \neq \alpha.dm$, $0 \leq \alpha \leq \frac{M}{dm} - 1$ and $\gcd(k_1, k_2, M) = dm$.

Case 3: $\gcd(k_1, k_2, M) = \alpha.dm$ and $p = dm$.

$\alpha.dm$ is a divisor of M , $2 \leq \alpha \leq \frac{M}{dm} - 1$ and

$k_1 = s \cdot \alpha \cdot dm$ and $k_2 = t \cdot \alpha \cdot dm$, $0 \leq s, \alpha, t \leq \frac{N}{dm} - 1$. Then replacing s and t in (3.5) with $s \cdot \alpha$ and $t \cdot \alpha$, z becomes

$$z = ((n_1 \cdot s \cdot \alpha \cdot dm + n_2 \cdot t \cdot \alpha \cdot dm))_N = (((n_1 \cdot s + n_2 \cdot t))_N ((\alpha \cdot dm))_N)_N$$

Thus z is an integral multiple of $\alpha \cdot dm$. But when $p = dm$, z should be equal to dm or $(1 + r) \cdot dm$ for Y_{k_1, k_2}^p to exist. For that α should be 1 which contradict our initial assumption that α is greater than 1. Thus Y_{k_1, k_2}^p does not exist for $p = dm$ when $\gcd(k_1, k_2, M) = \alpha \cdot dm$.

From the above, it can be seen that even though the number of real additions (a_r) involved in the computation of each of the DFT coefficient is same, due to the non existence of certain Y_{k_1, k_2}^p , the number of complex multiplications (m_c) and the number of complex additions (a_c) has been reduced. This reduction is proportional to the value of the divisors of M . The number of Y_{k_1, k_2}^p (n_p) required for DFT computation decreases as the value of the divisor, dm increases, where, $\gcd(k_1, k_2, M) = dm$. E.g., when $N = 16$, $Y_{1,1}^p$ has $p = 0, 1, 2, 3, \dots, 7$, $Y_{2,0}^p$ has $p = 0, 2, 4$ and 6 , $Y_{4,8}^p$ has $p = 0$ and 4 only and $Y_{8,8}^p$ has $p = 0$ only. It can be observed that when $dm = 1, 2, 4$ & 8 $n_p = M, \frac{M}{2}, \frac{M}{4}$ & $\frac{M}{8}$ respectively. Hence, $m_c = n_p - 1 = a_c$. This is clear from table 3.1 for $N = 16$. Thus the number of complex multiplication for a DFT coefficient is $M/dm - 1$.

Table 3.1: Influence of ‘ dm ’ when $N = 16$

$Y_{k_1, k_2}^p, \gcd(k_1, k_2, M) = dm$	dm	a_r	n_p	m_c	a_c
$Y_{0,1}, Y_{2,1}, Y_{3,2}, \text{ etc.}$	1	255	8	7	7
$Y_{0,2}, Y_{2,2}, Y_{2,4}, \text{ etc.}$	2	255	4	3	3
$Y_{0,4}, Y_{4,4}, Y_{4,8}, \text{ etc.}$	4	255	2	1	1
$Y_{0,8}, Y_{8,8}, Y_{8,0}, \text{ etc.}$	8	255	1	0	0

3.2.3.3 Classification of visual representation based on N

The visual representation shows specific properties depending on N and can be classified into four.

(i) $N/2$ prime

The visual representation of $N = 6$ is shown in fig. 3.3 where $N/2$ is prime. In fig. 3.3, $Y_{0,0}^{(p)}$, $Y_{0,3}^{(p)}$, $Y_{3,0}^{(p)}$, and $Y_{3,3}^{(p)}$ have $p = 0$ only, where as all other coefficients have $p = 0, 1$ and 2 . Here $M = 3$ is prime, hence $dm = 1$ and 3 . When ‘ $dm = 3$ ’, Y_{k_1, k_2}^p exists for $p = 0$ only. Similarly when ‘ $dm = 1$ ’

Y_{k_1, k_2}^p exists for all values of p , $0 \leq p \leq M - 1$. The above characteristics can be seen whenever $N/2$ is prime, since the divisors are 1 and M .

(ii) $((N))_4 = 2$ and $N/2$ not prime

The smallest positive integer in this category is 18 and the divisors 'dm' of M are 1, 3 and 9. When $dm = 1$, Y_{k_1, k_2}^p exists for all $0 \leq p \leq M - 1$ for a given (k_1, k_2) such that $\gcd(k_1, k_2, M) = 1$ as seen in the visual representation.

Similarly when $\gcd(k_1, k_2, M) = 3$, Y_{k_1, k_2}^p exists for $p = 0, 3,$ and 6 only and when $\gcd(k_1, k_2, M) = 9$, Y_{k_1, k_2}^p exists for $p = 0$ only. Here dm is odd, as M is odd and composite. Similar is the case when $N = 30, 42, 50$ etc. where $N/2$ is odd and composite.

(iii) N power of 2

The visual representation of $N = 8$ is shown in fig. 3.4 where N is a power of 2. When $N = 8$, the divisor 'dm' = 1, 2, and 4. When $dm = 1$, from the visual representations derived, Y_{k_1, k_2}^p exists for all $0 \leq p \leq M - 1$ for a given (k_1, k_2) such that $\gcd(k_1, k_2, M) = 1$.

Similarly, when $\gcd(k_1, k_2, M) = 2$, Y_{k_1, k_2}^p exists for $p = 0, \& 2$ and when $\gcd(k_1, k_2, M) = 4$, Y_{k_1, k_2}^p exists for $p = 0$ only. In this category dm is a power of 2. Same features can be seen when $N = 4, 16, 32, 64$ etc.

(iv) $((N))_4 = 0$ and N not a power of 2

The smallest positive integer in this category is 12 and the divisors 'dm' of M are 1, 2, 3 and 6. From the visual representations derived, when $dm = 1$, Y_{k_1, k_2}^p exists for all (k_1, k_2) such that $\gcd(k_1, k_2, M) = 1$, $0 \leq p \leq M - 1$. Similarly, when $\gcd(k_1, k_2, M) = 2$, Y_{k_1, k_2}^p exists for $p = 0, 2$ and 4 ; when $\gcd(k_1, k_2, M) = 3$, Y_{k_1, k_2}^p exists for $p = 0$ and 3 only and when $\gcd(k_1, k_2, M) = 6$, Y_{k_1, k_2}^p exists for $p = 0$ only. 'dm', the divisors of M in this case are both even and odd. Similar is the case when $N = 20, 24, 28$ etc. where $N/2$ is even.

3.2.4 Redundancy

Redundancy in the visual representation can be noticed at three different levels. In section 3.2.3.1, it is inferred that only one cell is enough to represent the coefficients in group 1 and one row/column for group 2 and 3 coefficients. This is the redundancy available within the visual representation of Y_{k_1, k_2}^p and termed as *first level* of redundancy.

In section 3.2.3.1, it is also inferred that if the visual representation of Y_{k_1, k_2}^p for $p = 0$ is available; the visual representation for other values of p (i.e. $p = 1, 2$, etc) could be obtained by circular shift of the pattern. Hence the visual representation of Y_{k_1, k_2}^p for any one value of p is enough to represent the DFT coefficient. This is the redundancy that can be observed within a DFT coefficient, termed as *second level*.

Several similarities can be noticed in the visual representations of the 1st column coefficients and the last column coefficients when $N = 8$, where the index k_2 is 1 and 7 respectively, which can be seen in fig. 3.4. The visual representation for Y_{k_1, k_2}^0 of 1st column coefficients ($Y_{1,1}^0, Y_{2,1}^0, Y_{3,1}^0$, etc.), from top to bottom are same as that of the corresponding coefficients in the last column, taken in the reverse order ($Y_{7,7}^0, Y_{6,7}^0, Y_{5,7}^0$, etc.), except for the 0th row coefficients. E.g., the visual representation of $Y_{1,1}^0$ is same as that of $Y_{7,7}^0$. Y_{k_1, k_2}^0 of 0th row coefficients of the above columns are same, i.e., the visual representation of $Y_{0,1}^0$ is same as that of $Y_{0,7}^0$. The representation for other values of p is also related. E.g., the visual representation of $Y_{7,7}^3$ is the sign reversed form of $Y_{1,1}^1$. Several such similarities can be noticed between other coefficients also. The above similarities are predominant in the visual representations of higher orders. E.g. when $N = 12$, the visual representations for $Y_{1,2}^0, Y_{11,10}^0, Y_{5,10}^0$ and $Y_{7,2}^0$ are similar. The frequency indices of $Y_{1,2}^0$ is N minus the corresponding indices of $Y_{11,10}^0$. The indices of $Y_{5,10}^0$ are 5 times that of $Y_{1,2}^0$. Similarly the visual representation of $Y_{11,10}^5$ and $Y_{7,2}^1$ is the sign reversed form of $Y_{1,2}^1$ and $Y_{5,10}^5$. Here (k_1, k_2) have the same relations as explained above. There is also a definite relationship between the values of p in such coefficients. Value of p in $Y_{11,10}^5$ is M minus the value of p in $Y_{1,2}^1$ and that of $Y_{5,10}^5$ is 5 times the value of p in $Y_{1,2}^1$. The above relations can be generalized and extended to other coefficients as shown below. The redundancy thus observed between Y_{k_1, k_2}^p of different DFT coefficients is termed as *third level* of redundancy.

$$Y_{(k_1, k_2)^*}^p = -Y_{(k_1, k_2)}^p \text{ for } p \neq 0 \quad (3.7)$$

where $p^* = ((M - p))_M$ and $(k_1, k_2)^* = (((N - k_1))_N, ((N - k_2))_N)$

$$\begin{aligned} & Y_{k_1', k_2'}^p = -Y_{k_1, k_2}^p \text{ if } ((k, p))_N \geq M \\ \text{else} & Y_{k_1', k_2'}^p = Y_{k_1, k_2}^p \end{aligned} \quad (3.8)$$

where $p' = ((p, k))_M$, $k_1' = ((k, k_1))_N$, $k_2' = ((k, k_2))_N$ and $\gcd(k, N) = 1$, $0 < k < M$.

$$Y_{(k_1', k_2')^*}^{(p)'} = -Y_{(k_1', k_2')}^{(p)'} \quad (3.9)$$

When $p = 0$,

$$Y_{(k_1, k_2)}^{(p)} = Y_{(k_1, k_2)^*}^{(p)*} = Y_{(k_1', k_2')}^{(p')} = Y_{(k_1', k_2')^*}^{(p')*} \quad (3.10)$$

Table 3.2 shows the indices for which the coefficients could be derived from one another for $N = 20$. In the table $k_1^* = ((N - k_1))_N$, $k_2^* = ((N - k_2))_N$ and $k = 3, 7$ and 9 . If Y_{k_1, k_2}^p for all values of p of the DFT coefficient with index pair $(1, 0)$ is available, then the DFT coefficients with index pairs as shown along the same row in the table 3.2 could be derived. There is lot of such redundancy in Y_{k_1, k_2}^p . Due to the above redundancy, only one of the DFT coefficients from among them named ‘basic DFT coefficient’ need be calculated and other coefficients could be derived.

Theorem 3.2

$Y_{((N-k_1))_N, ((N-k_2))_N}^{((M-p))M}$ could be derived from that of Y_{k_1, k_2}^p with a sign reversal when $p \neq 0$. When $p = 0$, $Y_{((N-k_1))_N, ((N-k_2))_N}^{((M-p))M} = Y_{k_1, k_2}^p$.

Proof

In (1.7) let $k_1^* = N - k_1$, $k_2^* = N - k_2$ and $p^* = M - p$. Then

$$z = ((n_1(N - k_1) + n_2(N - k_2)))_N = ((M - p))_N \text{ or } ((M - p + \frac{N}{2}))_N$$

$$\text{i.e. } z = ((n_1.k_1 + n_2.k_2))_N = (p + M) \text{ or } (p) \quad (3.11)$$

In (3.11) data is to be added when $z = p + M$ and subtracted when $z = p$ and hence the sign reversal.

$$\therefore Y_{k_1^*, k_2^*}^{p^*} = -Y_{k_1, k_2}^p$$

When $p = 0$, then from above,

$$Y_{k_1^*, k_2^*}^0 = Y_{k_1, k_2}^0 \quad (3.12)$$

Theorem 3.3

$Y_{((k.k_1))_N, ((k.k_2))_N}^{((k.p))M}$ could be derived from Y_{k_1, k_2}^p , with a sign reversal if $((k.p))_N \geq M$, else no sign reversal, where $\gcd(k, N) = 1$, $0 < k < M$. When $p = 0$, $Y_{((k.k_1))_N, ((k.k_2))_N}^{((k.p))M} = Y_{k_1, k_2}^p$.

Proof

In (1.7) let $k_1' = k.k_1$ and $k_2' = k.k_2$ then,

$$z = ((n_1.k_1' + n_2.k_2'))_N = ((k((n_1.k_1 + n_2.k_2)))_N)_N = ((k.p))_N \text{ or } ((k(p + \frac{N}{2})))_N \quad (3.13)$$

Case 1: $\gcd(k, N) = 1$

$$(3.13) \text{ becomes } z = ((k((n_1.k_1 + n_2.k_2)))_N)_N = ((k.p))_N \text{ or } (((kp))_N + \frac{N}{2})_N \quad (3.14)$$

When $((k.p))_N \geq M$, say $M + s$, where $0 \leq s \leq M - 1$, R.H.S of (3.14) becomes $M + s$ or s .

Therefore data is to be added when $z = M + s$ and subtracted when $z = s$ and hence the sign reversal.

Table 3.2: Index relation for $N = 20$

k, k_*	k_*, k_*	$3k, 3k_*$	$3k_*, 3k_*$	$7k, 7k_*$	$7k_*, 7k_*$	$9k, 9k_*$	$9k_*, 9k_*$
1 0	19 0	3 0	17 0	7 0	13 0	9 0	11 0
0 1	0 19	0 3	0 17	0 7	0 13	0 9	0 11
1 1	19 19	3 3	17 17	7 7	13 13	9 9	11 11
2 1	18 19	6 3	14 17	14 7	6 13	18 9	2 11
3 1	17 19	9 3	11 17	1 7	19 13	7 9	13 11
4 1	16 19	12 3	8 17	8 7	12 13	16 9	4 11
5 1	15 19	15 3	5 17	15 7	5 13	5 9	15 11
6 1	14 19	18 3	2 17	2 7	18 13	14 9	6 11
7 1	13 19	1 3	19 17	9 7	11 13	3 9	17 11
8 1	12 19	4 3	16 17	16 7	4 13	12 9	8 11
9 1	11 19	7 3	13 17	3 7	17 13	1 9	19 11
10 1	10 19	10 3	10 17	10 7	10 13	10 9	10 11
11 1	9 19	13 3	7 17	17 7	3 13	19 9	1 11
12 1	8 19	16 3	4 17	4 7	16 13	8 9	12 11
13 1	7 19	19 3	1 17	11 7	9 13	17 9	3 11
14 1	6 19	2 3	18 17	18 7	2 13	6 9	14 11
15 1	5 19	5 3	15 17	5 7	15 13	15 9	5 11
16 1	4 19	8 3	12 17	12 7	8 13	4 9	16 11
17 1	3 19	11 3	9 17	19 7	1 13	13 9	7 11
18 1	2 19	14 3	6 17	6 7	14 13	2 9	18 11
19 1	1 19	17 3	3 17	13 7	7 13	11 9	9 11
1 2	19 18	3 6	17 14	7 14	13 6	9 18	11 2
3 2	17 18	9 6	11 14	1 14	19 6	7 18	13 2
5 2	15 18	15 6	5 14	15 14	5 6	5 18	15 2
7 2	13 18	1 6	19 14	9 14	11 6	3 18	17 2
9 2	11 18	7 6	13 14	3 14	17 6	1 18	19 2
1 4	19 16	3 12	17 8	7 8	13 12	9 16	11 4
3 4	17 16	9 12	11 8	1 8	19 12	7 16	13 4
5 4	15 16	15 12	5 8	15 8	5 12	5 16	15 4
7 4	13 16	1 12	19 8	9 8	11 12	3 16	17 4
9 4	11 16	7 12	13 8	3 8	17 12	1 16	19 4
1 5	19 15	3 15	17 5	7 15	13 5	9 5	11 15
2 5	18 15	6 15	14 5	14 15	6 5	18 5	2 15
3 5	17 15	9 15	11 5	1 15	19 5	7 5	13 15
4 5	16 15	12 15	8 5	8 15	12 5	16 5	4 15
1 10	19 10	3 10	17 10	7 10	13 10	9 10	11 10
2 0	18 0	6 0	14 0				
4 0	16 0	12 0	8 0				
0 2	0 18	0 6	0 14				
2 2	18 18	6 6	14 14				
4 2	16 18	12 6	8 14				
8 2	12 18	4 6	16 14				
6 2	14 18	18 6	2 14				
10 2	10 18	10 6	10 14				
12 2	8 18	16 6	4 14				
14 2	6 18	2 6	18 14				
16 2	4 18	8 6	12 14				
18 2	2 18	14 6	6 14				
0 4	0 16	0 12	0 8				
2 4	18 16	6 12	14 8				
4 4	16 16	12 12	8 8				
6 4	14 16	18 12	2 8				
8 4	12 16	4 12	16 8				
10 4	10 16	10 12	10 8				
12 4	8 16	16 12	4 8				
14 4	6 16	2 12	18 8				
16 4	4 16	8 12	12 8				
18 4	2 16	14 12	6 8				
2 10	18 10	6 10	14 10				
4 10	16 10	12 10	8 10				
5 0	15 0						
0 5	0 15						
5 5	15 15						
10 5	10 15						
15 5	5 15						
5 10	15 10						

0 0									
10 0									
0 10									
10 10									

Case 2: $\gcd(k, N) \neq 1$

In (11) let $k = \alpha \cdot dm$, where ‘ dm ’ is any divisor of M and $0 \leq \alpha \leq \frac{M}{dm} - 1$ then,

$$z = ((((\alpha \cdot dm))_N((n_1 \cdot k_1 + n_2 \cdot k_2)))_N) \quad (3.15)$$

z in (3.15), is always an integral multiple of ‘ dm ’ and since ‘ dm ’ is a divisor of M , Y_{k_1, k_2}^p exists only for $\gcd(p, M) = dm$. But Y_{k_1, k_2}^p exists for all values of p , $0 \leq p \leq M - 1$ and thus proved.

In the Table 3.2 where $N = 20$, $dm = 1, 2, 5$ and 10 . Each of the coefficients in the first 12 rows, of the table 3.2, could derive 7 other coefficients in the same row, where $\gcd(k_1, k_2, M) = 1$. In the next four rows, each of the coefficient could derive 3 other coefficients in the same row, where $\gcd(k_1, k_2, M) = 2$; in the next three rows, each of the coefficient could derive 1 coefficient, where $\gcd(k_1, k_2, M) = 5$ and in the next two rows, the coefficient could not derive any other coefficients, where $\gcd(k_1, k_2, M) = 10$. Hence the number of DFT coefficients that could be derived by permutation over p depends on dm . When $\gcd(k_1, k_2, M) = dm$, the number of coefficients that could be derived is given by $\varphi\left(\frac{N}{dm}\right) - 1$ where, φ is the Euler Totient function [62] as defined in B.8. Number of DFT coefficients, ‘ nd_{dm} ’, that could be derived from a particular basic DFT coefficient, where $\gcd(k_1, k_2, M) = dm$ for different N is shown in table 3.3. E.g., when $N = 20$, we need compute only 70 DFT coefficients out of 400, as in table 3.2.

Table 3.3: nd_{dm} that could be derived from the basic DFT coefficients for different N

$dm \backslash N$	$nd_{dm} = \varphi\left(\frac{N}{dm}\right) - 1$, when $\gcd(k_1, k_2, M) = dm$									
	1	2	3	4	5	6	7	8	9	10
4	1	0	-	-	-	-	-	-	-	-
6	1	-	0	-	-	-	-	-	-	-
8	3	1	-	0	-	-	-	-	-	-
10	3	-	-	-	0	-	-	-	-	-
12	3	1	1	-	-	0	-	-	-	-
14	5	-	-	-	-	-	0	-	-	-
16	7	3	-	1	-	-	-	0	-	-
18	5	-	1	-	-	-	-	-	0	-
20	7	3	-	-	1	-	-	-	-	0

Table 3.3 shows that the number of coefficients that could be derived from the basic DFT coefficient decreases, as the value of dm increases for a particular N . This in turn increases ‘the number of basic DFT coefficients, nb ’ for the N . The number of basic DFT coefficients also increases as the number of dm increases. Hence the number of basic DFT coefficients depends on N . $\varphi(N/dm)$ is the redundancy factor for a particular basic DFT coefficient, since $\varphi(\frac{N}{dm}) - 1$ coefficients could be derived from any DFT coefficients whose $\gcd(k_1, k_2, M) = dm$.

Redundancy so far seen is in the visual representation of DFT coefficients within an N . On comparison of visual representation of different N , it is seen that the visual representation of lower orders are contained in that of higher orders. E.g., from fig. 3.2 and 3.4, it is seen that $Y_{2,2}^2$ of $N = 8$ contains the visual representation of $Y_{1,1}^1$ of $N = 4$ and repeated 4 times. Similarly $Y_{4,4}^2$ of $N = 12$ contains the visual representation of $Y_{2,2}^1$ of $N = 6$ and repeated 4 times. $Y_{1,1}^1$ of $N = 4$ is seen repeated 9 times in the visual representation of $Y_{3,3}^3$ of $N = 12$. This is illustrated with the following theorems.

Theorem 3.4

Y_{k_1, k_2}^p of N contains the visual representation of $Y_{\frac{k_1}{dm}, \frac{k_2}{dm}}^{\frac{p}{dm}}$ of N/dm where $\gcd(k_1, k_2, M) = dm$

and repeated dm^2 times.

Proof

Let $\gcd(k_1, k_2, M) = dm$, $k_1 = dm.k_1'$, $k_2 = dm.k_2'$, $p = dm.p'$ and $M = dm.M'$. Then from (1.7),

$$((n_1.dm.k_1' + n_2.dm.k_2'))_N = p'.dm \text{ or } (p' + M')dm$$

$$\text{i.e., } ((dm((n_1.k_1' + n_2.k_2'))_N))_N = p'.dm \text{ or } (p' + M')dm$$

Then from theorem B.4.1,

$$((n_1.k_1' + n_2.k_2'))_{\frac{N}{dm}} = p' \text{ or } p' + M' \quad (\because \gcd(dm, N) = dm).$$

Theorem 3.5

$Y_{c.k_1, c.k_2}^{c.p}$ of N contains the visual representation of Y_{k_1, k_2}^p of N/d where $d = \gcd(c, N)$.

Proof

Let $k_1 = c.k_1$ and $k_2 = c.k_2$ in (1.7) then,

$$((n_1.c.k_1 + n_2.c.k_2))_N = c.p \text{ or } c(p + M)$$

$$\text{i.e., } ((c((n_1.k_1 + n_2.k_2))_N))_N = c.p \text{ or } c(p + M).$$

Then from B.4.1, if $d = \gcd(c, N)$

$$((n_1.k_1 + n_2.k_2))_{N/d} = p \text{ or } p + M.$$

E.g., $Y_{6,6}^6$ of $N = 20$ contains the visual representation of $Y_{1,1}^1$ of $N = 10$ and repeated 4 times. It can be easily verified that the visual representation of $N = 4$ is contained in the visual representation of $N = 8$ as well as in $N = 12$. It can be inferred from theorem 3.6 that the visual representation of the DFT coefficient for an order N will contain the representation of all N/d such that $d \mid N$ and $2^n \parallel N$ but $2^n \nmid d$.

3.2.5 Calculation of number of basic DFT coefficients

For every dm , the total number of DFT coefficients whose $\gcd(k_1, k_2, M) = dm$ could be calculated. As discussed in section 3.2.4, $\varphi(N/dm)$ is the redundancy factor for a particular basic DFT coefficient. Hence the number of basic DFT coefficients whose $\gcd(k_1, k_2, M) = dm$ could be obtained by dividing the total number of DFT coefficients whose $\gcd(k_1, k_2, M) = dm$ by the redundancy factor. The number of basic DFT coefficients whose $\gcd(k_1, k_2, M) = dm$ is given by

$$nb_{dm} = npt_{dm} / \varphi(N/dm) \quad (3.16)$$

where npt_{dm} is the total number of DFT coefficients whose $\gcd(k_1, k_2, M) = dm$.

Summation of all “ nb_{dm} ” over all divisors, dm of M will give the total number of basic DFT coefficients (nb) for any even N ; i.e.,

$$nb = \sum_{dm} nb_{dm} = \sum_{dm} \frac{npt_{dm}}{\varphi\left(\frac{N}{dm}\right)}. \quad (3.17)$$

The total number of DFT coefficients (npt_{dm}) whose $\gcd(k_1, k_2, M) = dm$ can be computed using Principle of Inclusion-Exclusion [2], as defined in B.5, is explained below.

When $\gcd(k_1, k_2, M) = M$, k_1 and k_2 can have either of the values 0 or M , i.e., $N/M = 2$ values. From the definition of permutations, the number of permutations of 2 things, taken ‘2’ at a time, when each of them can be repeated 2 times is given by $2^2 = 4$; i.e., $(N/M)^2$. Here the coefficients are $(0, 0)$, $(0, M)$, $(M, 0)$ and (M, M) .

Now assume that $\gcd(k_1, k_2, M) = M/2$. Then both of the indices k_1 and k_2 can have the values 0 , $M/2$, M and $3M/2$, but cannot be 0 and M together, i.e., k_1 and k_2 have values except those for which $\gcd(k_1, k_2, M) = M$. The number of permutations of 4 things, taken 2 at a time, when each of them can be repeated 2 times is 4^2 , i.e., $(N/M/2)^2$. The above permutations include $(0, 0)$, $(0, M)$, $(M, 0)$ and (M, M) , i.e., the permutations for which $\gcd(k_1, k_2, M) = M$, which has to be excluded. The exclusion is necessary since M and $M/2$ have common multiples in the range 0 to $N - 1$. Therefore the number of DFT coefficients whose $\gcd(k_1, k_2, M)$ is $M/2 = (N/M/2)^2 - (N/M)^2 = 4^2 - 2^2 = 12$.

i.e., the coefficients are $(0, M/2)$, $(0, 3M/2)$, $(M/2, 0)$, $(M/2, M/2)$, $(M/2, M)$, $(M/2, 3M/2)$, $(M, M/2)$, $(M, 3M/2)$, $(3M/2, 0)$, $(3M/2, M/2)$, $(3M/2, M)$ and $(3M/2, 3M/2)$.

In general, the total number of DFT coefficients, when $\gcd(k_1, k_2, M) = dm$, is obtained by subtracting the number of permutations of indices whose $\gcd(k_1, k_2, M) = dm_o$, where $\gcd(dm_o, dm) = dm$ and $dm_o > dm$, from $(N/dm)^2$. Let npt_{dm} is the number of permutations of (k_1, k_2) when $\gcd(k_1, k_2, M) = dm$, then

$$npt_{dm} = (N/dm)^2 - \sum_{dm_o} npt_{dm_o}$$

This can be illustrated with an example.

When $N = 24$, $M = 12$ and the divisors dm of M are 12, 6, 4, 3, 2, and 1.

$$npt_{12} = (N/dm)^2 = (24/12)^2 = 4$$

$$npt_6 = (N/dm)^2 - npt_{12} = (24/6)^2 - (24/12)^2 = 12$$

$$npt_4 = (N/dm)^2 - npt_{12} = (24/4)^2 - (24/12)^2 = 32$$

$$npt_3 = (N/dm)^2 - npt_6 - npt_{12} = (N/3)^2 - ((N/6)^2 - npt_{12}) - npt_{12} = (N/3)^2 - (N/6)^2 = 64 - 16 = 48$$

$$\begin{aligned} npt_2 &= (N/dm)^2 - npt_4 - npt_6 - npt_{12} = (N/2)^2 - ((N/4)^2 - npt_{12}) - ((N/6)^2 - npt_{12}) - npt_{12} \\ &= (N/2)^2 - (N/4)^2 - (N/6)^2 + (N/12)^2 = 12^2 - 6^2 - 4^2 + 2^2 = 96 \end{aligned}$$

$$\begin{aligned} npt_1 &= (N/dm)^2 - npt_2 - npt_3 - npt_4 - npt_6 - npt_{12} = (N/1)^2 - (N/2)^2 - (N/3)^2 + (N/6)^2 \\ &= 576 - 144 - 64 + 16 = 384 \end{aligned}$$

In the expression for npt_{12} , npt_6 , npt_4 and npt_3 , there is only exclusion of common indices, whereas in npt_2 , $(N/2)^2$ is the number of DFT coefficients whose indices k_1 and k_2 is even. This set include those coefficients for which $\gcd(k_1, k_2, M) = 4$ and 6 which have to be excluded. $(N/4)^2$ is the number of coefficients whose k_1 and k_2 are divisible by 4 and $(N/6)^2$ is the number of coefficients whose k_1 and k_2 are divisible by 6. Both of them have common elements since 0 and 12 are divisible by both 4 and 6. While excluding those coefficients whose $\gcd(k_1, k_2, M) = 4$ and 6, the common elements are subtracted twice. Hence the intersection of the set of the coefficients for which $\gcd(k_1, k_2, M) = 4$ and 6 has to be included, i.e., $(N/12)^2$. Similarly, both inclusion and exclusion occurs in npt_1 also.

Since $\varphi(N/12) = 1$, $\varphi(N/6) = 2$, $\varphi(N/4) = 2$, $\varphi(N/3) = 4$, $\varphi(N/2) = 4$, and $\varphi(N/1) = 8$, the number of basic DFT coefficients whose $\gcd(k_1, k_2, M) = dm$ can be calculated using (3.16).

$$nb_{12} = npt_{12} / \varphi(N/12) = 4$$

Similarly, $nb_6 = 6$, $nb_4 = 16$, $nb_3 = 12$, $nb_2 = 24$, and $nb_1 = 48$.

From (3.17), total number of basic DFT coefficients

$$nb = \sum_{dm} nb_{dm} = 4 + 6 + 16 + 12 + 24 + 48 = 110.$$

The number of basic DFT coefficients (nb), for any even N , can be calculated using (3.17). The above results were verified with the visual representations also and found correct.

Tables 3.4 & 3.5 show the number of basic DFT coefficients when $N/2$ is prime and when N is a power of 2 respectively. From the tables, it can be seen that the number of basic DFT coefficients is $2.N + 8$ when $N/2$ is prime whereas when N is a power of 2, it is $3.N - 2$.

Table 3.4: nb when $N/2$ is prime

N	6	10	14	22	26	34	38	46
nb	20	28	36	52	60	76	84	100

Table 3.5: nb when N is a power of 2

N	4	8	16	32	64	128
nb	10	22	46	94	190	382

Fig. 3.5 shows the plot of the number of basic DFT coefficients required to be computed to obtain the entire coefficients for different N . In the plot, the overshoots are due to more number of dm for the corresponding N , w.r.t. the near by N 's. E.g., when N is any number like 12, 24, 36, 48 and 60, which are having more number of divisors, the sharp overshoot can be noticed.

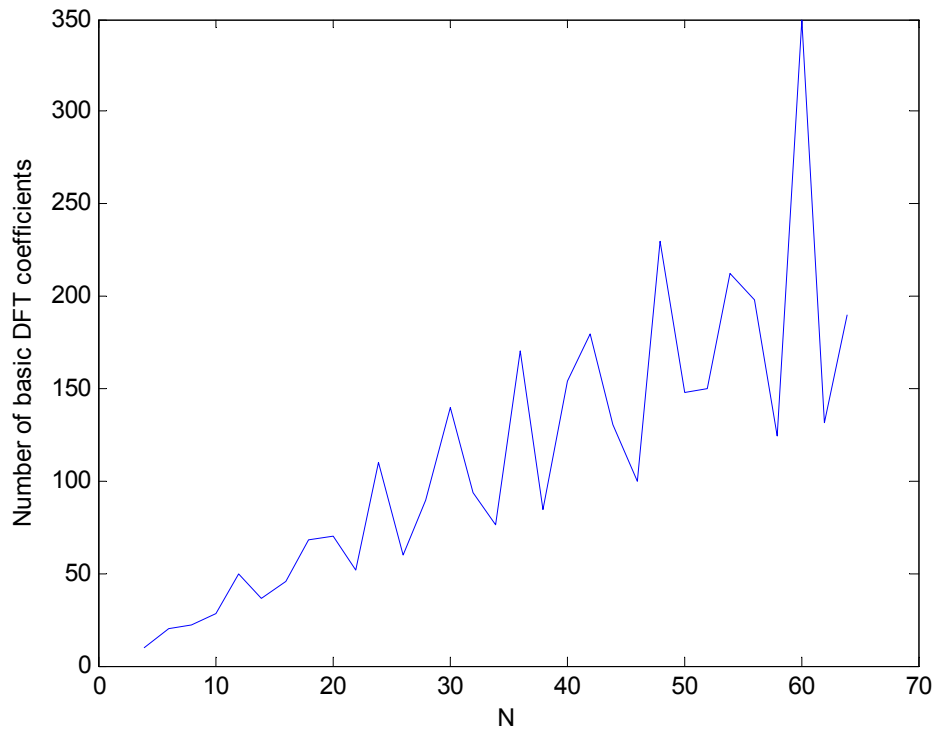


Fig. 3.5: nb for different N

3.2.6 Algorithm for computing the index of all basic DFT coefficients

The nb basic DFT coefficients become the true representatives of the entire N^2 DFT coefficients and they are to be identified. On analysis of the visual representation of DFT coefficients for $N = 4$ to 64, as done in section 3.2.4, and from (3.10) it is observed that the basic DFT coefficients can be identified from Y_{k_1, k_2}^p for $p = 0$ and that the same can be selected in three different ways namely (i) column wise, (ii) row wise, and (iii) quadrant wise.

(i) Column wise

In this method, selection of basic DFT coefficients is done from each column starting for column 0, row 0. Each of the coefficients is verified to see whether it matches with the already selected basic DFT coefficients; if so discard it, else it is included as one of the basic DFT coefficients. Once column 0 is completed, column 1 is verified, followed by other columns till the entire $N - 1$ columns were verified. Once the selection is completed, each of the DFT coefficients in the set of basic DFT coefficients is unique.

(ii) Row wise

It uses the same method as that in the column wise method except for the fact that, here selection was done from each row starting from row 0. On verification of the selected basic DFT coefficients based on this method show that the same can be obtained by exchanging k_1 and k_2 of that of column wise selection.

(iii) Quadrant wise

In this method, basic DFT coefficients are selected from one quadrant at a time starting from quadrant 1, followed by other quadrants.

On analysis of the indices of the basic DFT coefficients obtained from the column wise method, following observations were significant:

1. Basic DFT coefficients are selected from column numbers (implies k_2) which are *divisors mod N* of N . E.g., when $N = 12$, basic DFT coefficients are selected from columns 1, 2, 3, 4, 6 and 0.
2. In columns 0 and M , the basic DFT coefficients selected have k_1 which are *divisors mod N* of N . E.g., when $N = 12$, the frequency index k_1 , of basic DFT coefficients in column 0 and M , are 1, 2, 3, 4, 6 and 0.
3. If the index k_1 and k_2 of the basic DFT coefficients in column 0 are interchanged, index of basic DFT coefficients in row 0 will be obtained. E.g., when $N = 12$, basic DFT coefficients

in column 0 are (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), and (6, 0) where as that of row 0 are (0, 0), (0, 1), (0, 2), (0, 3), (0, 4) and (0, 6).

4. In column number $k_2 = 2.dm$ where $\gcd(2.dm, N) = 2.dm$ and $\gcd(2.dm, M) = dm$, the number of basic DFT coefficients and the corresponding frequency index k_1 will be same as that of column number $k_2/2 = dm$. E.g., when $N = 12$, the number of basic DFT coefficients and the corresponding frequency index k_1 in column numbers $((N))_N = 0$ and M are same. Similarly the number of basic DFT coefficients and the corresponding frequency index k_1 in column numbers 4 and 2 are same.
5. When $N/2$ is prime, apart from the basic DFT coefficients in column 0 and M , the entire DFT coefficients in column 1 and 2 are required to complete the set of basic DFT coefficients. All other columns are ignored. Since N is having only four divisors namely, 1, 2, M , and N , there are four basic DFT coefficients each in column 0 and M , and N basic DFT coefficients each in column 1 and 2 resulting in a total of $2N + 8$ basic DFT coefficients.
6. When N is a power of 2, in each column where $k_2 = dm$, select all DFT coefficients where $\gcd(k_1, k_2) = dm$. E.g., when $N = 16$, in column 4, (0, 4), (4, 4), (8, 4), and (12, 4), are basic DFT coefficients. Then for each divisors ' d_{dm} ', of dm other than dm , select the first $N/(2.dm)$ frequency index k_1 , where $\gcd(k_1, d_{dm}) = d_{dm}$. E.g., when $N = 16$, in column 4, 2 is a divisor of 4 and so select the first $N/(2 \times 4)$, i.e., 2 coefficients whose $\gcd(k_1, 2) = 2$ are (2, 4) and (6, 4). Similarly 1 is a divisor of 4 and so select the first two coefficients whose $\gcd(k_1, 1) = 1$, namely, (1, 4) and (3, 4).

In columns where $k_2 = ((2.dm))_N$, where $\gcd(2.dm, M) = dm$ and $\gcd(2.dm, N) = 2.dm$, the number of basic DFT coefficients and the frequency index k_1 is same as that of column $k_2 = dm$. E.g., when $N = 16$, there are five basic DFT coefficients namely, (0, 8), (1, 8), (2, 8), (4, 8), and (8, 8) in column 8 and five basic DFT coefficients namely, (0, 0), (1, 0), (2, 0), (4, 0), and (8, 0) in column $((16))_{16} = 0$.

7. However in general, in each column where $k_2 = dm$, all the DFT coefficients from (0, dm) to $(N/dm, dm)$ are included in the basic set of DFT coefficients. To find the basic DFT coefficients from the rest of the coefficients in that column, find all k which are co-prime to N , as defined in B.7, where $k = 1 + q.N/dm$ where q is any positive integer. Exclude all k_1 which are k . Select all coefficients whose frequency index $k_1 = ((k_1.k))_N$ and $((k_1.k))_N$ not equal to any of the k_1 already selected in that column. E.g., when $N = 12$, in column $k_2 = 3$, first select all DFT coefficients from (0, 3) to (4, 3). Now calculate all $k = 1 + q.N/dm$. The value of k is 5. So exclude (5, 3). Now when $k_1 = 6$, $((k_1.k))_N = 6$, therefore (6, 3) is a basic

DFT coefficient. When $k_1 = 7$, $((k_1.k))_N = 11$ which is not k_1 , but is not any of the k_1 already selected, so select (7, 3). Similarly we can find that (9, 3) is also a basic DFT coefficient, whereas (8, 3), (10, 3) and (11, 3) are not.

The above analysis has been used to derive an algorithm to compute the index of basic DFT coefficients for any even N .

1. Calculate the divisors 'dm' of M .
2. For each column where $k_2 = dm$
 - i) Select all coefficients from $(0, dm)$ to $((N/dm)_N, dm)$
 - ii) Find all $k = 1 + q.N/dm$ such that $\gcd(k, N) = 1$ and $k < N$
 - iii) For $k_1 = N/dm + 1$ to $N - 1$
 - a. For every k , if $k_1 = k$ exclude it and repeat step (a) for next k_1
 - b. Else calculate $((k.k_1))_N$
If $k_1 = ((k.k_1))_N$ select k_1 and
repeat step (a) for next k_1
Else if $((k_1.k))_N$ is equal to any of the k_1 already selected,
then discard k_1 and repeat step (a) for next k_1 ;
or else repeat step (b) with the next k .
3. Find $k_2 = ((2.dm))_N$ such that $\gcd(2.dm, M) = dm$ and $\gcd(2.dm, N) = 2.dm$.
4. For each column, where $k_2 = ((2.dm))_N$, calculated in step 3.,
the basic DFT coefficients will have same k_1 as that of $k_2 = dm$.
So copy k_1 from $k_2 = dm$ to that of $k_2 = ((2dm))_N$.

3.2.7 Patterns in basic DFT coefficients

Visual representation of $Y_{k_1.k_2}^p$ corresponding to the basic DFT coefficients has a definite pattern in the data involved in its computation. Fig. 3.6, 3.7, and 3.8 shows the visual representation of $Y_{k_1.k_2}^p$ corresponding to the basic DFT coefficients for $N = 4, 6$, and 8 respectively. Figures show that, computation of $Y_{0,0}^0$ involves the sum of all data for any N . Similarly for the computation of $Y_{1,0}^0$ of $N = 4, 6$ and 8 , all data in the 0^{th} row is involved with a plus sign and all data in the M^{th} row is involved with a minus sign. Here k_1 is a divisor 'dm' of M for $N = 4, 6$ and 8 . This can be generalized for any N as follows:

For every $k_1 = dm$, all the data in the rows $(-1)^k \frac{k.M}{dm}$, $0 \leq k \leq 2dm - 1$ are involved in the computation of $Y_{dm,0}^0$.

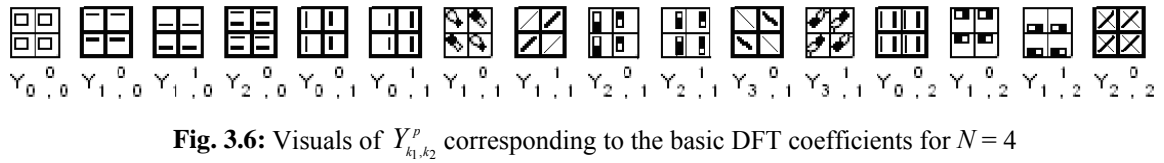


Fig. 3.6: Visuals of Y_{k_1, k_2}^p corresponding to the basic DFT coefficients for $N = 4$

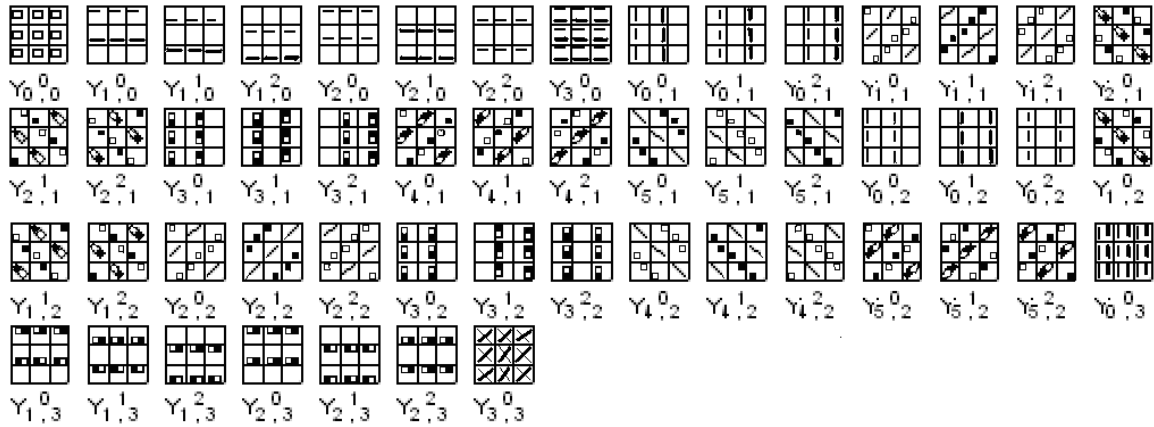


Fig. 3.7: Visuals of Y_{k_1, k_2}^p corresponding to the basic DFT coefficients for $N = 6$

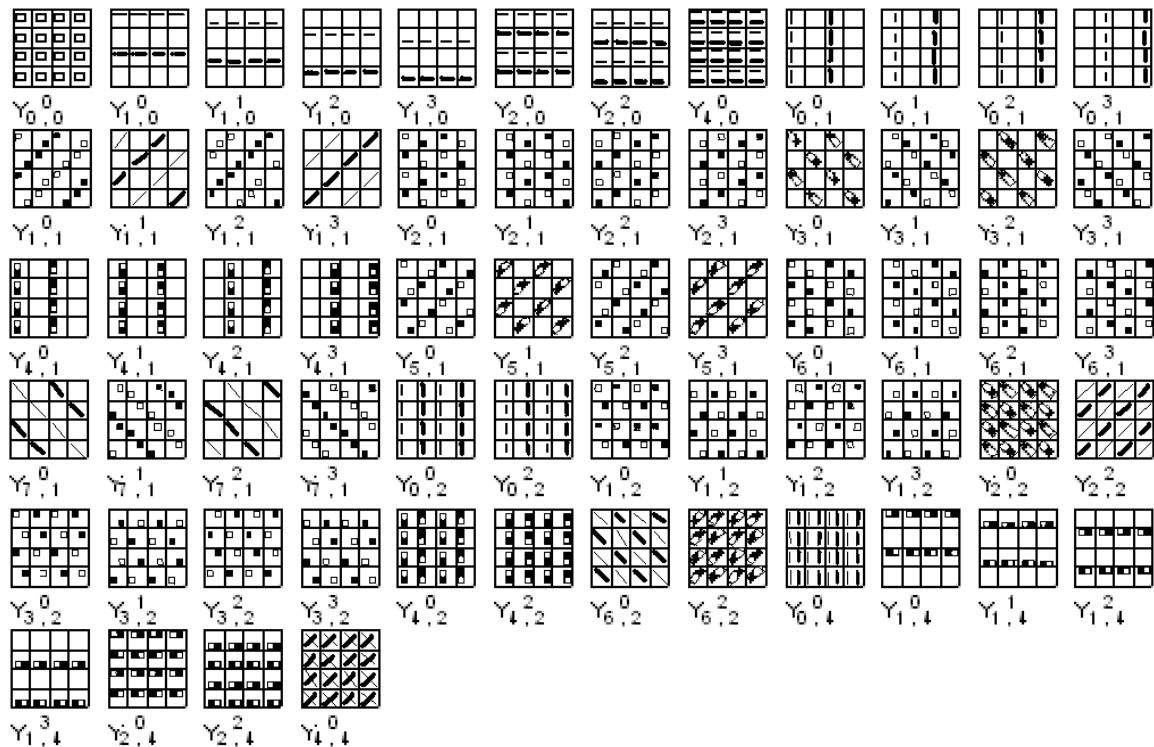


Fig. 3.8: Visuals of Y_{k_1, k_2}^p corresponding to the basic DFT coefficients for $N = 8$

For $Y_{2,0}^0$ of $N = 6$, all the data in the 0th row and M^{th} row are involved with a plus sign, whereas for $N = 8$, all data in the 0th row and M^{th} row involved with a plus sign and data in the 2nd

and $(2 + M)^{\text{th}}$ row involved with a minus sign. Here $k_l = 2.dm$ for $N = 6$, whereas $k_l = dm$ for $N = 8$. Therefore for $Y_{2,0}^0$ of $N = 8$, data of rows $(-1)^k \frac{k.M}{dm}$, $0 \leq k \leq 2dm - 1$ are involved. For $Y_{2,0}^0$ of $N = 6$, since $k_l = 2.dm$, rows $\frac{k.M}{dm}$, $0 \leq k \leq 2dm - 1$ are involved. Thus for every $k_l = 2.dm$ all the data in the rows $\frac{k.M}{dm}$, $0 \leq k \leq 2dm - 1$, are involved if $\gcd(k_l, N) = 2.dm$ and $\gcd(k_l, M) = dm$, to obtain all $Y_{2.dm,0}^0$.

Visual representation of $Y_{0,1}^0$ of $N = 4, 6$, and 8 can be obtained by rotating the visual representation of respective $Y_{1,0}^0$ by 90° . It is equivalent to interchanging k_l and k_2 , i.e., for every $k_2 = dm$, all the data in the columns $(-1)^k \frac{k.M}{dm}$, $0 \leq k \leq 2dm - 1$ are involved to obtain all $Y_{0,dm}^0$ and for every $k_2 = 2.dm$ all the data in the columns $\frac{k.M}{dm}$, $0 \leq k \leq 2dm - 1$ are involved if $\gcd(k_2, N) = 2.dm$ and $\gcd(k_2, M) = dm$, to obtain all $Y_{0,2.dm}^0$.

Now on analysis of visual representation of all $Y_{k_1, dm}^0$ in a column, it is noticed that the data involved in the computations differ from one another, as there is circular shifts of the data rows. These circular shifts are in a specific pattern. E.g., when $N = 8$ in fig. 3.8, the data involved in the computation of $Y_{1,1}^0$ and $Y_{0,1}^0$ differ. Even though same data rows are involved in their computations, each data row in $Y_{1,1}^0$ is circularly shifted to the left k_l times its row number (i.e., left shift of data row 0 by 0, row 1 by 1 time, row 2 by 2 times etc.). Here k_2 is a divisor of M . The data row involved in the computation depends on k_l and k_2 . E.g., when $N = 8$ in fig. 3.8, for the computation of $Y_{1,2}^0$, data in the rows 0, 2, 4, and 6 are involved whereas for $Y_{0,2}^0$ data in all the rows are involved. It can also be noticed that there is a circular shift left in the data rows involved in the computation of $Y_{1,2}^0$, i.e., left shift of data row 0 by 0 time, row 2 by 1 time, row 4 by 2 times and row 6 by 3 times. When $k_2 = 2.dm$, where $\gcd(k_2, N) = 2.dm$ and $\gcd(k_2, M) = dm$, the number of shift differs.

There is a specific pattern change in Y_{k_1, k_2}^p within a basic DFT coefficient. E.g., in fig. 3.8, where $N = 8$, all the data in the 0^{th} row are involved with a plus sign and all data in the M^{th} row are involved with a minus sign for the computation of $Y_{1,0}^0$. But for $Y_{1,0}^1$ all the data in the l^{st} row are involved with a plus sign and all data in the $(1 + M)^{\text{th}}$ row are involved with a minus sign, i.e., there

is a shift in the data rows involved in the computation when compared to $Y_{1,0}^0$. Similarly for the computation of $Y_{0,1}^0$ and $Y_{0,1}^1$, there is a shift in the data columns involved.

The above analysis has been used to derive an algorithm for the computation of Y_{k_1,k_2}^p .

3.3 DFT computation using visual method

In the modified 2-D DFT, the computational complexity for each DFT coefficient is reduced from N^2 complex multiplications required in direct DFT to that of $N/2$. In (1.5), Y_{k_1,k_2}^p is computed using a formula derived from the analysis of the visual representation as stated in section 3.2.7, rather than doing the computation in (1.6) and (1.7). Further by exploiting the redundancy present in Y_{k_1,k_2}^p of DFT coefficients as explained in section 3.2.4, Y_{k_1,k_2}^p of only basic DFT coefficients are computed. By permuting Y_{k_1,k_2}^p of basic DFT coefficients, over p as stated in theorem 3.3, the remaining DFT coefficients can be derived. The steps for DFT computation using visual method is shown in fig. 3.9 and the algorithm for important steps are as follows.

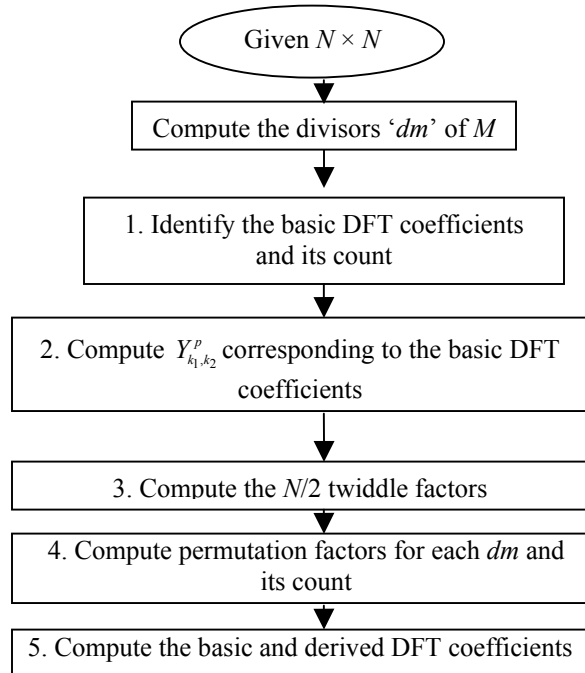


Fig. 3.9: Flow chart depicting the computation of DFT using visual method

1. Compute indices (k_1, k_2) of all the basic DFT coefficients and the no. of basic DFT (*no_of_basicDFT*) as in section 3.2.5. and 3.2.6.
2. Algorithm for computing Y_{k_1,k_2}^p of all the basic DFT coefficients

For $q = 1$ to $no_of_basicDFT$

If $((k2(q)))_M = 0$

$dm = k2(q)$

else

$dm = k2(q)/2$

$l = \gcd(k1(q), dm)$, $v = \gcd(k1(q), k2(q), M)$, $u = \gcd(k1(q), k2(q), N)$

For $p = 0$ to $M - 1$ in steps of v

compute particular solution $(n1, n2)$ using Extended Euclidean algorithm

If $((n1.k1(q) + n2.k2(q)))_N = p$

$ng = 0$

else

$ng = 1$, $ro_limit = N.l/dm - 1$, $col_limit = 2.dm - 1$

For $kr = 0$ to ro_limit

For $kc = 0$ to col_limit

$next_n1 = ((kr.dm/l + n1))_N$

If $((k2(q)))_M = 0$

$next_n2 = (-1)^{(kc + ng)}.((kc.M/dm - kr.k1(q)/l + n2))_N$

else

$next_n2 = (-1)^{(kr.u/v + ng)}.((kc.M/dm + kr.k1(q).(N - 2.dm)/(4.l.dm) + n2))_N$

$Y(k1(q), k2(q), p) = Y(k1(q), k2(q), p) + x(next_n1, next_n2)$

3. Compute $N/2$ twiddle factors

For $p = 0$ to $M - 1$

$w(p) = \exp(-j.2. \pi .p)/N$

4. Compute $\varphi(N/dm)$ and the co-prime integers of every N/dm up to N/dm defined as $coprime.Nbydm(r)$

5. Algorithm for computing all DFT coefficients using permutations and combinations of Y_{k_1, k_2}^p of basic DFT coefficients

For $q = 1$ to $no_ofbasicDFT$

$v = \gcd(k1(q), k2(q), M)$

For $r = 1$ to $\varphi(N/dm)$

$kcp = coprime.Nbydm(r)$

$fk_1 = ((kcp.k1(q)))_N$, $fk_2 = ((kcp.k2(q)))_N$

For $p = 0$ to $M - 1$ in steps of v

$$kpn = ((kcp.p))_N, kpm = ((kcp.p))_M$$

If ($kpn < M$)

$$Y(fk_1, fk_2) = Y(fk_1, fk_2) + Y(k1(q), k2(q), p) \cdot w(kpm)$$

else

$$Y(fk_1, fk_2) = Y(fk_1, fk_2) - Y(k1(q), k2(q), p) \cdot w(kpm)$$

3.4 Conclusion

The visual representation of DFT, in terms of 2×2 data for any even value of N , can be constructed and the same can be used for signal analysis. This representation is quite useful in applications that require only a selected few DFT coefficients. The analysis of visual representation shows specific pattern in the visual representation, depending on the appearance and hence can be used to derive efficient computational schemes. It is shown that the existence of Y_{k_1, k_2}^p depends on the divisors of M and that the number of complex multiplication is not exactly $N/2$, but less than that depending on the divisors. By exploiting the redundancy at various levels, the computational complexity can be reduced.

The modified 2-D DFT representation enables 2-D signal representation and frequency domain analysis in terms of few DFT coefficients dependent on the size of the matrix N . A mathematical relation is developed for the number of basic DFT coefficients depending on N and its validity is verified for different values of N . When N is a power of 2, $3.N - 2$ coefficients need be computed whereas if $N/2$ is prime, there will be only $2.N + 8$ basic DFT coefficients. The number of basic DFT coefficients is higher for N , having more number of factors. The algorithm presented gives a procedure for computing the index values of the basic set of DFT coefficients. The 2-D DFT computation can be simplified using the basic DFT coefficients identified.

The patterns in the basic DFT coefficients have been used to derive an algorithm for its computation. Computation of selected DFT coefficients is possible. The complex multiplication can further be reduced if the signal representation and analysis are done with the basic DFT coefficients alone. The complex multiplication can be avoided, if the signal is represented in terms of the MRT coefficients and then the computation requires only real addition. In such a case, the number of computation will be less, when $N/2$ is prime.

CHAPTER 4

PARALLEL DISTRIBUTED ARCHITECTURE FOR $N \times N$ DFT

It is difficult to have significant improvement in the speed of conventional computers due to the physical limitation imposed by the speed of light. Further improvement in speed, required for real time DSP applications, can be achieved by reducing the number of multiplications and/or by parallel processing. New architectures can be evolved for high speed applications by exploiting parallelism inherent in algorithms, and by employing pipeline techniques. This can be met by one-to-one mapping of algorithm onto multiple processing elements. The motivating factor is that the expected performance improvements in the VLSI technology will essentially come from the ability to fabricate a large number of transistors on a chip. Only a minor contribution will come from increased circuit speed. Hence, it is of importance to develop efficient methods for mapping DSP algorithms onto optimal architectures that efficiently utilizes the parallelism. The performance of dedicated VLSI DSP circuits relies on underlying architectures and implementation styles. Thus it is necessary to design not just a single architecture, but a family of architectures out of which an appropriate architecture can be selected for a specified application [94]. Another way of looking at it is to design new algorithms, which are possessed with concurrency, by keeping architecture and implementations in mind.

Parallel distributed architecture for $N \times N$ point DFT computation where $((N))_4 = 2$ was developed in [71]. This was evolved from the analysis of the visual representation based on 2×2 DFT. In most of the FFT computations in use, the size of N is limited to power of 2. However, this limitation on N is not a natural choice for many of the applications. Thus there is a strong rationale for developing an efficient, high performance, scalable architecture that is suitable for applications in need of DFT sizes that are not necessarily a power of 2 [111]. The parallel distributed architecture developed in [71] is required to be extended to any N to suit such applications. A parallel distributed architecture for $((N))_4 = 0$ if developed, closely following the approach in [71], can be combined with that of $((N))_4 = 2$ to have a generalized architecture. In this direction, to start with a small value of N where $((N))_4 = 0$, the design of an 8×8 point DFT is attempted first.

4.1 Development for 8×8 DFT based on 2×2 DFT

In order to develop a parallel distributed architecture, for the computation of 8×8 point DFT, it is necessary to analyze the visual representation based on 2×2 DFT. The fig. A.2 shows the visual representation of the DFT coefficients for $N = 8$. Only a unique set of Y_{k_1, k_2}^p need be computed from the entire set of Y_{k_1, k_2}^p for $N = 8$, as in section 3.2.4. Visual representation for 64 unique set of Y_{k_1, k_2}^p , corresponding to the 22 basic DFT coefficients, required to be computed is shown in fig. 4.1.

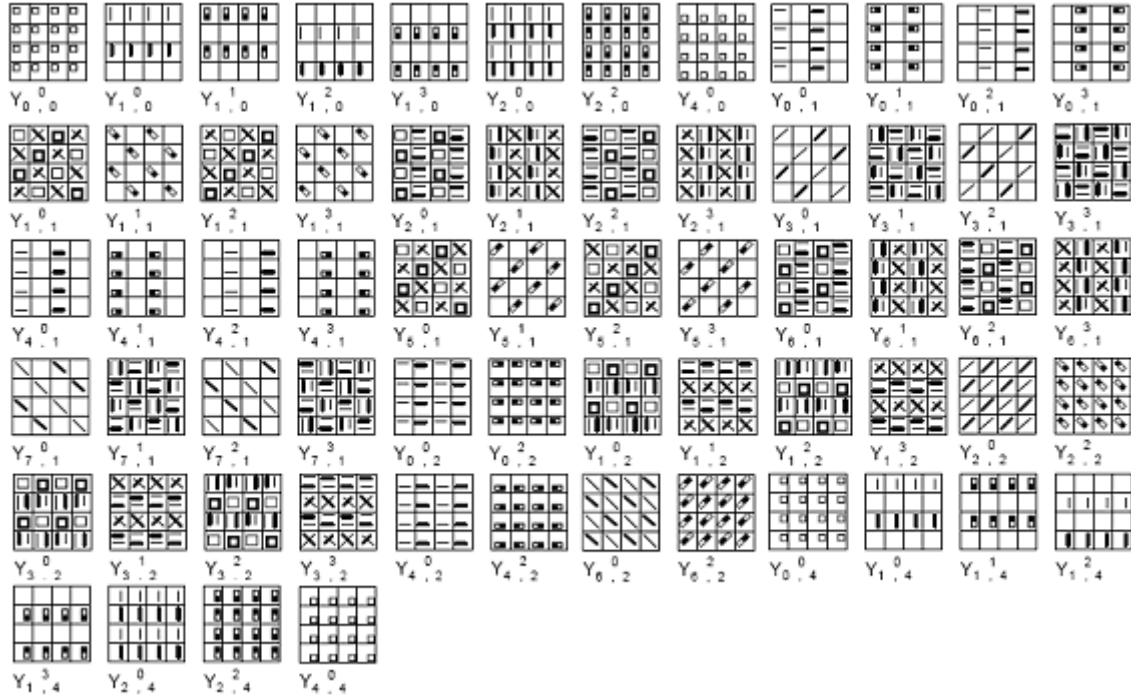


Fig. 4.1: Visual representation of 64 unique set of Y_{k_1, k_2}^p for $N = 8$

The grouping of DFT coefficients correspond to 8×8 point DFT as was done in [88] is shown in fig A.3. The DFT coefficients are classified into three groups depending on the way the primitive symbols are present in them. The circled coefficients (i.e., $Y_{0,0}$, $Y_{0,4}$, $Y_{4,0}$, $Y_{4,4}$) represent group 1. The group 1 coefficients depend on one DFT coefficient from each 2×2 cell. The above group have real coefficients with visual representation only for $p = 0$ and have identical cells. Thus the group 1 coefficients have the same property as that of $((N)_4) = 2$.

The coefficients $Y_{1,0}$, $Y_{2,0}$, $Y_{3,0}$, $Y_{5,0}$, $Y_{6,0}$, $Y_{7,0}$, $Y_{0,1}$, $Y_{0,2}$, $Y_{0,3}$, $Y_{0,5}$, $Y_{0,6}$, $Y_{0,7}$, $Y_{1,4}$, $Y_{2,4}$, $Y_{3,4}$, $Y_{5,4}$, $Y_{6,4}$, $Y_{7,4}$, $Y_{4,1}$, $Y_{4,2}$, $Y_{4,3}$, $Y_{4,5}$, $Y_{4,6}$, $Y_{4,7}$ that are between two dotted lines represent group 2. The group 2 coefficients also show a positional relation as shown in table 4.1. In the table the group 2 coefficients in column 0 will depend on the two coefficients in the left of the cells, i.e., VLP, VLN for even p and VLPA, VLPB for odd p . Similarly group 2 coefficients in column 4 will depend on the two coefficients in the right of the cells, i.e., VRP, VRN for even p and VRPA, VRPB for odd p . In this group, the symbols in one row/column of cells are sufficient to represent

the complete coefficient. The representation for different values of p can be derived by circularly shifting.

Table 4.1: Mnemonics used to represent the DFT coefficients for $N = 8$

Group	DFT coefficients	Even p	Odd p
1	$Y_{0,0}$	LAP ($p = 0$ only)	BLANK
	$Y_{0,4}$	RAP ($p = 0$ only)	BLANK
	$Y_{4,0}$	LBP ($p = 0$ only)	BLANK
	$Y_{4,4}$	RBP ($p = 0$ only)	BLANK
2	Column 0	VLP, VLN	VLPA, VLPB
	Column 4	VRP, VRN	VRPA, VRPB
	Row 0	HAP, HAN	HAPL, HAPR
	Row 4	HBP, HBN	HBPL, HBPR
3	$Y_{1,1}, Y_{3,3}, Y_{5,5}, Y_{7,7}$	MP, MN, MPD, MPC	DPA, DPB
	$Y_{5,1}, Y_{7,3}, Y_{1,5}, Y_{3,7}$	MP, MN, MPD, MPC	CPA, CPB
	$Y_{7,1}, Y_{5,3}, Y_{3,5}, Y_{1,7}$	DP, DN	MPL, MPR, MPA, MPB
	$Y_{3,1}, Y_{1,3}, Y_{7,5}, Y_{5,7}$	CP, CN	MPL, MPR, MPA, MPB
	$Y_{2,2}, Y_{6,6}$	CP, CN for $p = 0$ DPA, DPB for $p = 2$	BLANK
	$Y_{2,6}, Y_{6,2}$	DP, DN for $p = 0$ CPA, CPB for $p = 2$	BLANK

The rest of the coefficients form the group 3. They also show a specific pattern depending on the position of the frequency index. In group 3 coefficients, even though other rows/columns are not identical, they can be obtained by circular shift of one row/column. Table 4.1 also shows the Mnemonics for the visual representation of group 3 coefficients. Thus, when $N = 8$, the pictorial representation corresponding to even p will use one set of primitive symbols and that for odd p will use another set of primitive symbols in groups 2 & 3. The above analysis is used for the development of the parallel distributed architecture for the computation of 8×8 DFT.

4.1.1 Hierarchical computation scheme

The regular pattern present in the visual representation of the DFT coefficients was used to derive a hierarchical computation scheme [71] for the computation of $N \times N$ point DFT when $((N))_4 = 2$. The computation scheme for 8×8 point DFT, as far as possible should follow the above structure, since the aim is to develop a generalized architecture for $N \times N$ point DFT where N is any even integer. The primitive symbols are computed first. Different combinations of the primitive symbols and their circularly shifted versions are formed next to represent one row/column. The

third step is to combine the various row/column of symbols so that a set of the coefficients Y_{k_1, k_2}^p can be obtained for a selected set of (k_1, k_2) . The last step is to combine Y_{k_1, k_2}^p in proper order, scaled by the twiddle factor to obtain the N^2 DFT coefficients. Similar operations are grouped together to enable parallel distributed computation, so that the speed of computation can be improved significantly.

4.1.2 Development of Version I architecture

The 22 DFT coefficients can be classified into three groups, as discussed in section 4.1, each depending on a separate set of primitive symbols. In all the three groups of coefficients, a hierarchy of computation can be derived using the structure available in the visual representation. All the cells are identical in group 1 where as all the rows/columns are identical in group 2. In the case of group 3 coefficients, even though other rows/columns are not identical, they can be obtained by circular shift of one row/column. Also, many coefficients will have the same combination of primitive symbols with different circular shift depending on the frequency index. Thus the DFT coefficients can be obtained in a hierarchy of four levels. In the first level the primitive symbols corresponding to all the 2×2 matrices are calculated. A few primitive symbols are chosen to form a combination representing a row / column of symbols corresponding to a selected set of Y_{k_1, k_2}^p for $p = 0$ and all such combinations are computed in the second level. In the third level, proper combination of rows / columns of primitive symbols are chosen to obtain the selected set of Y_{k_1, k_2}^p , $0 \leq p \leq M - 1$. The output of the fourth level will be the complete set of DFT coefficients, derived from the weighted sum of few Y_{k_1, k_2}^p and a set of pre-computed twiddle factor values.

The block schematic of the model is given in fig. 4.2. Layer L1 is similar to that used in the parallel distributed architecture for $(N)_4 = 2$. In layer L1, all the cell-planes are of dimension 4×4 . The cell-planes are named after the operations performed to generate them from the partitioned 2×2 data matrices. In layer L2, the group L2G1 consists of 4 cell planes of dimension 4×1 and L2G2 with 4 planes of dimension 1×4 . Group L2G3 consists of three subgroups namely, L2G3.1, L2G3.2 and L2G3.3. L2G3.1 consists of 4 planes of size 4×1 ; L2G3.2 has 4 planes of dimension 1×4 and L2G3.3 with 4 planes of size 4×2 . In layer L3, L3G1 consists of 4 cell planes of single cells. Groups L3G2 and L3G3 have 2 and 4 numbers of subgroups respectively. L3G2 consists of two subgroups namely, L3G2.1 with 8 cell planes of single cells, and L3G2.2 with 8 planes of dimension 2×1 . L3G3 consists of four subgroups namely, L3G3.1 and L3G3.2 with 2 planes of size 2×2 ; L3G3.3 has 2 planes of dimension 2×1 and L3G3.4 with 4 planes of

size 2×2 . The outputs of layer L4 are the DFT coefficients that are grouped into three. The outputs of group L4G1 are the real coefficients. L4G2 and L4G3 respectively give 24 and 36 complex coefficients. Thus L4G1 has similar cells as in other layers whereas L4G2 and L4G3 have complex cells involving scalar multiplications of complex numbers.

In the layer L1, primitive symbols VLP, VLPA, HAP and HAPL are used in L1G1; VRP, VRPA, HBP and HBPL are used in L1G2 and MP, MPL, MPA and MPD are used in L1G3. Corresponding to each primitive symbol, a plane is formed with the name of the plane indicating the primitive symbol. Each plane in layer 1 is a square array of 4×4 cells. Corresponding cells in each of these planes operate on the same 2×2 data matrix obtained from L0. However the operations done on these differ and these planes are grouped together based on the type of operation. In the first group of planes, L1G1, cell values are obtained as the sum of two particular elements of the 2×2 data matrix. The corresponding cell values in the counterpart planes in L1G2 are obtained from the same set of two elements in L1G1. However the operation performed on them is subtraction. Cell values of the planes in L1G3 are obtained from a single specific element of the partitioned 2×2 data matrices. Once computations on the first layer are completed, the input layer L0 is no more required.

The second layer, L2, also consists of three groups. The planes in L2G1 are row vectors of dimension 4 with names indicating the input plane and the operation to be carried out. The planes are RVLP, RVLPA, RVRP and RVRPA in which 'R' indicates a row sum. RVLP indicates that each cell value in this plane is obtained from the sum of the cell values of the corresponding row of the plane VLP in L1G1. RVLPA, RVRP and RVRPA are similarly obtained from the planes VLPA, VRP and VRPA. The planes in the second group, L2G2, are CHAP, CHBP, CHAPL and CHBPL, each a column vector of dimension M. Here 'C' indicates the column sum and planes are generated from HAP, HBP, HAPL and HBPL. There are three subgroups in L2G3, each with 4 planes. The planes in subgroup L2G3.1 & L2G3.2 are row vectors & column vectors respectively of dimension M and L2G3.3 with dimension 4×2 , with names indicating the input plane and the operation to be carried out. E.g., the four planes of subgroup L2G3.3 are DMP, DMPL, DMPA and DMPD in which 'D' indicates a difference. The $(0, 0)^{\text{th}}$ cell of the plane DMP is obtained from the difference of MP_{00} and MP_{02} .

Layer L3 has three groups. L3 is computed using planes from L2 and is hence independent of both L0 and L1. L3G1 contains cell planes SRVLP, SRVLPA, SRVRP and SRVRPA each possessing a single cell. Here the 'S' in the nomenclature indicates that the sum of the cell values in the respective cell planes from L2 is used to obtain the cell value in this group.

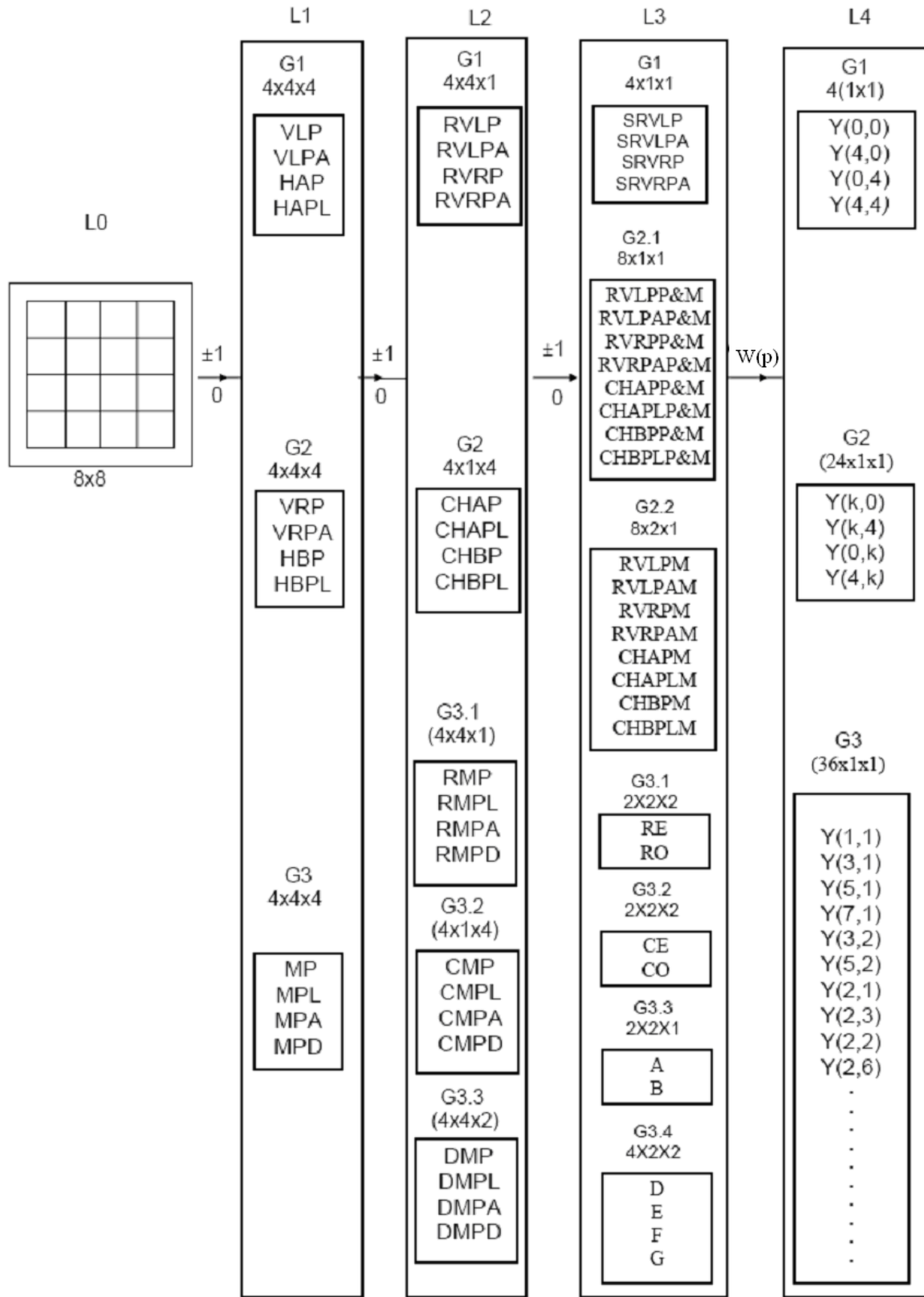


Fig. 4.2: Version I parallel distributed architecture for 8×8 point DFT computation

Hence to obtain SRVRP, the cell values of RVLP are to be summed. Similarly SRVLPA, SRVRP and SRVRPA are obtained from RVLPA, RVRP and RVRPA respectively. There are two subgroups in second group namely, L3G2.1 and L3G2.2. L3G2.1 consists of 8 cell planes with one cell each. The cell planes are RVLPP&M, RVLPA&M, RVRPP&M, RVRPA&M, CHAPP&M, CHAPLP&M, CHBPP&M and CHBPLP&M. Cell values in RVLPP&M are obtained by addition or subtraction of four specific cells from the planes RVLP. L3G2.2 consists of 8 planes with size 2×1 . Cell values in RVLPM are obtained by subtracting two specific cells from the plane RVLP. In the group 2 a 'P&M' in the name of the plane indicates plus and minus and a 'M' indicates minus. Group 3 consists of four subgroups namely, L3G3.1, L3G3.2, L3G3.3, and L3G3.4. L3G3.1 and L3G3.2 has 2 planes of size 2×2 , L3G3.3 consists of 2 cell planes of dimension 2×1 whereas L3G3.4 has 4 planes of size 2×2 . The cell values in the planes L3G3.1 and L3G3.3 are obtained by adding values of selected cells from L2G3.1. The inputs for L3G3.2 are from L2G3.2 whereas the inputs for L3G3.4 are from L2G3.3.

Layer L4 gives the N^2 DFT coefficients. The planes in L4 are grouped into three. Group 1 has 4 planes with single cells in each, giving the real coefficients $Y_{0,0}$, $Y_{0,4}$, $Y_{4,0}$ and $Y_{4,4}$. Group 2 consists of 24 cell planes with one cell in each plane. The outputs from this group correspond to the coefficients from row 0, row M, column 0 and column M other than group 1 coefficients. Group 3 in this layer consists of 36 cell planes with one cell each. The output from this group will be the DFT coefficients of group 3. The cells in L4G2 and L4G3 involve scalar multiplication of complex coefficients, whereas the cells in all other layers and groups involve only real addition and subtraction. Since the computations in a layer are dependent only on the cell planes of the previous layer, the similar calculations in a layer can be grouped in different ways.

Fig. 4.3 shows the detailed schematic diagram of parallel distributed computation of 8×8 point DFT. In the fig, the dots in each rectangle represent the cells and the dimension of each plane, represented by rectangles with continuous line, can easily be identified. Next section outlines the algorithm for the computation of each layer.

4.1.2.1 Algorithm

Layer1

For L1G1

For $0 \leq i, j < M = N/2$

$$VLP(i, j) = x(2.i, 2.j) + x(2.i, 2.j+1)$$

$$VLPA(i, j) = x(2.i+1, 2.j) + x(2.i+1, 2.j+1)$$

$$HAP(i, j) = x(2.i, 2.j) - x(2.i+1, 2.j)$$

$$HAPL(i, j) = x(2.i, 2.j+1) - x(2.i+1, 2.j+1)$$

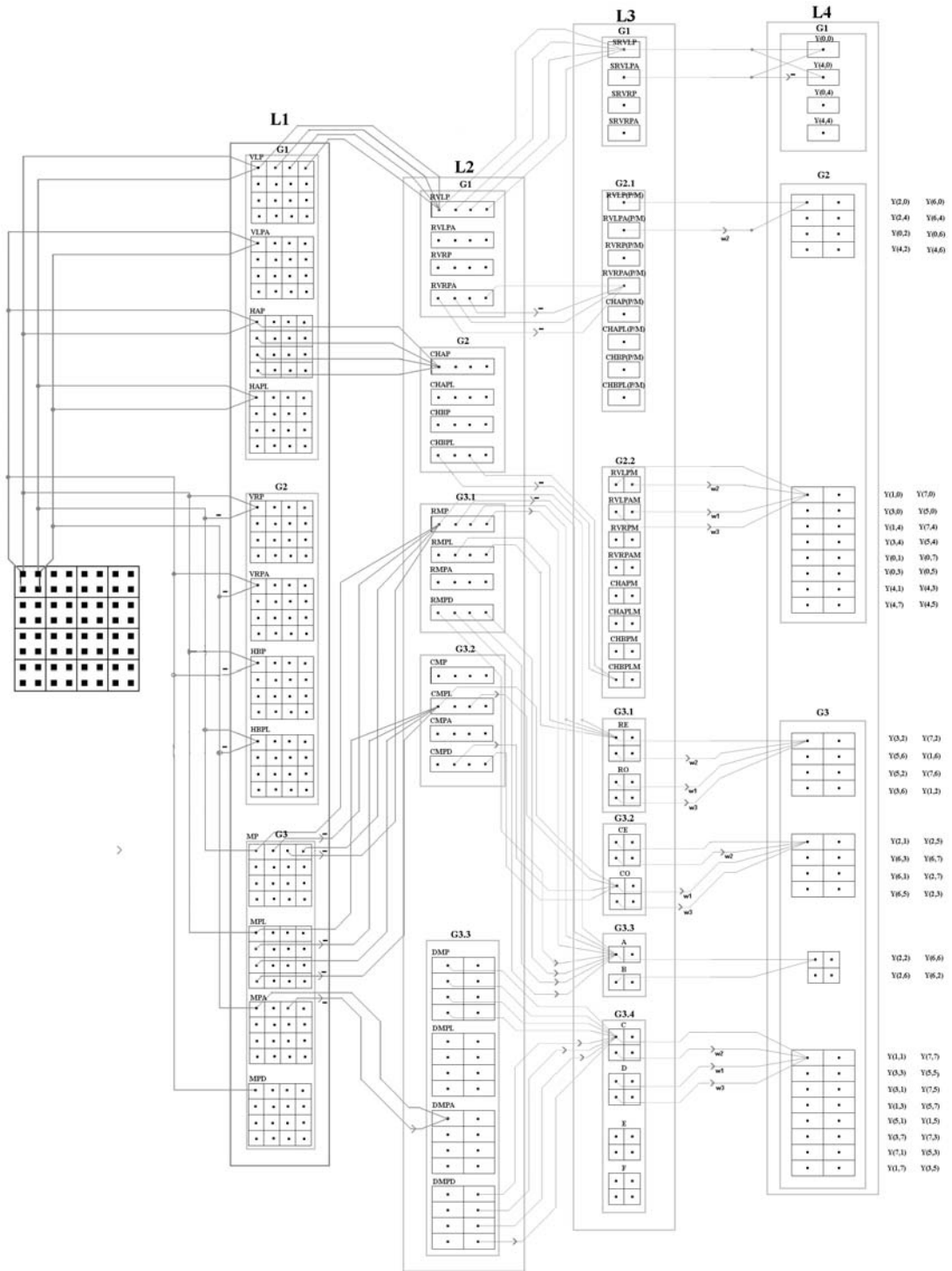


Fig. 4.3: Schematic diagram for version I architecture of 8×8 point DFT

For L1G2

$$\begin{aligned} \text{VRP}(i, j) &= x(2i, 2j) - x(2i, 2j + 1) & \text{VRPA}(i, j) &= x(2i + 1, 2j) - x(2i + 1, 2j + 1) \\ \text{HBP}(i, j) &= x(2i, 2j) - x(2i + 1, 2j) & \text{HBPL}(i, j) &= x(2i, 2j + 1) - x(2i + 1, 2j + 1) \end{aligned}$$

For L1G3

$$\begin{aligned} \text{MP}(i, j) &= x(2i, 2j) & \text{MPL}(i, j) &= x(2i, 2j + 1) \\ \text{MPA}(i, j) &= x(2i + 1, 2j) & \text{MPD}(i, j) &= x(2i + 1, 2j + 1) \end{aligned}$$

Layer 2

For L2G1

$$\text{RVLP}(i) = \sum_{j=0}^{M-1} \text{VLP}(i, j), \quad 0 \leq i < M$$

Similarly, the values of the arrays RVLPA, RVRP and RVRPA could be arrived at by replacing VLP with VLPA, VRP, and VRPA respectively in the above equation.

For L2G2

$$\text{CHAP}(i) = \sum_{j=0}^{M-1} \text{HAP}(j, i), \quad 0 \leq i < M$$

Similarly, the values of the arrays CHBP, CHAPL and CHBPL are obtained by replacing HAP with HBP, HAPL and HBPL respectively in the above equation.

For L2G3.1

$$\text{RMP}(i) = \sum_{j=0}^{M-1} (-1)^j \text{MP}(i, j), \quad 0 \leq i < M$$

Similarly the values of the arrays RMPL, RMPA, RMPD are obtained by replacing MP with MPL, MPA, MPD respectively in the above equation.

For L2G3.2

$$\text{CMP}(i) = \sum_{j=0}^{M-1} (-1)^j \text{MP}(j, i), \quad 0 \leq i < M$$

Similarly the values of the arrays CMPL, CMPA, CMPD are obtained by replacing MP with MPL, MPA, MPD respectively in the above equation.

For L2G3.3

$$\text{DMP}(i, j) = \text{MP}(i, ((j + i))_M) - \text{MP}(i, ((j + 2 + i))_M), \quad 0 \leq i < M, \quad 0 \leq j < M/2$$

Similarly DMPL, DMPA, DMPD are obtained by replacing MP with MPL, MPA, MPD respectively in the above equation.

Layer 3

i) For L3G1

$$SRVLP = \sum_{i=0}^{M-1} RVLP(i)$$

Similarly, the values of the cells SRVLP, SRVLP A, SRVRPA are derived by replacing RVLP with RVLPA, RVRP and RVRPA respectively in the above equation.

ii) For L3G2.1

$$RVLP\&M = \sum_{i=0}^{M-1} (-1)^i RVLP(i)$$

Similarly, RVLPA&M, RVRPP&M, RVRPA&M, CHAPP&M, CHAPLP&M, CHBPP&M and CHBPLP&M are derived by replacing RVLP with RVLPA, RVRP, RVRPA, CHAP, CHAPL, CHBP and CHBPL respectively in the above equation.

iii) For L3G2.2

$$RVLPM(i) = RVLP(i) - RVLP(i+2), \quad 0 \leq i < M/2$$

Similarly RVLPAM, CHAPM, CHAPLM, RVRPM, RVRPAM, CHBPM, CHBPLM are obtained by replacing RVLP with RVLPA, CHAP, CHAPL, RVRP, RVRPA, CHBP, CHBPL respectively in the above equation.

iv) For L3G3.1

For $0 \leq i, j < M/2$

$$RE(i, j) = (-1)^{i*j} [RMP(i) + (-1)^j RMPL(i+1) - RMP((i+2))_M + (-1)^{j+1} RMPL((i+3))_M]$$

$$RO(i, j) = (-1)^{i*j} [(-1)^{j+1} RMPD(i+j) + RMPA(i+j+1) + (-1)^j RMPD((i+j+2))_M - RMPA((i+j+3))_M]$$

v) For L3G3.2

For $0 \leq i, j < M/2$

$$CE(i, j) = CMP(i) + (-1)^{j+1} CMPA(i+1) - CMP(i+2) + (-1)^j CMPA((i+3))_M$$

$$CO(i, j) = CMPL(i) + (-1)^{j+1} CMPD(i+1) - CMPL(i+2) - (-1)^j CMPD((i+3))_M$$

vi) For L3G3.3

$$A(j) = \sum_{i=0}^{M-1} (-1)^i [RMP(i) + (-1)^{j+1} RMPD(i)], \quad 0 \leq j < M/2$$

Similarly B(j) is derived by replacing RMP & RMPD with RMPA & RMPL respectively in the above equation.

vii) For L3G3.4

For $0 \leq k, j < M/2$

$$D(j, k) = \sum_{i=0}^{M-1} (-1)^i DMP(i, j) + (-1)^{i+k+(i+j)M/2} DMPD(i, (j+1))_{M/2}$$

$$E(j,k) = \sum_{i=0}^{M-1} (-1)^{i+k} [DMPA(i,j) + (-1)^k DMPL(i,j)]$$

$$F(j,k) = \sum_{i=0}^{M-1} DMP(i,j) + (-1)^{k+1} DMPD(i,j)$$

$$G(j,k) = \sum_{i=0}^{M-1} DMPL(i,j) + (-1)^{j+k+1} DMPA(i,((j+1))_{M/2})$$

Layer 4

i) For L4G1

$$\begin{aligned} Y_{0,0} &= SRVLP + SRVLP & Y_{4,0} &= SRVLP - SRVLP \\ Y_{0,4} &= SRVRP + SRVRP & Y_{4,4} &= SRVRP - SRVRP \end{aligned}$$

i) For L4G2

$$\begin{aligned} Y_{1,0} &= RVLPM(0) + RVLPA(0).W1 + RVLPM(1).W2 + RVLPA(1).W3 \\ Y_{7,0} &= RVLPM(0) - RVLPA(1).W1 - RVLPM(1).W2 - RVLPA(0).W3 \\ Y_{3,0} &= RVLPM(0) + RVLPA(1).W1 - RVLPM(1).W2 + RVLPA(0).W3 \\ Y_{5,0} &= RVLPM(0) - RVLPA(0).W1 + RVLPM(1).W2 - RVLPA(1).W3 \\ Y_{2,0} &= RVLPP\&M + RVLPA\&M.W2 & Y_{6,0} &= RVLPP\&M - RVLPA\&M.W2 \\ Y_{1,4} &= RVRPM(0) + RVRPA(0).W1 + RVRPM(1).W2 + RVRPA(1).W3 \\ Y_{7,4} &= RVRPM(0) - RVRPA(1).W1 - RVRPM(1).W2 - RVRPA(0).W3 \\ Y_{3,4} &= RVRPM(0) + RVRPA(1).W1 - RVRPM(1).W2 + RVRPA(0).W3 \\ Y_{5,4} &= RVRPM(0) - RVRPA(0).W1 + RVRPM(1).W2 - RVRPA(1).W3 \\ Y_{2,4} &= RVRPP\&M + RVRPA\&M.W2 & Y_{6,4} &= RVRPP\&M - RVRPA\&M.W2 \\ Y_{0,1} &= CHAPM(0) + CHAPL(0).W1 + CHAPM(1).W2 + CHAPL(1).W3 \\ Y_{0,7} &= CHAPM(0) - CHAPL(1).W1 - CHAPM(1).W2 - CHAPL(0).W3 \\ Y_{0,3} &= CHAPM(0) + CHAPL(1).W1 - CHAPM(1).W2 + CHAPL(0).W3 \\ Y_{0,5} &= CHAPM(0) - CHAPL(0).W1 + CHAPM(1).W2 - CHAPL(1).W3 \\ Y_{0,2} &= CHAPP\&M + CHAPL\&M.W2 & Y_{0,6} &= CHAPP\&M - CHAPL\&M.W2 \\ Y_{4,1} &= CHBPM(0) + CHBPL(0).W1 + CHBPM(1).W2 + CHBPL(1).W3 \\ Y_{4,7} &= CHBPM(0) - CHBPL(1).W1 - CHBPM(1).W2 - CHBPL(0).W3 \\ Y_{4,3} &= CHBPM(0) + CHBPL(1).W1 - CHBPM(1).W2 + CHBPL(0).W3 \\ Y_{4,5} &= CHBPM(0) - CHBPL(0).W1 + CHBPM(1).W2 - CHBPL(1).W3 \\ Y_{4,2} &= CHBPP\&M + CHBPL\&M.W2 & Y_{4,6} &= CHBPP\&M - CHBPL\&M.W2 \end{aligned}$$

i) For L4G3

$$\begin{aligned} Y_{1,1} &= D(0,0) + E(0,0).W1 + D(1,0).W2 + E(1,0).W3 \\ Y_{7,7} &= D(0,0) - E(1,0).W1 - D(1,0).W2 - E(0,0).W3 \\ Y_{3,3} &= D(0,0) + E(1,0).W1 - D(1,0).W2 + E(0,0).W3 \end{aligned}$$

$$\begin{aligned}
Y_{5,5} &= D(0,0) - E(0,0).W1 + D(1,0).W2 - E(0,0).W3 \\
Y_{5,1} &= D(0,1) + E(0,1).W1 + D(1,1).W2 + E(1,1).W3 \\
Y_{1,5} &= D(0,1) - E(0,1).W1 + D(1,1).W2 - E(1,1).W3 \\
Y_{3,7} &= D(0,1) - E(1,1).W1 - D(1,1).W2 - E(0,1).W3 \\
Y_{7,3} &= D(0,1) + E(1,1).W1 - D(1,1).W2 + E(0,1).W3 \\
Y_{3,1} &= F(0,0) + G(0,0).W1 + F(1,0).W2 + G(1,0).W3 \\
Y_{7,5} &= F(0,0) - G(0,0).W1 + F(1,0).W2 - G(1,0).W3 \\
Y_{1,3} &= F(0,0) + G(1,0).W1 - F(1,0).W2 + G(0,0).W3 \\
Y_{5,7} &= F(0,0) - G(1,0).W1 - F(1,0).W2 - G(0,0).W3 \\
Y_{7,1} &= F(0,1) + G(0,1).W1 + F(1,1).W2 + G(1,1).W3 \\
Y_{5,3} &= F(0,1) + G(1,1).W1 - F(1,1).W2 + G(0,1).W3 \\
Y_{1,7} &= F(0,1) - G(1,1).W1 - F(1,1).W2 - G(0,1).W3 \\
Y_{3,5} &= F(0,1) - G(0,1).W1 + F(1,1).W2 - G(1,1).W3 \\
Y_{3,2} &= RE(0,0) + RO(0,0).W1 - RE(1,0).W2 - RO(1,0).W3 \\
Y_{7,2} &= RE(0,0) - RO(0,0).W1 - RE(1,0).W2 + RO(1,0).W3 \\
Y_{5,6} &= RE(0,0) + RO(1,0).W1 + RE(1,0).W2 - RO(0,0).W3 \\
Y_{1,6} &= RE(0,0) - RO(1,0).W1 + RE(1,0).W2 + RO(0,0).W3 \\
Y_{5,2} &= RE(0,1) + RO(0,1).W1 - RE(1,1).W2 - RO(1,1).W3 \\
Y_{7,6} &= RE(0,1) - RO(1,1).W1 + RE(1,1).W2 + RO(0,1).W3 \\
Y_{3,6} &= RE(0,1) + RO(1,1).W1 + RE(1,1).W2 - RO(0,1).W3 \\
Y_{1,2} &= RE(0,1) - RO(0,1).W1 - RE(1,1).W2 + RO(1,1).W3 \\
Y_{2,1} &= CE(0,0) + CO(0,0).W1 + CE(1,0).W2 + CO(1,0).W3 \\
Y_{2,5} &= CE(0,0) - CO(0,0).W1 + CE(1,0).W2 - CO(1,0).W3 \\
Y_{6,3} &= CE(0,0) + CO(1,0).W1 - CE(1,0).W2 + CO(0,0).W3 \\
Y_{6,7} &= CE(0,0) - CO(1,0).W1 - CE(1,0).W2 - CO(0,0).W3 \\
Y_{6,1} &= CE(0,1) + CO(0,1).W1 + CE(1,1).W2 + CO(1,1).W3 \\
Y_{2,7} &= CE(0,1) - CO(1,1).W1 - CE(1,1).W2 - CO(0,1).W3 \\
Y_{6,5} &= CE(0,1) - CO(0,1).W1 + CE(1,1).W2 - CO(1,1).W3 \\
Y_{2,3} &= CE(0,1) + CO(1,1).W1 - CE(1,1).W2 + CO(0,1).W3 \\
Y_{2,2} &= A(0) + B(1).W2 \quad Y_{6,6} = A(0) - B(1).W2 \\
Y_{2,6} &= A(1) + B(0).W2 \quad Y_{6,2} = A(1) - B(0).W2
\end{aligned}$$

4.1.2.2 Sample computation

For the data matrix $[x]$ shown below, the computation of 8×8 point DFT is as follows

$$[x] = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 & 0 & 2 & 1 \\ 0 & 3 & 1 & 3 & 2 & 1 & 1 & 1 \\ 1 & 2 & 3 & 0 & 0 & 2 & 1 & 3 \\ 3 & 1 & 0 & 2 & 0 & 1 & 1 & 3 \\ 0 & 2 & 3 & 1 & 2 & 1 & 2 & 1 \\ 1 & 2 & 0 & 1 & 3 & 1 & 0 & 1 \\ 1 & 1 & 2 & 2 & 0 & 0 & 2 & 2 \\ 2 & 2 & 0 & 0 & 1 & 3 & 1 & 2 \end{bmatrix}$$

L1G1

VLP	VLPA	HAP	HAPL
$\begin{bmatrix} 3 & 5 & 1 & 3 \\ 3 & 3 & 2 & 4 \\ 2 & 4 & 3 & 3 \\ 2 & 4 & 0 & 4 \end{bmatrix}$	$\begin{bmatrix} 3 & 4 & 3 & 2 \\ 4 & 2 & 1 & 4 \\ 3 & 1 & 4 & 1 \\ 4 & 0 & 4 & 3 \end{bmatrix}$	$\begin{bmatrix} 1 & 4 & 3 & 3 \\ 4 & 3 & 0 & 2 \\ 1 & 3 & 5 & 2 \\ 3 & 2 & 1 & 3 \end{bmatrix}$	$\begin{bmatrix} 5 & 5 & 1 & 2 \\ 3 & 2 & 3 & 6 \\ 4 & 2 & 2 & 2 \\ 3 & 2 & 3 & 4 \end{bmatrix}$

L1G2

VRP	VRPA	HBP	HBPL
$\begin{bmatrix} -1 & 1 & 1 & 3 \\ -1 & 3 & -2 & -2 \\ -2 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -3 & -2 & 1 & 0 \\ 2 & -2 & -1 & -2 \\ -1 & -1 & 2 & -1 \\ 0 & 0 & -2 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & -1 & 1 \\ -2 & 3 & 0 & 0 \\ -1 & 3 & -1 & 2 \\ -1 & 2 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 2 & -3 & 0 \end{bmatrix}$

L1G3

MP	MPL	MPA	MPD
$\begin{bmatrix} 1 & 3 & 1 & 2 \\ 1 & 3 & 0 & 1 \\ 0 & 3 & 2 & 2 \\ 1 & 2 & 0 & 2 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 & 0 & 1 \\ 2 & 0 & 2 & 3 \\ 2 & 1 & 1 & 1 \\ 1 & 2 & 0 & 2 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 2 & 1 \\ 3 & 0 & 0 & 1 \\ 1 & 0 & 3 & 0 \\ 2 & 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 3 & 3 & 1 & 1 \\ 1 & 2 & 1 & 3 \\ 2 & 1 & 1 & 1 \\ 2 & 0 & 3 & 2 \end{bmatrix}$

L2G1

RVLP	RVLPA	RVRP	RVRPA
$\begin{bmatrix} 12 \\ 12 \\ 12 \\ 10 \end{bmatrix}$	$\begin{bmatrix} 12 \\ 11 \\ 9 \\ 11 \end{bmatrix}$	$\begin{bmatrix} 2 \\ -2 \\ 2 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -4 \\ -3 \\ -1 \\ -3 \end{bmatrix}$

L2G2

CHAP	CHAPL	CHBP	CHBPL
[9 12 9 10]	[15 11 9 14]	[-3 10 -3 4]	[-1 -1 -3 0]

L2G3.1

RMP	RMPL	RMPA	RMPD
$\begin{bmatrix} -3 \\ -3 \\ -3 \\ -3 \end{bmatrix}$	$\begin{bmatrix} -1 \\ 1 \\ 1 \\ -3 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 2 \\ 4 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 0 \\ -3 \\ 1 \\ 3 \end{bmatrix}$

L2G3.2

CMP	CMPL	CMPA	CMPD
[-1 1 3 1]	[1 1 -1 -3]	[-4 1 4 -1]	[2 2 -2 -3]

L2G3.3

DMP	DMPL	DMPA	DMPD
$\begin{bmatrix} 0 & 1 \\ 2 & -1 \\ 2 & -1 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 1 \\ -3 & 0 \\ -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -2 & 0 \\ -1 & -3 \\ 2 & 0 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 \\ -1 & 0 \\ -1 & 0 \\ 2 & -1 \end{bmatrix}$

L3G1

SRVLP	SRVLPA	SRVRP	SRVRPA
[46]	[43]	[2]	[-1]

L3G2.1

RVLPP&M	RVLPA&M	RVRPP&M	RVRPA&M
[2]	[-1]	[6]	[1]
CHAPP&M	CHAPLP&M	CHBPP&M	CHBPLP&M
[-4]	[-1]	[-20]	[-3]

L3G2.2

RVLPM	RVLPA&M	RVRPM	RVRPA&M
[0 2]	[3 0]	[0 -2]	[-3 0]
CHAPM	CHAPLM	CHBPM	CHBPLM
[0 2]	[6 -3]	[0 6]	[2 -1]

L3G3.1

$$\begin{array}{cc} \text{RE} & \text{RO} \\ \begin{bmatrix} 4 & -4 \\ 2 & 2 \end{bmatrix} & \begin{bmatrix} 1 & -2 \\ 10 & -1 \end{bmatrix} \end{array}$$

L3G3.2

$$\begin{array}{cc} \text{CE} & \text{CO} \\ \begin{bmatrix} -6 & -2 \\ -8 & 8 \end{bmatrix} & \begin{bmatrix} -3 & 7 \\ 8 & 0 \end{bmatrix} \end{array}$$

L3G3.3

$$\begin{array}{cc} \text{A} & \text{B} \\ \begin{bmatrix} -1 & 1 \end{bmatrix} & \begin{bmatrix} -2 & 2 \end{bmatrix} \end{array}$$

L3G3.4

$$\begin{array}{ccc} \text{D} & \text{E} & \text{F} & \text{G} \\ \begin{bmatrix} -3 & 3 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 4 & 4 \\ 2 & -2 \end{bmatrix} & \begin{bmatrix} 2 & 6 \\ -1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -4 \\ 2 & 2 \end{bmatrix} \end{array}$$

L4G1L4G2L4G3

$$\begin{array}{ccc} \begin{bmatrix} 89 \\ 3 \\ -9 \\ 13 \end{bmatrix} & \begin{bmatrix} 2.1213 - j4.1213 \\ 2 + j \\ -2.1213 - j0.1213 \\ -2.1213 + j0.1213 \\ 2 - j \\ 2.1213 + j4.1213 \\ -2.1213 + j4.1213 \\ 6 - j \\ 2.1213 + j0.1213 \\ 2.1213 - j0.1213 \\ 6 + j \\ -2.1213 - j4.1213 \\ 6.364 - j4.1213 \\ -4 + j \\ -6.364 - j0.1213 \\ -6.364 + j0.1213 \\ -4 - j \\ 6.364 + j4.1213 \\ 2.1213 - j6.7071 \\ -20 + j3 \\ -2.1213 + j5.2929 \\ -2.1213 - j5.2929 \\ -20 - j3 \\ 2.1213 + j6.7071 \end{bmatrix} & \begin{bmatrix} -1.5858 - j4.2426 \\ -13.7782 + j4.4645 \\ 0.5858 - j0.4142 \\ 7.2426 - j1.4142 \\ 2.9497 - j12.9497 \\ 1.7574 + j0.4142 \\ -1.8787 + j1.2929 \\ -1 - j2 \\ 11.7782 + j8.364 \\ -6.1213 + j2.7071 \\ 1 - j2 \\ -3.7782 - j4.364 \\ 3.4142 - j2.4142 \\ -6.9497 + j3.0503 \\ -4.4142 - j4.2426 \\ 10.2426 + j2.4142 \\ 1.7782 - j11.5355 \\ -1.2426 - j1.4142 \end{bmatrix} & \begin{bmatrix} -1.2426 + j1.4142 \\ 1.7782 + j11.5355 \\ 10.2426 - j2.4142 \\ -4.4142 + j4.2426 \\ -6.9497 - j3.0503 \\ 3.4142 + j2.4142 \\ -3.7782 + j4.364 \\ 1 + j2 \\ -6.1213 - j2.7071 \\ 11.7782 - j8.364 \\ -1 + j2 \\ -1.8787 - j1.2929 \\ 1.7574 - j0.4142 \\ 2.9497 + j12.9497 \\ 7.2426 + j1.4142 \\ 0.5858 + j0.4142 \\ -13.7782 - j4.4645 \\ -1.5858 + j4.2426 \end{bmatrix} \end{array}$$

4.1.3 Version II architecture

In the version I parallel distributed model developed in section 4.1.2.1, effort has been made to preserve many layers and planes of the architecture model for the computation of $N \times N$ point DFT [88] where $((N))_4 = 2$ so as to develop a generalized architecture for any even N . E.g., layer 1 is exactly same in both the model. In layer 2 cell-planes of group G1 and G2 are same in both the models, whereas the cell-planes of group 3 has been divided into three subgroups namely G3.1, G3.2 and G3.3 in 8×8 point DFT. But in layer 3 only one group is common to both $((N))_4 = 2$ and 8×8 point DFT, while all other groups are different. On analysis of the version I architecture for 8×8 point DFT computation, it can be seen that the number of computations in each cell-planes of different groups in a particular layer differ. For example in layer 1, there is one addition in each cell of planes in group G1 and G2 whereas there is no computation in the cell-planes of group G3. Due to this the number of computations in the cell-planes of group G3 in layer 3 has increased. In a computation scheme where the computation of each group of a particular layer is to be completed before the commencement of the computation in the subsequent layer, this will cause a bottleneck. Even though each group of a particular layer processes the output of the corresponding group in the previous layer, the above delay affects the total execution time. On analysis of the version I architecture it is noticed that the computation accumulated in group 3 of layer 3 can be distributed to different layers, if the group 3 computations are modified.

In the version II architecture shown in fig 4.4, the number of computation in each cell-plane of layer 1 and 2 are made equal. In layer 3 however the number of computation in the cell-plane of subgroup G3.2 is two more than the other planes. In layer 4, the number of computations in each cell-plane of different groups depends on the number of Y_{k_1, k_2}^p involved in the computation of the DFT coefficients. E.g., there are no complex multiplications in L4G1, whereas in L4G2, computation of the DFT coefficient $Y_{1,0}$ require three complex multiplications and $Y_{2,0}$ require one complex multiplication. In L4G3, computation of the DFT coefficient $Y_{1,1}$ require three complex multiplications and $Y_{2,2}$ require one complex multiplication. So the variations in the number of computations in each cell-plane of groups L4G2 and L4G3 cannot be eliminated. The algorithm for group 3 in all the layers of version II model is shown below.

4.1.3.1 Algorithm

The computation of group 1 and 2 in all the layers are same as that of version I architecture. The computations of all the layers of group 3 are shown below.

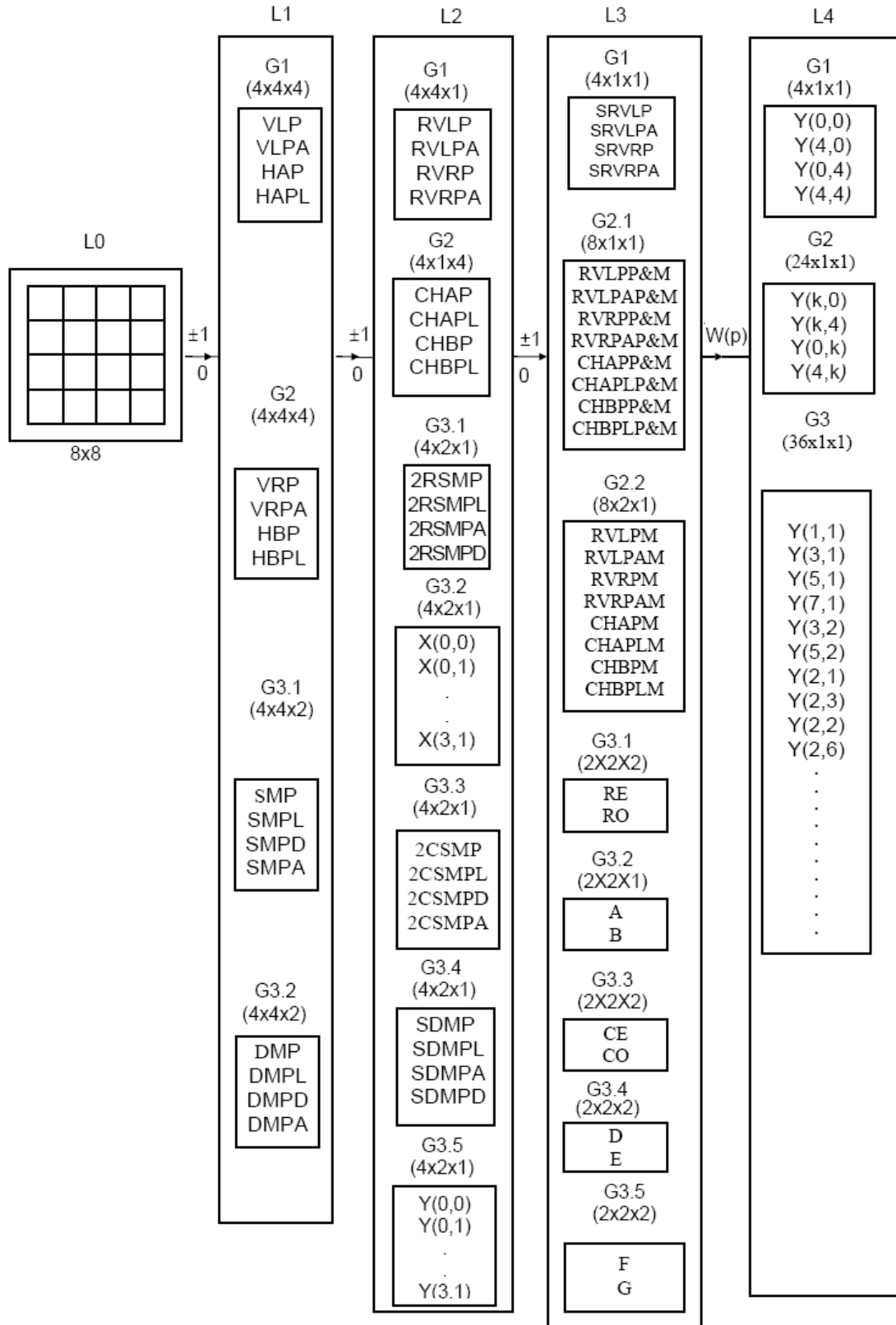


Fig. 4.4: Version II parallel distributed architecture for 8×8 point DFT

Layer 1

The input-output relation for each cell in L1 for group 3 is as follows.

For L1G3.1

For $0 \leq i < M$

For $0 \leq j < M/2$

$$SMP(i, j) = MP(i, j) + MP(i, j+2) = \sum_{k=0}^1 MP(i, j+2k) = x(2i, 2j) + x(2i, 2(j+2))$$

Similarly SMPL, SMPD and SMPA can be obtained by replacing MP with MPL, MPD and MPA or with the corresponding data in the 2×2 matrix.

For L1G3.2

For $0 \leq i < M$

For $0 \leq j < M/2$

$$\begin{aligned} DMP(i, j) &= MP(i, j+i) - MP(i, ((j+2+i))_M) = \sum_{k=0}^1 (-1)^k MP(i, ((j+i+2k))_M) \\ &= x(2i, ((2j+2i))_N) - x(2i, ((2(j+2)+2i))_N) \end{aligned}$$

Similarly DMPL, DMPD and DMPA can be obtained by replacing MP with MPL, MPD and MPA or with the corresponding data in the 2×2 matrix.

Layer 2

L2G3.1

$$2RSMP(i) = \sum_{j=0}^{M/2-1} \sum_{k=0}^1 (-1)^{k+j} SMP(i + jxM/2, k), \quad 0 \leq i < M/2$$

Similarly 2RSMPD, 2RSMPA can be obtained by replacing SMP with SMPL, SMPD and SMPA respectively.

L2G3.2

$$X(o, i) = \sum_{k=0}^{M/2-1} \sum_{l=0}^{M/2-1} (-1)^l SMP(2k+i, l), \quad 0 \leq i \leq M/2$$

Similarly $X(1, i)$, $X(2, i)$ and $X(3, i)$ can be obtained by replacing SMP with SMPA, SMPD and SMPL respectively in the above equation.

L2G3.3

$$2CSMP(i) = \sum_{j=0}^{M/2-1} (-1)^{jxi} (DMP((i+j-2xjxi), j) - DMP((2+i+j-2xjxi), j)), \quad 0 \leq i < M/2$$

Similarly 2CSMPD, 2CSMPA can be obtained by replacing DMP with DMPL, DMPD and DMPA respectively in the above equation.

L2G3.4

$$SDMP(i) = \sum_{j=0}^{M/2-1} (-1)^j DMP(j, i), \quad 0 \leq i < M/2$$

Similarly SDMPPL, SDMPD and SDMPA can be obtained by replacing DMP with DMPL, DMPD and DMPA respectively in the above equation.

L2G3.5

$$Y(0, i) = \sum_{k=0}^{M/2-1} DMP(k, i), \quad 0 \leq i \leq M/2$$

Similarly $Y(1, i)$, $Y(2, i)$ and $Y(3, i)$ can be obtained by replacing DMP with DMPA, DMPD and DMPL respectively in the above equation.

Layer 3

L3G3.1

$$RE(i, j) = (-1)^{ij} (2RSMP(i) + (-1)^{i+j} 2RSMPPL(((i+1))_{M/2})), \quad 0 \leq i, j < M/2$$

$$RO(i, j) = (-1)^{j+1} 2RSMPD(((i+j))_{M/2}) + (-1)^{i+j} 2RSMPA(((i+j+1))_{M/2}), \quad 0 \leq i, j < M/2$$

L3G3.2

$$A(i, j) = \sum_{k=0}^{M/2-1} \sum_{l=0}^{M/2-1} (-1)^{i+k+lxj} X(2k+i, l), \quad 0 \leq i, j \leq M/2 - 1$$

L3G3.3

$$CE(i, j) = 2CSMP(i) + (-1)^{j+1} 2CSMPA((i+1))_{M/2}, \quad 0 \leq i, j < M/2$$

$$CO(i, j) = 2CSMPPL(i) + (-1)^{i+j+1} 2CSMPD((i+1))_{M/2}, \quad 0 \leq i, j < M/2$$

L3G3.4

$$D(i, j) = SDMP(i) + (-1)^{i+j+1} SDMPD((i+1))_{M/2}, \quad 0 \leq i, j \leq M/2$$

$$E(i, j) = (-1)^{i+ixj} (SDMPA(i) + (-1)^j SDMPPL(i)), \quad 0 \leq i, j \leq M/2$$

L3G3.5

$$F(i, j) = \sum_{k=0}^{M/2-1} (-1)^{(1+j)k} Y(2k, i), \quad 0 \leq i, j \leq M/2$$

$$G(i, j) = (-1)^{i+ixj} (Y(1, ((i+1))_{M/2}) + (-1)^{i+j+1} Y(3, i)), \quad 0 \leq i, j \leq M/2$$

Layer 4

i) For L4G3

$$Y_{1,1} = D(0,0) + E(0,0).W1 + D(1,0).W2 - E(1,0).W3$$

$$Y_{7,7} = D(0,0) + E(1,0).W1 - D(1,0).W2 - E(0,0).W3$$

$$Y_{3,3} = D(0,0) - E(1,0).W1 - D(1,0).W2 + E(0,0).W3$$

$$Y_{5,5} = D(0,0) - E(0,0).W1 + D(1,0).W2 + E(0,0).W3$$

$$\begin{aligned}
Y_{5,1} &= D(0,1) - E(0,1).W1 + D(1,1).W2 - E(1,1).W3 \\
Y_{1,5} &= D(0,1) + E(0,1).W1 + D(1,1).W2 + E(1,1).W3 \\
Y_{3,7} &= D(0,1) + E(1,1).W1 - D(1,1).W2 + E(0,1).W3 \\
Y_{7,3} &= D(0,1) - E(1,1).W1 - D(1,1).W2 - E(0,1).W3 \\
Y_{3,1} &= F(0,0) - G(0,0).W1 + F(1,0).W2 - G(1,0).W3 \\
Y_{7,5} &= F(0,0) + G(0,0).W1 + F(1,0).W2 + G(1,0).W3 \\
Y_{1,3} &= F(0,0) - G(1,0).W1 - F(1,0).W2 - G(0,0).W3 \\
Y_{5,7} &= F(0,0) + G(1,0).W1 - F(1,0).W2 + G(0,0).W3 \\
Y_{7,1} &= F(0,1) + G(0,1).W1 + F(1,1).W2 - G(1,1).W3 \\
Y_{5,3} &= F(0,1) - G(1,1).W1 - F(1,1).W2 + G(0,1).W3 \\
Y_{1,7} &= F(0,1) + G(1,1).W1 - F(1,1).W2 - G(0,1).W3 \\
Y_{3,5} &= F(0,1) - G(0,1).W1 + F(1,1).W2 + G(1,1).W3 \\
Y_{3,2} &= RE(0,0) + RO(0,0).W1 - RE(1,0).W2 - RO(1,0).W3 \\
Y_{7,2} &= RE(0,0) - RO(0,0).W1 - RE(1,0).W2 + RO(1,0).W3 \\
Y_{5,6} &= RE(0,0) + RO(1,0).W1 + RE(1,0).W2 - RO(0,0).W3 \\
Y_{1,6} &= RE(0,0) - RO(1,0).W1 + RE(1,0).W2 + RO(0,0).W3 \\
Y_{5,2} &= RE(0,1) + RO(0,1).W1 - RE(1,1).W2 - RO(1,1).W3 \\
Y_{7,6} &= RE(0,1) - RO(1,1).W1 + RE(1,1).W2 + RO(0,1).W3 \\
Y_{3,6} &= RE(0,1) + RO(1,1).W1 + RE(1,1).W2 - RO(0,1).W3 \\
Y_{1,2} &= RE(0,1) - RO(0,1).W1 - RE(1,1).W2 + RO(1,1).W3 \\
Y_{2,1} &= CE(0,1) + CO(0,1).W1 - CE(1,0).W2 - CO(1,1).W3 \\
Y_{2,5} &= CE(0,1) - CO(0,1).W1 - CE(1,0).W2 + CO(1,1).W3 \\
Y_{6,3} &= CE(0,1) - CO(1,1).W1 + CE(1,0).W2 + CO(0,1).W3 \\
Y_{6,7} &= CE(0,1) + CO(1,1).W1 + CE(1,0).W2 - CO(0,1).W3 \\
Y_{6,1} &= CE(0,0) + CO(0,0).W1 - CE(1,1).W2 - CO(1,0).W3 \\
Y_{2,7} &= CE(0,0) + CO(1,0).W1 + CE(1,1).W2 - CO(0,0).W3 \\
Y_{6,5} &= CE(0,0) - CO(0,0).W1 - CE(1,1).W2 + CO(1,0).W3 \\
Y_{2,3} &= CE(0,0) - CO(1,0).W1 + CE(1,1).W2 + CO(0,0).W3 \\
Y_{2,2} &= A(0,0) + A(1,1).W2 & Y_{6,6} &= A(0,0) - A(1,1).W2 \\
Y_{2,6} &= A(0,1) + A(1,0).W2 & Y_{6,2} &= A(0,1) - A(1,0).W2
\end{aligned}$$

4.1.4 Comparison of version I & II models with the model for $((N))_4 = 2$

Version I and version II parallel distributed architecture for the computation of 8×8 point DFT are developed using the visual representation based on 2×2 DFT. Since the aim is to develop a generalized architecture for $N \times N$ point DFT where N is any even integer, both the models are designed in the same way as that for $((N))_4 = 2$ [71]. E.g., the computation of group 1 coefficients in version I and II models are same as that of the model for $((N))_4 = 2$. The computation for group 2 coefficients in layer 1 and 2 are same in all the models. Similarity ends there. The computation for group 2 coefficients is different in layer 3 for the 8×8 point computation models, from that of computation of 2-D DFT for $((N))_4 = 2$. Primitive symbol combinations for group 2 and 3 coefficients differs for $((N))_4 = 2$ and $N = 8$. Hence a generalized architecture based on the above model is not feasible.

4.2 Development of M spacing based architecture for $N \times N$ DFT

Analysis of visual representation of Y_{k_1, k_2}^p based on 2×2 data can be used to derive simple and efficient computational scheme as the representation shows a direct relationship between data and the frequency domain representation. Analysis shows similarities in the representation for Y_{k_1, k_2}^p of several DFT coefficients as in section 3.2.4. Due to the redundancy, only the basic DFT coefficients need be calculated and other coefficients could be derived. Hence the analysis of visual representation can be confined to the Y_{k_1, k_2}^p of basic DFT coefficients, since others are redundant.

4.2.1 Patterns in Y_{k_1, k_2}^p of basic DFT coefficients

The computation of Y_{k_1, k_2}^p using the visual representation involves only real additions as is evident from fig. 3.6, 3.7 and 3.8. The theorem 3.1 says that the existence of Y_{k_1, k_2}^p depends on ‘ dm ’, where ‘ dm ’ is the divisor of M . But the number of data points involved in the computation of each of the DFT coefficient is same. Table 4.2 shows the number of data points involved in the computation of Y_{k_1, k_2}^p for $N = 4, 6$ and 8 corresponding to each $\gcd(k_1, k_2, M)$. From the table, for $N = 4$ and $\gcd(k_1, k_2, M) = 1$, there are two Y_{k_1, k_2}^p each having eight data points, i.e., sixteen data points have been equally distributed between the two Y_{k_1, k_2}^p , whereas for $\gcd(k_1, k_2, M) = 2$, there is only one Y_{k_1, k_2}^p with sixteen data points involved in the computation. When $N = 6$, there are three Y_{k_1, k_2}^p each having twelve data points when $\gcd(k_1, k_2, M) = 1$ and one Y_{k_1, k_2}^p with thirty six data points when $\gcd(k_1, k_2, M) = 3$. Similarly for $N = 8$, there are four Y_{k_1, k_2}^p with sixteen data points each when $\gcd(k_1, k_2, M) =$

1, two Y_{k_1, k_2}^p with thirty two data points each when $\gcd(k_1, k_2, M) = 2$ and one Y_{k_1, k_2}^p with sixteen data points each when $\gcd(k_1, k_2, M) = 4$. Similar pattern is observed in the visual representation of higher values of N . Thus the number of data points involved in the computation of Y_{k_1, k_2}^p can be computed and can be generalized for any even N .

$$\text{Number of } Y_{k_1, k_2}^p, \text{ when } \gcd(k_1, k_2, M) \text{ is } dm, n_p = M/dm$$

$$\therefore \text{Number of data points involved in the computation of } Y_{k_1, k_2}^p = N^2/n_p = 2.N.dm \quad (4.1)$$

Since N is even, from (4.1), the number of data points in any Y_{k_1, k_2}^p will always be an integral multiple of four.

Table 4.2: Number of data points involved in the computation of Y_{k_1, k_2}^p for $N = 4, 6$ & 8

N	$\gcd(k_1, k_2, M) = dm$	Number of Y_{k_1, k_2}^p	Number of data in each Y_{k_1, k_2}^p
4	1	2	8
	2	1	16
6	1	3	12
	3	1	36
8	1	4	16
	2	2	32
	4	1	64

4.2.2 M spacing based data availability

On analysis of the visual representation of Y_{k_1, k_2}^p , a pattern is seen present among the data involved in its computation and is illustrated in the following theorem.

Theorem 4.1

If a data at (n_1, n_2) is present in the visual representation of Y_{k_1, k_2}^p then the data at $(n_1, n_2 + M)$, $(n_1 + M, n_2)$ and $(n_1 + M, n_2 + M)$ will also be present.

Proof

Let us assume that there is a data at (n_1, n_2) .

Then from (1.6) and (1.7),

$$((n_1.k_1 + n_2.k_2))_N = p \text{ or } p + M. \quad (4.2)$$

a). Now for the data point which is M space apart from the above i.e., $(n_1, n_2 + M)$ is given by

$$\begin{aligned} ((n_1.k_1 + (n_2 + M).k_2))_N &= ((n_1.k_1 + n_2.k_2 + M.k_2))_N = (((n_1.k_1 + n_2.k_2))_N + M.k_2)_N \\ &= ((p + M.k_2))_N \text{ or } ((p + M + M.k_2))_N \end{aligned} \quad (4.3)$$

There are two cases

Case 1: k_2 is even or 0

Let $k_2 = 2.t$, then (4.3) becomes

$$((p + M.2.t))_N \text{ or } ((p + M + M.2.t))_N = p \text{ or } p + M \quad (4.4)$$

Case 2: k_2 is odd

Let $k_2 = 2.t + 1$, then (4.3) becomes

$$((p + M(2.t + 1)))_N \text{ or } ((p + M + M(2.t + 1)))_N = p + M \text{ or } p \quad (4.5)$$

From (4.4) and (4.5) it can be inferred that if there is a data at (n_1, n_2) for Y_{k_1, k_2}^p , then there will always be a data at $(n_1, n_2 + M)$.

b) Now for the point $(n_1 + M, n_2)$ which is also M space apart from (n_1, n_2)

$$\begin{aligned} (((n_1 + M)k_1 + n_2.k_2))_N &= ((n_1.k_1 + M.k_1 + n_2.k_2))_N = (((n_1.k_1 + n_2.k_2))_N + M.k_1)_N \\ &= ((p + M.k_1))_N \text{ or } ((p + M + M.k_1))_N \end{aligned} \quad (4.6)$$

There are two cases here:

Case 1: k_1 is even or 0

Let $k_1 = 2.s$, then (4.6) becomes

$$((p + M.2.s))_N \text{ or } ((p + M + M.2.s))_N = p \text{ or } p + M \quad (4.7)$$

Case 2: k_1 is odd

Let $k_1 = 2.s + 1$, then (4.6) becomes

$$((p + M(2.s + 1)))_N \text{ or } ((p + M + M(2.s + 1)))_N = p + M \text{ or } p \quad (4.8)$$

From (4.7) and (4.8) it is proved that if there is a data at (n_1, n_2) for a Y_{k_1, k_2}^p , then there will always be a data at $(n_1 + M, n_2)$ also. From (4.4) and (4.5), it follows that if there is a data at $(n_1 + M, n_2)$, then there will always be data at $(n_1 + M, n_2 + M)$. Thus the theorem is proved.

Section 4.2.1 shows that the number of data points in Y_{k_1, k_2}^p is always a multiple of four and theorem 4.1 proves that the four data points are available at a spacing of M data points. In fig. 3.6, 3.7, and 3.8, one of the four patterns shown in fig. 4.5 is seen repeated in Y_{k_1, k_2}^p . In the fig. 4.5 “□” and “■” denote that the data from the respective position is to be added and subtracted respectively. Hence the data at the above four points can be clubbed together and computed. The data is to be added or subtracted depends on whether k_1 and k_2 is even or odd as can be seen in the following theorem.

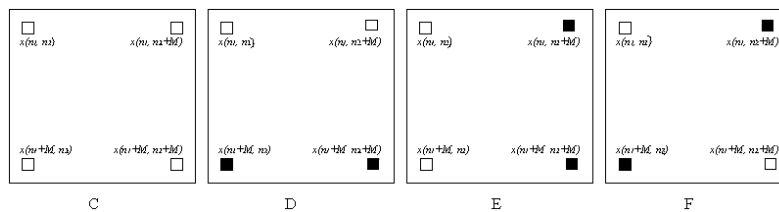


Fig. 4.5: Patterns seen repeated in Y_{k_1, k_2}^p

Theorem 4.2

One type of pattern 'C', 'D', 'E', or 'F' and/or its sign reversed form in fig. 4.5 will be present in the visual representation of Y_{k_1, k_2}^p . The type of the pattern depends on whether the frequency index k_1 and/or k_2 is even or odd.

Proof

Case 1: k_1 and k_2 even or 0

Let the data at (n_1, n_2) is to be added in the computation of Y_{k_1, k_2}^p . Then $((n_1.k_1 + n_2.k_2))_N = p$

For data point (n_1, n_2+M) , from (4.4),

$$((n_1.k_1 + (n_2 + M)k_2))_N = p$$

For data point $(n_1 + M, n_2)$, from (4.7),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p$$

For data point $(n_1 + M, n_2 + M)$,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p.$$

Thus, it can be seen that if the data at (n_1, n_2) is to be added, then all the other three data at M spacing are also to be added in the computation.

Similarly if (n_1, n_2) is to be subtracted, then

$$((n_1.k_1 + n_2.k_2))_N = p + M$$

For data point $(n_1, n_2 + M)$ from (4.4),

$$((n_1.k_1 + (n_2 + M)k_2))_N = p + M$$

For data point $(n_1 + M, n_2)$ from (4.7),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p + M$$

For data point $(n_1 + M, n_2 + M)$,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p + M.$$

Hence the data at $(n_1, n_2 + M)$, $(n_1 + M, n_2)$ and $(n_1 + M, n_2 + M)$ are to be subtracted, if the data at (n_1, n_2) is to be subtracted. So the addition of the above four points can be defined as $C(n_1, n_2)$ when k_1 and k_2 is either even or 0. For a Y_{k_1, k_2}^p , depending on the position of (n_1, n_2) , $C(n_1, n_2)$ has to be added or subtracted in the computation of Y_{k_1, k_2}^p .

Case 2: k_1 and k_2 odd

Let the data at (n_1, n_2) is to be added in the computation of Y_{k_1, k_2}^p . Then

$$((n_1.k_1 + n_2.k_2))_N = p.$$

For data point $(n_1, n_2 + M)$ from (4.5),

$$((n_1.k_1 + (n_2 + M)k_2))_N = p + M.$$

For data point $(n_1 + M, n_2)$ from (4.8),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p + M.$$

For data point $(n_1 + M, n_2 + M)$,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p.$$

Thus, it can be seen that if (n_1, n_2) is to be added, then $(n_1, n_2 + M)$, $(n_1 + M, n_2)$ is to be subtracted and $(n_1 + M, n_2 + M)$ is to be added in the computation.

Similarly if (n_1, n_2) is to be subtracted, then

$$((n_1.k_1 + n_2.k_2))_N = p + M.$$

For data point $(n_1, n_2 + M)$ from (4.5),

$$((n_1.k_1 + (n_2 + M)k_2))_N = p.$$

For data point $(n_1 + M, n_2)$ from (4.8),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p.$$

For data point $(n_1 + M, n_2 + M)$,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p + M.$$

i.e., if data at (n_1, n_2) is to be subtracted, then the data at $(n_1, n_2 + M)$, $(n_1 + M, n_2)$ are to be added and $(n_1 + M, n_2 + M)$ is to be subtracted in the computation.

So if k_1 and k_2 are odd, then the data at (n_1, n_2) & $(n_1 + M, n_2 + M)$ are to be added and that at $(n_1, n_2 + M)$ & $(n_1 + M, n_2)$ are to be subtracted or vice-versa. So the addition of the above four points can be defined as $F(n_1, n_2)$ when k_1 and k_2 are odd. For a Y_{k_1, k_2}^p , depending on the position of (n_1, n_2) , $F(n_1, n_2)$ has to be added or subtracted in the computation of Y_{k_1, k_2}^p .

Case 3: k_1 even and k_2 odd

Let the data at (n_1, n_2) is to be added in the computation of Y_{k_1, k_2}^p . Then

$$((n_1.k_1 + n_2.k_2))_N = p.$$

For data point $(n_1, n_2 + M)$ from (4.5),

$$((n_1.k_1 + (n_2 + M)k_2))_N = p + M.$$

For data point $(n_1 + M, n_2)$ from (4.7),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p.$$

For data point $(n_1 + M, n_2 + M)$,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p + M.$$

Thus, if the data at (n_1, n_2) is to be added, then that at $(n_1, n_2 + M)$, $(n_1 + M, n_2 + M)$ are to be subtracted and $(n_1 + M, n_2)$ is to be added in the computation.

Similarly if the data at (n_1, n_2) is to be subtracted, then

$$((n_1.k_1 + n_2.k_2))_N = p + M.$$

For data point $(n_1, n_2 + M)$ from (4.5),

$$((n_1.k_1 + (n_2+M)k_2))_N = p.$$

For data point $(n_1 + M, n_2)$ from (4.7),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p + M.$$

For data point (n_1+M, n_2+M) ,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p.$$

i.e., if data at (n_1, n_2) is to be subtracted, then those at $(n_1, n_2 + M)$ & $(n_1 + M, n_2 + M)$ are to be added and that at $(n_1 + M, n_2)$ is to be subtracted in the computation.

So if k_1 is even and k_2 is odd, then the data at (n_1, n_2) & $(n_1 + M, n_2)$ are to be added and $(n_1, n_2 + M)$ & $(n_1 + M, n_2 + M)$ are to be subtracted or vice-versa. So the addition of the above four points can be defined as $E(n_1, n_2)$ when k_1 is even and k_2 is odd. For a Y_{k_1, k_2}^p , depending on the position of (n_1, n_2) , $E(n_1, n_2)$ has to be added or subtracted in the computation of Y_{k_1, k_2}^p .

Case 4: k_1 odd and k_2 even

Let the data at (n_1, n_2) is to be added in the computation of Y_{k_1, k_2}^p . Then $((n_1.k_1 + n_2.k_2))_N = p$.

For data point $(n_1, n_2 + M)$ from (4.4),

$$((n_1.k_1 + (n_2 + M)k_2))_N = p.$$

For data point $(n_1 + M, n_2)$ from (4.8),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p + M.$$

For data point $(n_1 + M, n_2 + M)$,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p + M.$$

Thus if the data at (n_1, n_2) is to be added, then those at $(n_1 + M, n_2)$ & $(n_1 + M, n_2 + M)$ are to be subtracted and that at $(n_1, n_2 + M)$ is to be added in the computation.

Similarly if the data at (n_1, n_2) is to be subtracted, then

$$((n_1.k_1 + n_2.k_2))_N = p + M.$$

For data point $(n_1, n_2 + M)$ from (4.4),

$$((n_1.k_1 + (n_2 + M)k_2))_N = p + M.$$

For data point $(n_1 + M, n_2)$ from (4.8),

$$(((n_1 + M)k_1 + n_2.k_2))_N = p.$$

For data point $(n_1 + M, n_2 + M)$,

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p.$$

i.e., if the data at (n_1, n_2) is to be subtracted, then those at $(n_1 + M, n_2)$ & $(n_1 + M, n_2 + M)$ are to be added and that at $(n_1, n_2 + M)$ is to be subtracted in the computation.

So if k_1 is odd and k_2 is even, then the data at (n_1, n_2) & $(n_1, n_2 + M)$ are to be added and those at $(n_1 + M, n_2)$ & $(n_1 + M, n_2 + M)$ are to be subtracted or vice-versa. So the addition of the

above four points can be defined as $D(n_1, n_2)$ when k_1 is even and k_2 is odd. For a Y_{k_1, k_2}^p , depending on the position of (n_1, n_2) , $D(n_1, n_2)$ has to be added or subtracted in the computation of Y_{k_1, k_2}^p .

4.2.3 Five layer architecture for 8×8 DFT

The five layer M spacing based model is designed using a hierarchical structure in a layered architecture as shown in fig. 4.6. It consists of a cascade connection of a number of modular structures preceded by an input layer L0. L0 is a 2-D array of input data, $x(i, j)$, $0 \leq i, j \leq N-1$. There are five layers L1 to L5 other than the input layer.

Redundancy in the computation of patterns C and D in fig. 4.5 can be noticed. Similar redundancy is present in the computation of E and F. Hence the four data at M spacing is computed as per the patterns shown in fig. 4.5 in two steps, as in layer 1 and 2 of fig. 4.6, so as to eliminate the redundancy. In the first step, the sum and difference of data $x(i, j)$ and $x(i, j + M)$ are computed and stored in matrix A and B respectively, each of size 8×4 . In the second step, the sum and difference of $A(i, j)$ and $A(i + M, j)$ as well as that of $B(i, j)$ and $B(i + M, j)$ are computed resulting in the computation of the patterns C, D, E, and F, each of size 4×4 .

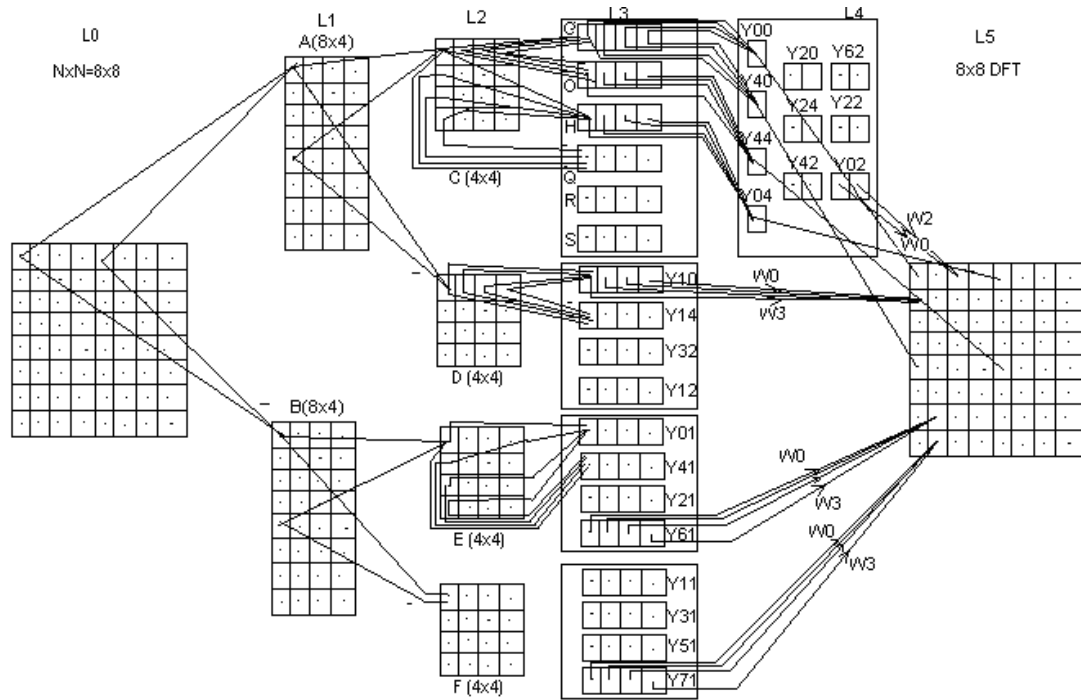


Fig. 4.6: M spacing based five layer architecture for 8×8 DFT

Y_{k_1, k_2}^p corresponding to those basic DFT coefficients with $\gcd(k_1, k_2, M) = 1$ are completely computed in layer 3 and that of the remaining are completed only in Layer 4. In layer 4, Y_{k_1, k_2}^p of

the basic DFT coefficients corresponding to the indices (0, 0), (4, 0), (0, 4), (4, 4) and (2, 0), (2, 4), (4, 2), (2, 2), (6, 2), (0, 2) with $\gcd(k_1, k_2, M) = 4$ and 2 respectively are computed completely. These Y_{k_1, k_2}^p require additional computation due to the involvement of more number of data. Complete set of DFT coefficients are computed in layer 5 as per the steps 3, 4 and 5 given in the algorithm in section 3.3.

The algorithm for computation is as shown below.

Layer 1

In layer 1, there are two groups A and B. Algorithm for layer 1 is given below.

For $0 \leq i < N - 1$, $0 \leq j < M - 1$

$$A(i, j) = x(i, j) + x(i, j + M) \quad B(i, j) = x(i, j) - x(i, j + M)$$

Both A and B will be of size $N \times M$.

Layer 2

In layer 2 there are four groups C, D, E and F. Algorithm for layer 2 is given below.

For $0 \leq j < M - 1$

$$C(i, j) = A(i, j) + A(i + M, j) \quad D(i, j) = A(i, j) - A(i + M, j)$$

$$E(i, j) = B(i, j) + B(i + M, j) \quad F(i, j) = B(i, j) - B(i + M, j)$$

C, D, E and F will be of size $M \times M$.

Layer 3

C group

For $0 \leq j < M - 1$

$$G(j) = \sum_{i=0}^{M-1} C(j, i) \quad O(j) = \sum_{i=0}^{M-1} (-1)^i C(j, i)$$

$$H(j) = \sum_{i=0}^{M-1} C(i, j) \quad Q(j) = \sum_{i=0}^{M-1} (-1)^i C(i, j)$$

$$R(j) = \sum_{i=0}^{M-1} C(i, ((j + i(M - 1)))_M) \quad S(j) = \sum_{i=0}^{M-1} C(i, ((j + i.3(M - 1)))_M)$$

D group

For $0 \leq j < M - 1$

$$Y_{1,0}^j = \sum_{i=0}^{M-1} D(j, i) \quad Y_{1,4}^j = \sum_{i=0}^{M-1} (-1)^i D(j, i)$$

$$Y_{3,2}^0 = D(0,0) - D(0,2) + D(2,1) - D(2,3) \quad Y_{3,2}^1 = -D(1,1) + D(1,3) + D(3,0) - D(3,2)$$

$$Y_{3,2}^2 = D(0,1) - D(0,3) - D(2,0) + D(2,2) \quad Y_{3,2}^3 = D(1,0) - D(1,2) + D(3,1) - D(3,3)$$

$$Y_{1,2}^0 = D(0,0) - D(0,2) - D(2,1) + D(2,3) \quad Y_{1,2}^1 = D(1,0) - D(1,2) - D(3,1) + D(3,3)$$

$$Y_{1,2}^2 = D(0,1) - D(0,3) + D(2,0) - D(2,2)$$

$$Y_{1,2}^3 = D(1,1) - D(1,3) + D(3,0) - D(3,2)$$

E group

For $0 \leq j < M - 1$

$$Y_{0,1}^j = \sum_{i=0}^{M-1} E(i, j)$$

$$Y_{4,1}^j = \sum_{i=0}^{M-1} (-1)^i E(i, j)$$

$$Y_{6,1}^0 = E(0,0) + E(1,2) - E(2,0) - E(3,2)$$

$$Y_{6,1}^1 = E(0,1) + E(1,3) - E(2,1) - E(3,3)$$

$$Y_{6,1}^2 = E(0,2) - E(1,0) - E(2,2) + E(3,0)$$

$$Y_{6,1}^3 = E(0,3) - E(1,1) - E(2,3) + E(3,1)$$

$$Y_{2,1}^0 = E(0,0) - E(1,2) - E(2,0) + E(3,2)$$

$$Y_{2,1}^1 = E(0,1) - E(1,3) - E(2,1) + E(3,3)$$

$$Y_{2,1}^2 = E(0,2) + E(1,0) - E(2,2) - E(3,0)$$

$$Y_{2,1}^3 = E(0,3) + E(1,1) - E(2,3) - E(3,1)$$

F group

$$Y_{1,1}^0 = F(0,0) - F(1,3) - F(2,2) - F(3,1)$$

$$Y_{1,1}^1 = F(0,1) + F(1,0) - F(2,3) - F(3,2)$$

$$Y_{1,1}^2 = F(0,2) + F(1,1) + F(2,0) - F(3,3)$$

$$Y_{1,1}^3 = F(0,3) + F(1,2) + F(2,1) + F(3,0)$$

$$Y_{3,1}^0 = F(0,0) - F(1,1) + F(2,2) - F(2,3)$$

$$Y_{3,1}^1 = F(0,1) - F(1,2) + F(2,3) + F(3,0)$$

$$Y_{3,1}^2 = F(0,2) - F(1,3) - F(2,0) + F(3,1)$$

$$Y_{3,1}^3 = F(0,3) + F(1,0) - F(2,1) + F(3,2)$$

$$Y_{5,1}^0 = F(0,0) + F(1,3) - F(2,2) + F(3,1)$$

$$Y_{5,1}^1 = F(0,1) - F(1,0) - F(2,3) + F(3,2)$$

$$Y_{5,1}^2 = F(0,2) - F(1,1) + F(2,0) + F(3,3)$$

$$Y_{5,1}^3 = F(0,3) - F(1,2) + F(2,1) - F(3,0)$$

$$Y_{7,1}^0 = F(0,0) + F(1,1) + F(2,2) + F(3,3)$$

$$Y_{7,1}^1 = F(0,1) + F(1,2) + F(2,3) - F(3,0)$$

$$Y_{7,1}^2 = F(0,2) + F(1,3) - F(2,0) - F(3,1)$$

$$Y_{7,1}^3 = F(0,3) - F(1,0) - F(2,1) - F(3,2)$$

Layer 4

$$Y_{0,1}^0 = \sum_{i=0}^{M-1} G(i)$$

$$Y_{4,0}^0 = \sum_{i=0}^{M-1} (-1)^i G(i)$$

$$Y_{0,4}^0 = \sum_{i=0}^{M-1} (-1)^i H(i)$$

$$Y_{4,4}^0 = \sum_{i=0}^{M-1} (-1)^i O(i)$$

$$Y_{2,0}^0 = G(0) - G(2), Y_{2,0}^2 = G(1) - G(3)$$

$$Y_{2,4}^0 = O(0) - O(2), Y_{2,4}^2 = O(1) - O(3)$$

$$Y_{4,2}^0 = Q(0) - Q(2), Y_{4,2}^2 = Q(1) - Q(3)$$

$$Y_{0,2}^0 = H(0) - H(2), Y_{0,2}^2 = H(1) - H(3)$$

$$Y_{2,2}^0 = R(0) - R(2), Y_{2,2}^2 = R(1) - R(3)$$

$$Y_{6,2}^0 = S(0) - S(2), Y_{6,2}^2 = S(1) - S(3)$$

Layer 5

The computations in layer 5 are same as that of the steps 3, 4 and 5 of the algorithm shown in section 3.3.

4.2.4 Four layer architecture for 8×8 DFT

In the architecture shown in fig. 4.6, Y_{k_1, k_2}^p corresponding to all basic DFT coefficients with $\gcd(k_1, k_2, M) = 1$ are available at the output of layer 3. The number of additions/subtractions in layer 3 is same for all groups, but that in layer 4 depends on $\gcd(k_1, k_2, M)$. For a Y_{k_1, k_2}^p when $\gcd(k_1, k_2, M) = 1$, there is no computation in layer 4, whereas when $\gcd(k_1, k_2, M) = 2$ and 4, the number of additions/subtractions is 1 and 3 respectively. The number of computations is $\gcd(k_1, k_2, M) - 1$ and hence cannot be made equal. Therefore layer 3 and 4 are combined in the four layer M spacing architecture as shown in fig. 4.7. The computation for layer 3 is given below, whereas the computations for layer 1, 2 & 4 remain same as that of layer 1, 2 & 5 respectively as in section 4.2.3.

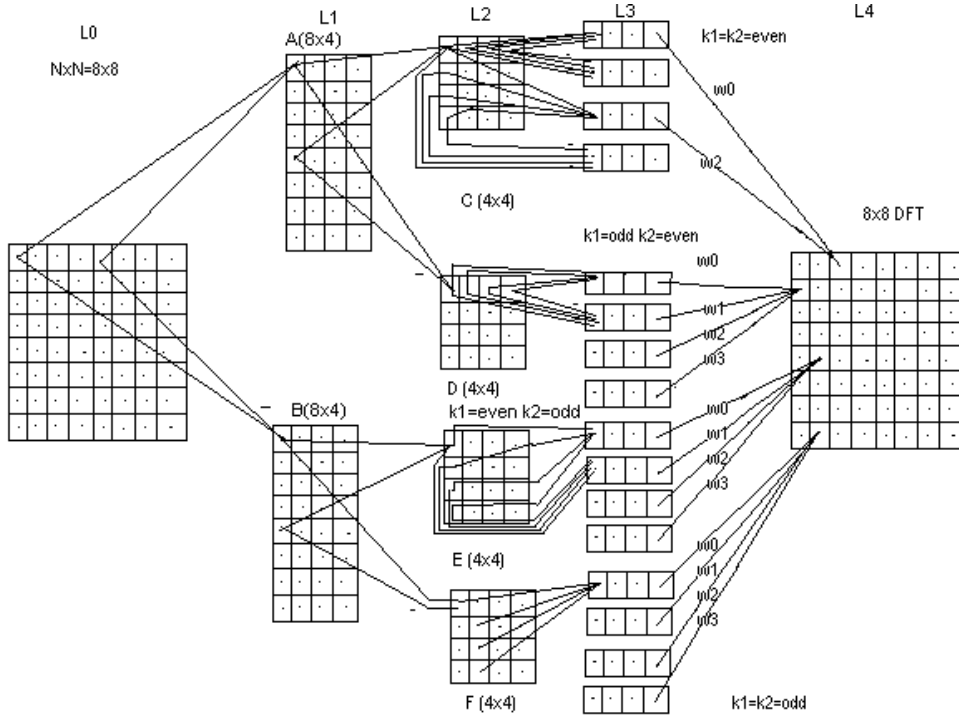


Fig. 4.7: M spacing based four layer architecture for 8×8 DFT

Layer 3 computations

C group

$$Y_{0,0}^0 = C(0,0)+C(1,0)+C(2,0)+C(3,0)+C(0,1)+C(1,1)+C(2,1)+C(3,1)+C(0,2)+C(1,2)+C(2,2)+C(3,2) \\ +C(0,3)+C(1,3)+C(2,3)+C(3,3)$$

$$Y_{4,0}^0 = C(0,0)-C(1,0)+C(2,0)-C(3,0)+C(0,1)-C(1,1)+C(2,1)-C(3,1)+C(0,2)-C(1,2)+C(2,2)-C(3,2) \\ +C(0,3)-C(1,3)+C(2,3)-C(3,3)$$

$$Y_{0,4}^0 = C(0,0)+C(1,0)+C(2,0)+C(3,0)-C(0,1)-C(1,1)-C(2,1)-C(3,1)+C(0,2)+C(1,2)+C(2,2)+C(3,2) - \\ C(0,3)-C(1,3)-C(2,3)-C(3,3)$$

$$Y_{4,4}^0 = C(0,0)-C(1,0)+C(2,0)-C(3,0)-C(0,1)+C(1,1)-C(2,1)+C(3,1)+C(0,2)-C(1,2)+C(2,2)-C(3,2)- \\ C(0,3)+C(1,3)-C(2,3)+C(3,3)$$

$$Y_{2,0}^0 = C(0,0)+C(0,2)-C(2,0)-C(2,2)+C(0,1)+C(0,3)-C(2,1)-C(2,3)$$

$$Y_{2,0}^2 = C(1,0)+C(1,2)-C(3,0)-C(3,2)+C(1,1)+C(1,3)-C(3,1)-C(3,3)$$

$$Y_{0,2}^0 = C(0,0)+C(2,0)+C(1,0)+C(3,0)-C(0,2)-C(2,2)-C(1,2)-C(3,2)$$

$$Y_{0,2}^2 = C(0,1)+C(2,1)+C(1,1)+C(3,1)-C(0,3)-C(2,3)-C(1,3)-C(3,3)$$

$$Y_{2,4}^0 = C(0,0)+C(0,2)+C(2,1)+C(2,3)-C(0,1)-C(0,3)-C(2,0)-C(2,2)$$

$$Y_{2,4}^2 = C(1,0)+C(1,2)+C(3,1)+C(3,3)-C(1,1)-C(1,3)-C(3,0)-C(3,2)$$

$$Y_{4,2}^0 = C(0,0)+C(2,0)+C(1,2)+C(3,2)-C(0,2)-C(2,2)-C(1,0)-C(3,0)$$

$$Y_{4,2}^2 = C(0,1)+C(2,1)+C(1,3)+C(3,3)-C(0,3)-C(2,3)-C(1,1)-C(3,1)$$

$$Y_{6,2}^0 = C(0,0)+C(1,1)+C(2,2)+C(3,3)-C(0,2)-C(1,3)-C(2,0)-C(3,1)$$

$$Y_{6,2}^2 = C(0,1)+C(1,2)+C(2,3)+C(3,0)-C(0,3)-C(1,0)-C(2,1)-C(3,2)$$

$$Y_{2,2}^0 = C(0,0)+C(2,2)-C(1,1)-C(3,3)-C(0,2)-C(2,0)+C(1,3)+C(3,1)$$

$$Y_{2,2}^2 = C(0,1)+C(2,3)-C(1,2)-C(3,0)-C(0,3)-C(2,1)+C(1,0)+C(3,2)$$

D group

$$Y_{1,0}^0 = D(0,0)+D(0,1)+D(0,2)+D(0,3) \quad Y_{1,0}^1 = D(1,0)+D(1,1)+D(1,2)+D(1,3)$$

$$Y_{1,0}^2 = D(2,0)+D(2,1)+D(2,2)+D(2,3) \quad Y_{1,0}^3 = D(3,0)+D(3,1)+D(3,2)+D(3,3)$$

$$Y_{1,2}^0 = D(0,0)-D(0,2)-D(2,1)+D(2,3) \quad Y_{1,2}^1 = D(1,0)-D(1,2)-D(3,1)+D(3,3)$$

$$Y_{1,2}^2 = D(0,1)-D(0,3)+D(2,0)-D(2,2) \quad Y_{1,2}^3 = D(1,1)-D(1,3)+D(3,0)-D(3,2)$$

$$Y_{3,2}^0 = D(0,0)-D(0,2)+D(2,1)-D(2,3) \quad Y_{3,2}^1 = -D(1,1)+D(1,3)+D(3,0)-D(3,2)$$

$$Y_{3,2}^2 = D(0,1)-D(0,3)-D(2,0)+D(2,2) \quad Y_{3,2}^3 = D(1,0)-D(1,2)+D(3,1)-D(3,3)$$

$$Y_{1,4}^0 = D(0,0)-D(0,1)+D(0,2)-D(0,3) \quad Y_{1,4}^1 = D(1,0)-D(1,1)+D(1,2)-D(1,3)$$

$$Y_{1,4}^2 = D(2,0)-D(2,1)+D(2,2)-D(2,3) \quad Y_{1,4}^3 = D(3,0)-D(3,1)+D(3,2)-D(3,3)$$

E group

$$Y_{0,1}^0 = E(0,0)+E(1,0)+E(2,0)+E(3,0) \quad Y_{0,1}^1 = E(0,1)+E(1,1)+E(2,1)+E(3,1)$$

$$Y_{0,1}^2 = E(0,2)+E(1,2)+E(2,2)+E(3,2) \quad Y_{0,1}^3 = E(0,3)+E(1,3)+E(2,3)+E(3,3)$$

$$Y_{2,1}^0 = E(0,0)-E(1,2)-E(2,0)+E(3,2) \quad Y_{2,1}^1 = E(0,1)-E(1,3)-E(2,1)+E(3,3)$$

$$Y_{2,1}^2 = E(0,2)+E(1,0)-E(2,2)-E(3,0) \quad Y_{2,1}^3 = E(0,3)+E(1,1)-E(2,3)-E(3,1)$$

$$Y_{4,1}^0 = E(0,0)-E(1,0)+E(2,0)-E(3,0) \quad Y_{4,1}^1 = E(0,1)-E(1,1)+E(2,1)-E(3,1)$$

$$Y_{4,1}^2 = E(0,2)-E(1,2)+E(2,2)-E(3,2) \quad Y_{4,1}^3 = E(0,3)-E(1,3)+E(2,3)-E(3,3)$$

$$Y_{6,1}^0 = E(0,0)+E(1,2)-E(2,0)-E(3,2) \quad Y_{6,1}^1 = E(0,1)+E(1,3)-E(2,1)-E(3,3)$$

$$Y_{6,1}^2 = E(0,2)-E(1,0)-E(2,2)+E(3,0) \quad Y_{6,1}^3 = E(0,3)-E(1,1)-E(2,3)+E(3,1)$$

F group

$$Y_{1,1}^0 = F(0,0)-F(1,3)-F(2,2)-F(3,1) \quad Y_{1,1}^1 = F(0,1)+F(1,0)-F(2,3)-F(3,2)$$

$$Y_{1,1}^2 = F(0,2)+F(1,1)+F(2,0)-F(3,3) \quad Y_{1,1}^3 = F(0,3)+F(1,2)+F(2,1)+F(3,0)$$

$$Y_{3,1}^0 = F(0,0)-F(1,1)+F(2,2)+F(3,3) \quad Y_{3,1}^1 = F(0,1)-F(1,2)+F(2,3)+F(3,0)$$

$$Y_{3,1}^2 = F(0,2)-F(1,3)-F(2,0)+F(3,1) \quad Y_{3,1}^3 = F(0,3)+F(1,0)-F(2,1)+F(3,2)$$

$$Y_{5,1}^0 = F(0,0)+F(1,3)-F(2,2)+F(3,1) \quad Y_{5,1}^1 = F(0,1)-F(1,0)-F(2,3)+F(3,2)$$

$$Y_{5,1}^2 = F(0,2)-F(1,1)+F(2,0)+F(3,3) \quad Y_{5,1}^3 = F(0,3)-F(1,2)+F(2,1)-F(3,0)$$

$$Y_{7,1}^0 = F(0,0)+F(1,1)+F(2,2)+F(3,3) \quad Y_{7,1}^1 = F(0,1)+F(1,2)+F(2,3)-F(3,0)$$

$$Y_{7,1}^2 = F(0,2)+F(1,3)-F(2,0)-F(3,1) \quad Y_{7,1}^3 = F(0,3)-F(1,0)-F(2,1)-F(3,2)$$

4.2.4.1 Sample computation

For the data matrix $[x]$ shown below, the computation of 8×8 point DFT is as follows

$$[x] = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 & 0 & 2 & 1 \\ 0 & 3 & 1 & 3 & 2 & 1 & 1 & 1 \\ 1 & 2 & 3 & 0 & 0 & 2 & 1 & 3 \\ 3 & 1 & 0 & 2 & 0 & 1 & 1 & 3 \\ 0 & 2 & 3 & 1 & 2 & 1 & 2 & 1 \\ 1 & 2 & 0 & 1 & 3 & 1 & 0 & 1 \\ 1 & 1 & 2 & 2 & 0 & 0 & 2 & 2 \\ 2 & 2 & 0 & 0 & 1 & 3 & 1 & 2 \end{bmatrix}$$

Layer 1

A

$$\begin{bmatrix} 2 & 2 & 5 & 3 \\ 2 & 4 & 2 & 4 \\ 1 & 4 & 4 & 3 \\ 3 & 2 & 1 & 5 \\ 2 & 3 & 5 & 2 \\ 4 & 3 & 0 & 2 \\ 1 & 1 & 4 & 4 \\ 3 & 5 & 1 & 2 \end{bmatrix}$$

B

$$\begin{bmatrix} 0 & 2 & 1 & 1 \\ -2 & 2 & 0 & 2 \\ 1 & 0 & 2 & -3 \\ 3 & 0 & -1 & -1 \\ -2 & 1 & 1 & 0 \\ -2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & -1 & -2 \end{bmatrix}$$

Layer 2

C

$$\begin{bmatrix} 4 & 5 & 10 & 5 \\ 6 & 7 & 2 & 6 \\ 2 & 5 & 8 & 7 \\ 6 & 7 & 2 & 7 \end{bmatrix}$$

D

$$\begin{bmatrix} 0 & -1 & 0 & 1 \\ -2 & 1 & 2 & 2 \\ 0 & 3 & 0 & -1 \\ 0 & -3 & 0 & 3 \end{bmatrix}$$

E

$$\begin{bmatrix} -2 & 3 & 2 & 1 \\ -4 & 3 & 0 & 2 \\ 2 & 1 & 2 & -3 \\ 4 & -1 & -2 & -3 \end{bmatrix}$$

F

$$\begin{bmatrix} 2 & 1 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & -1 & 2 & -3 \\ 2 & 1 & 0 & 1 \end{bmatrix}$$

Layer 3

C Group

$$\begin{bmatrix} Y_{0,0}^0 & Y_{4,0}^0 & Y_{0,4}^0 & Y_{4,4}^0 \\ [89 & 3 & -9 & 13] \\ Y_{2,0}^0 & Y_{2,0}^2 & Y_{0,2}^0 & Y_{0,2}^2 \\ [2 & -1 & -4 & -1] \\ Y_{2,4}^0 & Y_{2,4}^2 & Y_{4,2}^0 & Y_{4,2}^2 \\ [6 & 1 & -20 & -3] \\ Y_{2,2}^0 & Y_{2,2}^2 & Y_{6,2}^0 & Y_{6,2}^2 \\ [-1 & 2 & 1 & 2] \end{bmatrix}$$

D Group

$$\begin{bmatrix} Y_{1,0}^0 & Y_{1,0}^1 & Y_{1,0}^2 & Y_{1,0}^3 \\ [0 & 3 & 2 & 0] \\ Y_{1,2}^0 & Y_{1,2}^1 & Y_{1,2}^2 & Y_{1,2}^3 \\ [-4 & 2 & -2 & -1] \\ Y_{3,2}^0 & Y_{3,2}^1 & Y_{3,2}^2 & Y_{3,2}^3 \\ [4 & 1 & -2 & -10] \\ Y_{1,4}^0 & Y_{1,4}^1 & Y_{1,4}^2 & Y_{1,4}^3 \\ [0 & -3 & -2 & 0] \end{bmatrix}$$

E Group

$$\begin{bmatrix} Y_{0,1}^0 & Y_{0,1}^1 & Y_{0,1}^2 & Y_{0,1}^3 \\ [0 & 6 & 2 & -3] \\ Y_{2,1}^0 & Y_{2,1}^1 & Y_{2,1}^2 & Y_{2,1}^3 \\ [-6 & -3 & -8 & 8] \\ Y_{4,1}^0 & Y_{4,1}^1 & Y_{4,1}^2 & Y_{4,1}^3 \\ [0 & 2 & 6 & -1] \\ Y_{6,1}^0 & Y_{6,1}^1 & Y_{6,1}^2 & Y_{6,1}^3 \\ [-2 & 7 & 8 & 0] \end{bmatrix}$$

F Group

$$\begin{bmatrix} Y_{1,1}^0 & Y_{1,1}^1 & Y_{1,1}^2 & Y_{1,1}^3 \\ [-3 & 4 & 0 & 2] \\ Y_{3,1}^0 & Y_{3,1}^1 & Y_{3,1}^2 & Y_{3,1}^3 \\ [2 & 0 & -1 & 2] \\ Y_{5,1}^0 & Y_{5,1}^1 & Y_{5,1}^2 & Y_{5,1}^3 \\ [3 & 4 & 0 & -2] \\ Y_{7,1}^0 & Y_{7,1}^1 & Y_{7,1}^2 & Y_{7,1}^3 \\ [6 & -4 & 1 & 2] \end{bmatrix}$$

Layer 4

$$\begin{bmatrix} 89 & 6.364-j4.1213 & -4+j & -6.364-j0.1213 & -9 & -6.364+j0.1213 & -4-j & 6.364+j4.1213 \\ 2.1213-j4.1213 & -1.5858-j4.2426 & -1.8787+j1.2929 & 3.4142-j2.4142 & -2.1213+j4.1213 & -1.2426+j1.4142 & -3.7782+j4.364 & 1.7574-j4142 \\ 2+j & -13.7782+j4.4645 & -1-j2 & -6.9497+j3.0503 & 6-j & 1.7782+j11.5355 & 1+j2 & 2.9497+j12.9497 \\ -2.1213-j0.1213 & 0.5858-j0.4142 & 11.7782+j8.364 & -4.4142-j4.2426 & 2.1213+j0.1213 & 10.2426-j2.4142 & -6.1213-j2.7071 & 7.2426+j1.4142 \\ 3 & 2.1212-j6.7071 & -20+j3 & -2.1213+j5.2929 & 13 & -2.1213-j5.2929 & -20-j3 & 2.1213+j6.7071 \\ -2.1213+j0.1213 & 7.2426-j1.4142 & -6.1213+j2.7071 & 10.2426+j2.4142 & 2.1213-j0.1213 & -4.4142+j4.2426 & 11.7782-j8.364 & 0.5858+j4142 \\ 2-j & 2.9497-j12.9497 & 1-j2 & 1.7782-j11.5355 & 6+j & -6.9497-j3.0503 & -1+j2 & -13.7782-4.4645 \\ 2.1213+j4.1213 & 1.7574+j0.4142 & -3.7782-j4.364 & -1.2426-j1.4142 & -2.1213-j4.1213 & 3.4142+j2.4142 & -1.8787-j1.2929 & -1.5858+j4.2426 \end{bmatrix}$$

4.2.5 Proposed architecture for $N \times N$ DFT

In the five layer architecture, Layer 3 and 4 consists of four groups C, D, E and F. The number of computations for a Y_{k_1, k_2}^p in layer 3 and 4 is $M - 1$ and $dm - 1$ respectively, i.e., the number of computations in layer 4 varies. On analysis it is seen that Y_{k_1, k_2}^p with different $\gcd(k_1, k_2, M)$ are evenly distributed between the four groups when $((N))_4 = 2$ and hence the computations are also evenly distributed in layer 4. When $((N))_4 = 0$ and N not a power of 2, Y_{k_1, k_2}^p with different $\gcd(k_1, k_2, M)$ are unevenly distributed between the four groups. When N is a power of 2, Y_{k_1, k_2}^p with $\gcd(k_1, k_2, M) = 1$ which are completely computed in layer 3 are in group D, E and F, whereas the remaining are in group C. Hence there is no computation block for group D, E and F in layer 4 when N is a power of 2, as in fig 4.6. This irregularity is rectified in four layer architecture by combining layer 3 and 4 as in section 4.2.4. In the four layer architecture, when $((N))_4 = 0$, even though the number of computations in layer 3 is different for each group, the number of Y_{k_1, k_2}^p to be computed is same. When $((N))_4 = 2$, the number of computations in each group will also be same. Due to this advantage, further analysis is done only for four layer architecture.

Four layer M spacing based architecture for the computation of 2-D DFT for $N = 4, 6, 8, 10$ and 12 are developed. Based on the analysis of these architectures, a generalized architecture is developed for any even N , as in fig 4.8.

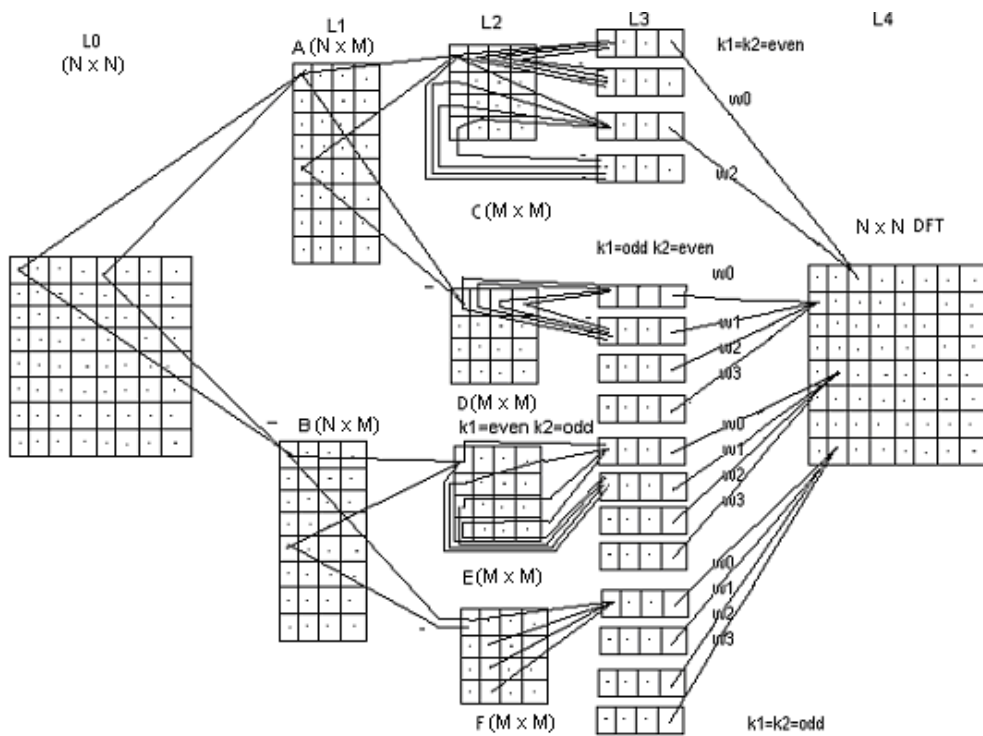


Fig. 4.8: Four layer architecture for $N \times N$ DFT

In layer 1, A and B are of dimension $N \times M$. C, D, E and F in layer 3 are of dimension $M \times M$. Layer 4 is of dimension $N \times N$. The computation of layer 1, 2 and 4 uses a common algorithm and the total size of each layer is $N \times N$. The size of layer 3 is N^2 and $N^2 + 2.N + 4$ when N is a power of 2 and $N/2$ prime respectively. In general, the size of layer 3 can be computed as follows.

Number of Y_{k_1, k_2}^p in a basic DFT coefficient = M/dm

$$\therefore \text{Total number of } Y_{k_1, k_2}^p \text{ in } N = \sum_{dm} nb_{dm} \cdot \frac{M}{dm}$$

Analysis of layer 3 for different sizes and coefficients are carried out in the following sections to develop a common algorithm for its computation for any even N .

4.2.5.1 Number of additions for each Y_{k_1, k_2}^p in layer 3

When the data at the above four points are clubbed together and considered as a single unit, then

$$((n_1.k_1 + n_2.k_2))_N = p \text{ or } p + M \quad (4.9)$$

$$((n_1.k_1 + (n_2 + M)k_2))_N = p \text{ or } p + M \quad (4.10)$$

$$(((n_1 + M)k_1 + n_2.k_2))_N = p \text{ or } p + M \quad (4.11)$$

$$(((n_1 + M)k_1 + (n_2 + M)k_2))_N = p \text{ or } p + M \quad (4.12)$$

Adding (4.9), (4.10), (4.11) and (4.12)

$$\begin{aligned} & ((n_1.k_1 + n_2.k_2))_N + ((n_1.k_1 + (n_2 + M)k_2))_N + (((n_1 + M)k_1 + n_2.k_2))_N + (((n_1 + M)k_1 + (n_2 + M)k_2))_N \\ & = 4p \text{ or } 4(p + M) \end{aligned}$$

$$\text{i.e., } 4((n_1.k_1 + n_2.k_2))_N = 4p \text{ or } 4(p + M) \quad (4.13)$$

Since N is even, from theorem B.4.1 (4.13) becomes,

$$((n_1.k_1 + n_2.k_2))_M = p \text{ or } (p + M) \quad (4.14)$$

As a special case, if $((N))_4 = 0$, from theorem B.4.1 (4.13) becomes,

$$((n_1.k_1 + n_2.k_2))_{M/2} = p \text{ or } (p + M)$$

From theorem B.4.2, the linear congruence equation (4.14) has solution if and only if $dm \mid p$ where $dm = \gcd(k_1, k_2, M)$. E.g., when $N = 8$ and $\gcd(k_1, k_2, M) = 2$, then Y_{k_1, k_2}^p exists only for $p = 0$ and 2 , whereas for $\gcd(k_1, k_2, M) = 1$, Y_{k_1, k_2}^p exists for $0 \leq p \leq M - 1$, which supports theorem 3.1.

If $\gcd(k_1, M) = 1$ or $\gcd(k_2, M) = 1$ then, from theorem B.4.3, (4.14) has exactly M solutions. E.g. when $N = 8$ $Y_{1,1}^2$ has exactly four F terms in its expression, as in F group computations in layer 3 of section 4.2.4. From theorem B.4.4, (4.14) has exactly ' $dm.M$ ' incongruent solutions. E.g. when $N = 8$, $Y_{2,4}^0$ has eight C terms in its expression, as in C group computations in layer 3 of section 4.2.4.

Therefore the number of terms to be added in layer 3 ranges from M to M^2 since dm varies from 1 to M .

4.2.5.2 Development of algorithm for layer 3 computation

The expression for the computation of layer 3 is simplified, based on the relation between the terms in the expression for the Y_{k_1, k_2}^p , as shown below for $N = 8$.

Computation in layer 3 for $N = 8$

C group

$$Y_{0,0}^0 = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} C(i, j)$$

$$Y_{4,0}^0 = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} (-1)^i C(i, j)$$

$$Y_{0,4}^0 = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} (-1)^j C(i, j)$$

$$Y_{4,4}^0 = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} (-1)^{i+j} C(i, j)$$

$$Y_{2,2}^l = \sum_{i=0}^1 \sum_{j=0}^{M-1} (-1)^i C(2.i + l, j), \quad 0 \leq l \leq 2$$

$$Y_{0,2}^l = \sum_{j=0}^1 \sum_{i=0}^{M-1} (-1)^j C(i, 2.j + l), \quad 0 \leq l \leq 2$$

$$Y_{2,4}^l = \sum_{i=0}^1 \sum_{j=0}^{M-1} (-1)^{i+j} C(2.i + l, j) \quad 0 \leq l \leq 2$$

For $l = 0, 2$

For $j = 0$ to 1

For $i = 0$ to $M - 1$

$$k1 = 4, k2 = 2, n1 = i, n2 = j + l/2$$

If $((n1.k1 + n2.k2))_N \geq M$, temp = $-C(i, 2.j + l/2)$ else temp = $C(i, 2.j + l/2)$

$$Y_{4,2}^l = Y_{4,2}^l + \text{temp}$$

(For the computation of $Y_{2,0}$ and $Y_{6,2}$ replace $C(i, 2.j + l/2)$ with $C(i, i + 2.j + l/2)$ in the above algorithm)

D group

For $p = 0$ to $M - 1$

For $i = 0$ to $M - 1$

$$k1 = 1, k2 = 0, n1 = ((p + (M - k2)i))_M, n2 = ((k1.i))_M$$

If $((n1.k1 + n2.k2))_N \geq M$

$$\text{temp} = -D(((p + (M - k2)i))_M, ((k1.i))_M)$$

Else temp = $D(((p + (M - k2)i))_M, ((k1.i))_M)$

$$Y_{1,0}^p = Y_{1,0}^p + \text{temp}$$

(For the computation of $Y_{1,2}^p$, $Y_{3,2}^p$ and $Y_{1,4}^p$ replace $k1$ and $k2$ with the corresponding values.)

E group

For $p = 0$ to $M - 1$

For $i = 0$ to $M - 1$

$$k1 = 0, k2 = 1, n1 = i, n2 = ((p + (M - k1)i))_M$$

If $((n1.k1 + n2.k2))_N \geq M$

$$\text{temp} = -E(i, ((p + (M - k1)i))_M)$$

$$\text{Else temp} = E(i, ((p + (M - k1)i))_M)$$

$$Y_{0,1}^p = Y_{0,1}^p + \text{temp}$$

(For the computation of $Y_{2,1}^p$, $Y_{4,1}^p$ and $Y_{6,1}^p$ replace $k1$ and $k2$ with the corresponding values.)

F group

For $p = 0$ to $M - 1$

For $i = 0$ to $M - 1$

$$k1 = 1, k2 = 1, n1 = i, n2 = ((p + (M - k1)i))_M$$

If $((n1.k1 + n2.k2))_N \geq M$

$$\text{temp} = -F(i, ((p + (M - k1)i))_M)$$

$$\text{Else temp} = F(i, ((p + (M - k1)i))_M)$$

$$Y_{1,1}^p = Y_{1,1}^p + \text{temp}$$

(For the computation of $Y_{3,1}^p$, $Y_{5,1}^p$ and $Y_{7,1}^p$ replace $k1$ and $k2$ with the corresponding values.)

From the algorithm described, it can be seen that the indices of the terms in the expression for Y_{k_1, k_2}^p are related. Hence, if the indices of one of the terms are available, others can be derived. To find the indices for one of the terms in Y_{k_1, k_2}^p , it is necessary to compute the particular solution (n_1, n_2) for (4.14). The computation of particular solution is explained in the next section. Subsequently, the indices of the next term to be added/subtracted are computed. Based on the analysis of equation /algorithms for the computation of layer 3 for $N = 8$, it is found that the relation between the index of the terms in the expression for Y_{k_1, k_2}^p depends on k_1 and k_2 . Appendix C shows the computation of layer 3 for $N = 4, 6, 10$ and 12 . On analysis of those computations, it is found that the relation between index of the terms also depends on $\text{gcd}(k_1, k_2, M)$, $\text{gcd}(k_2, N)$, $\text{gcd}(k_2, M)$ and N . The general index relation between the terms, for any even N , is derived based on the above analysis. The next term is to be added if $((n_1.k_1 + n_2.k_2))_N < M$, or else subtract it.

4.2.5.3 Particular solution

It is necessary to find the particular solution (n_1, n_2) for the basic equation $((n_1.k_1 + n_2.k_2))_N = p$ or $p + M$, i.e., to find the position of one of the data points of Y_{k_1, k_2}^p whose frequency indices are k_1 and k_2

and the phase index is p . Since the four data points are clubbed together it is enough to find one particular solution for (4.14) in order to obtain the general solution.

The particular solution for (4.14) can be obtained by 1) Trial and error, 2) Using extended Euclidean algorithm 3) Combination of visual approach and other methods 4) Modified trial and error method.

4.2.5.3.1 Trial and error method

One of the most general methods of finding the particular solution of linear Diophantine equation [3] is by trial and error. A Diophantine equation is one in which the solutions are required to be integers as stated in B.2. Congruence mod N is an equivalence relation. Hence, congruence has many of the same properties as ordinary equations. The particular solution for (4.14), is to find one integer value for n_1 and n_2 for a given k_1 , k_2 and p . Since $0 \leq n_1, n_2 \leq M - 1$, find n_1 and n_2 which satisfies (4.14) by trial and error. Algorithm is as follows:

1. Given k_1, k_2 and p
 - For $n_1 = 0, M - 1$
 - For $n_2 = 0, M - 1$
 - If $((n_1.k_1 + n_2.k_2))_M = p$
 - select n_1 and n_2 and return
 - else if $((n_1.k_1 + n_2.k_2))_M = p + M$
 - select n_1 and n_2 and return

4.2.5.3.2 Using extended Euclidean algorithm

The extended Euclidean algorithm as described in B.6 is an extension to the Euclidean algorithm to find the gcd of integers a and b : it also finds the integers s and t in the Bezout's identity (see B.3), $a.s + b.t = \gcd(a, b)$. In (4.14) two cases arise depending upon the value of k_1 and k_2 .

case 1: $\gcd(k_1, k_2) = 1$

When $\gcd(k_1, k_2) = 1$, (4.14) can be initially considered as if a linear Diophantine equation as shown below:

$$n_1.k_1 + n_2.k_2 = p$$

By extended Euclidean algorithm find $\gcd(k_1, k_2)$ and s and t such that

$$k_1.s + k_2.t = \gcd(k_1, k_2)$$

Multiplying by p

$$k_1.s.p + k_2.t.p = \gcd(k_1, k_2).p.$$

Divide it by $\gcd(k_1, k_2)$

$$k_1 \cdot \frac{s.p}{\gcd(k_1, k_2)} + k_2 \cdot \frac{t.p}{\gcd(k_1, k_2)} = p$$

Particular solutions are

$$n_1 = \left(\left(\frac{((s))_M \cdot p}{(\gcd(k_1, k_2))_M} \right) \right)_M \text{ and } n_2 = \left(\left(\frac{((t))_M \cdot p}{(\gcd(k_1, k_2))_M} \right) \right)_M$$

case 2: $\gcd(k_1, k_2) \neq 1$

When $\gcd(k_1, k_2) \neq 1$, (4.14) can be written as

$$n_1.k_1 + n_2.k_2 + M.q = p$$

The above equation is the most general one. First, factor $\gcd(k_1, k_2)$ out of the first two terms:

$$\gcd(k_1, k_2) \cdot \left(\frac{k_1}{\gcd(k_1, k_2)} n_1 + \frac{k_2}{\gcd(k_1, k_2)} n_2 \right) + M.q = p$$

Let $w = \frac{k_1}{\gcd(k_1, k_2)} n_1 + \frac{k_2}{\gcd(k_1, k_2)} n_2$, then the above equation becomes

$$\gcd(k_1, k_2) \cdot w + M.q = p$$

This two variable equation is solvable using extended Euclidean algorithm. So find a particular solution for w and ignore q . Now find n_1 and n_2 from

$$\frac{k_1}{\gcd(k_1, k_2)} n_1 + \frac{k_2}{\gcd(k_1, k_2)} n_2 = w$$

This is a two variable equation, which can be solved as in case 1.

Algorithm to find the particular solution for $n_1.k_1 + n_2.k_2 + M.q = p$ is as follows:

1. Given k_1, k_2, M and p
Find $\gcd(k_1, k_2)$
2. For the reduced equation $\gcd(k_1, k_2) \cdot w + M.q = p$ apply extended Euclidean algorithm to find $\gcd(\gcd(k_1, k_2), M)$ and s and t such that
 $\gcd(k_1, k_2) \cdot s + M \cdot t = \gcd(\gcd(k_1, k_2), M)$

3. Calculate $w = \frac{s \cdot p}{\gcd(\gcd(k_1, k_2), M)}$

4. For the equation $w = \frac{k_1}{\gcd(k_1, k_2)} n_1 + \frac{k_2}{\gcd(k_1, k_2)} n_2$ apply extended Euclidean

algorithm to find $\gcd\left(\frac{k_1}{\gcd(k_1, k_2)}, \frac{k_2}{\gcd(k_1, k_2)}\right)$ and u and v such that

$$\gcd\left(\frac{k_1}{\gcd(k_1, k_2)}, \frac{k_2}{\gcd(k_1, k_2)}\right) = \frac{k_1}{\gcd(k_1, k_2)} \cdot u + \frac{k_2}{\gcd(k_1, k_2)} \cdot v$$

5. Calculate $n_1 = \left(\left(\frac{u.w}{\gcd\left(\frac{k_1}{\gcd(k_1, k_2)}, \frac{k_2}{\gcd(k_1, k_2)}\right)} \right) \right)_M$ and

$$n_2 = \left(\left(\frac{v.w}{\gcd\left(\frac{k_1}{\gcd(k_1, k_2)}, \frac{k_2}{\gcd(k_1, k_2)}\right)} \right) \right)_M$$

The above algorithm can be illustrated with an example. Given $k_1 = 2$, $k_2 = 6$, and $p = 4$ for $N = 20$.

$$((2.n_1 + 6.n_2))_{10} = 4.$$

Then

$$2.n_1 + 6.n_2 + 10.q = 4 \text{ for some } q.$$

Set

$$w = \frac{2}{\gcd(2,6)}.n_1 + \frac{6}{\gcd(2,6)}.n_2$$

Then the above equation becomes, $\gcd(2, 6).w + 10.q = 4$

$$w + 5.q = 2.$$

$w = -3$, $q = 1$, is a particular solution. Ignore q and substitute for $((w))_{10}$:

$$7 = \frac{2}{\gcd(2,6)}.n_1 + \frac{6}{\gcd(2,6)}.n_2$$

$$n_1 + 3.n_2 = 7$$

$\therefore n_1 = 1$ and $n_2 = 2$ is a particular solution.

4.2.5.3.3 *Combination of visual approach and other methods*

By direct computation of particular solution for certain Y_{k_1, k_2}^p based on the analysis of visual representation, as shown below, and the rest of the coefficients may be calculated by either of the above two methods:

Analysis of the visual representation has enabled to come out with the particular solution of certain Y_{k_1, k_2}^p even without calculating the same. E.g., whatever be the value of k_1 and k_2 , if $p = 0$ then $n_1 = 0$ and $n_2 = 0$ is a particular solution. Similarly when $k_2 = 0$, then $n_2 = 0$, and $n_1 = p/k_1$ provided $((p/k_1))_1 = 0$ else $n_1 = (p + M)/k_1$. Similarly when $k_1 = 0$, then $n_1 = 0$, and $n_2 = p/k_2$ provided $((p/k_2))_1 = 0$ else $n_2 = (p + M)/k_2$.

4.2.5.3.4 *Modified trial and error method*

On analysis of the visual representation of Y_{k_1, k_2}^p for different values of N , it is found that either of n_1 or n_2 has a maximum value of 2. So trial and error method can be limited to checking those values only. This method takes the minimum time for obtaining particular solution for all Y_{k_1, k_2}^p . The algorithm is described below:

1. Given k_1, k_2 and p
2. if $p = 0$, then $n_1 = 0$ and $n_2 = 0$
3. else if $k_2 = 0$, then $n_2 = 0$, and $n_1 = p/k_1$ provided $((p/k_1))_1 = 0$ else $n_1 = (p + M)/k_1$
4. else if $k_1 = 0$, then $n_1 = 0$ and $n_2 = p/k_2$ provided $((p/k_2))_1 = 0$ else $n_2 = (p + M)/k_2$
5. else if p/k_1 is an integer, then $n_1 = p/k_1$ and $n_2 = 0$
6. else for $i = 0$ to $M - 1$
 - if $((p - i.k_1)/k_2)_1 = 0$
 - $n_2 = (p - i.k_1)/k_2, n_1 = i$, return
 - else if $((p + M - i.k_1)/k_2)_1 = 0$
 - $n_2 = (p + M - i.k_1)/k_2, n_1 = i$, return
 - else if $((p - i.k_2)/k_1)_1 = 0$
 - $n_1 = (p - i.k_2)/k_1, n_2 = i$, return
 - else $((p + M - i.k_2)/k_1)_1 = 0$
 - $n_1 = (p + M - i.k_2)/k_1, n_2 = i$, return
7. end

4.2.5.3.5 Simulation results

Three algorithms, developed to find the particular solution in the M spacing based 2-D DFT computational scheme namely, trial and error method, extended Euclidean and modified trial and error method are simulated using Matlab® 7.0. Time of execution in seconds, on Intel® Pentium®4 CPU 1.5 GHZ, machine, for the three methods is shown in table 4.3 and plotted in fig. 4.9. Simulation results show that modified trial and error method performs better for $N > 8$.

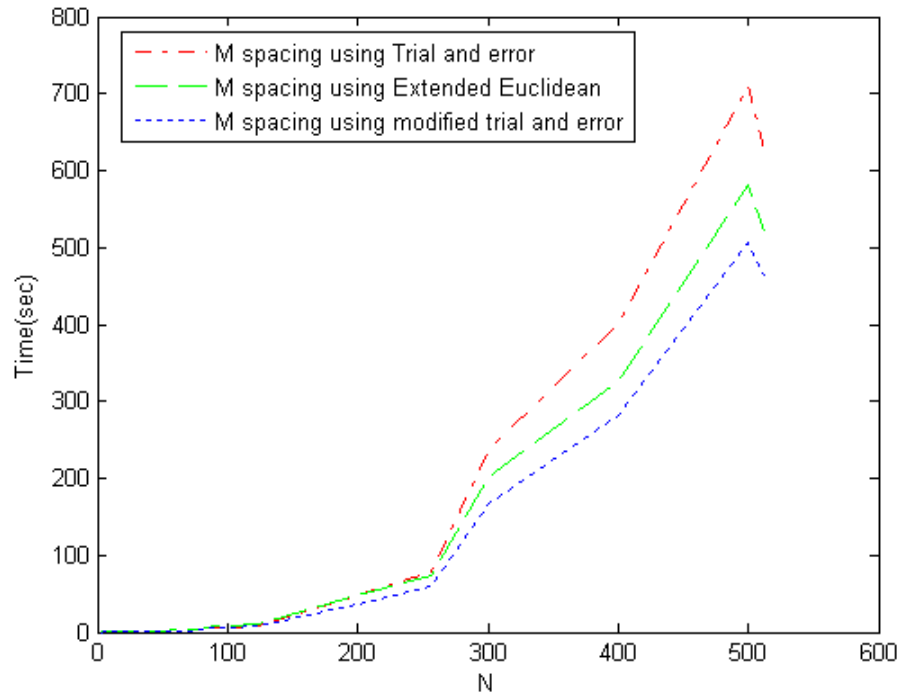


Fig. 4.9: Comparison of execution time when employing different particular solution

Table 4.3: Execution time of different methods for particular solution

N	Trial and	Extended Euclidean	Modified Trial and
4	0	0.015	0
6	0.016	0.031	0.016
8	0.015	0.031	0.016
10	0.032	0.047	0.032
12	0.063	0.141	0.062
14	0.062	0.079	0.047
16	0.078	0.172	0.079
18	0.125	0.25	0.125
20	0.141	0.266	0.125
22	0.187	0.266	0.109
24	0.296	0.422	0.281
26	0.218	0.328	0.172
28	0.296	0.39	0.234
30	0.468	0.672	0.438
32	0.328	0.453	0.328
34	0.359	0.516	0.328
36	0.625	0.922	0.547
38	0.422	0.688	0.406
40	0.672	0.968	0.594
42	0.829	1.265	0.796
44	0.688	0.969	0.625
46	0.672	1.015	0.61
48	1.109	1.641	1.047
50	0.938	1.39	0.86
60	2.391	3.219	2.032
64	1.547	2.156	1.344
70	2.625	3.531	2.188
80	3.625	4.531	3.031
90	6.828	8.375	5.407
100	6.329	7.641	5.141
124	8.985	10.704	7.359
200	48.688	47.984	36.688
256	77.781	74.281	59.969
300	237.062	203.234	166.359
400	400.203	325.719	280.828
500	710.813	580.172	505.657
512	628.344	520.031	460.969

4.2.5.4 M spacing based algorithm for any even N

An algorithm is developed based on the above analysis and the same is verified for different values of N for $N = 4$ to 300 and suitably modified so that the algorithm gives exact result for any even N . Algorithm for layer 1, 2 and 4 are same as in section 4.2.3. Layer 3 computations give Y_{k_1, k_2}^p of all the basic DFT coefficients. Complete set of DFT coefficients are computed in layer 5 as per the steps 3, 4 and 5 given in the algorithm in section 3.3. The important steps of M spacing algorithm for DFT computation are as follows.

1. Compute indices (k_1, k_2) of all the basic DFT coefficients and the no. of basic DFT ($no_of_basicDFT$) as in section 3.2.6 and 3.2.5.
2. Algorithm for computing Y_{k_1, k_2}^p of all the basic DFT coefficients

Layer 1 and 2

Layer 1 and 2 computations are same as in section 4.2.3.

Layer 3

(In layer 3, for every basic DFT coefficient, corresponding Y_{k_1, k_2}^p are computed)

For $q = 1$ to $no_of_basicDFT$

compute $dm = \gcd(k1(q), k2(q), M)$, $h = \gcd(k1(q), M)$, $v = \gcd(k2(q), M)$, $z = \gcd(k2(q), N)$

If $((k1(q)))_2 = 0$ ---depending on the nature of (k_1, k_2) C, D, E or F block is selected for computation

if $((k2(q)))_2 = 0$

U = C

else U = E

elseif $((k2(q)))_2 = 0$

U = D

else U = F

For $p = 0$ to $M - 1$ in steps of dm

compute particular solution $(n1, n2)$ using modified trial and error algorithm as in section 4.2.5.3.4

For $r = 0$ to dm

For $s = 0$ to dm

For $t = 0$ to $M/dm - 1$

$next_n1 = ((n1+r.v/dm + k2(q).t))_M$ --computation for index of next element of U

If $((M))_{k2(q)} = 0$

$next_n2 = ((n2+r.(M- k1(q)))_{M/dm+s.M/v+(M- k1(q)).t})_M$ --index when $k_2 \nmid M$

else

$next_n2 = ((n2+((r[N - z]. k1(q)/[2. z.dm]))_{M/dm+ s.M/v + [M- k1(q)].t})_M$ --index when $k_2 \nmid M$

If $((next_n1.k1(q) + next_n2.k2(q)))_N \geq M$ --testing for element to be added or subtracted

$next_term = -U(next_n1, next_n2)$

else

$next_term = U(next_n1, next_n2)$

$Y(k1(q), k2(q), p) = Y(k1(q), k2(q), p) + next_term$

Layer 4

The computations in layer 5 are same as that of the steps 3, 4 and 5 of the algorithm shown in section 3.3.

4.3 Conclusion

High speed of computation and reduced memory requirement were the salient features of the parallel distributed model developed in [88] to implement 2-D DFT for a particular order N such that $((N))_4 = 2$. Version I and II parallel distributed architecture for the computation of 8×8 point DFT (developed by the analysis of the visual representation based on 2×2 DFT) are designed following the above model. The aim was to develop a model for $((N))_4 = 0$ and then combine it with the one developed in [88] to form a generalized architecture for any even N . But since the primitive symbol combinations for group 2 and 3 coefficients differ for $((N))_4 = 2$ and $N = 8$, the architecture developed also differs. Hence a generalized architecture based on the above model is not feasible.

Four layer and five layer M spacing based architectures developed by the analysis of visual representation of 2-D DFT based on 2×2 data, on the other hand, can be used to compute $N \times N$ DFT for any even N and is scalable. Four layer architecture is more suitable for computation due to the irregularity in the structure of layer 4 of the five layer architecture. Since the computations are evenly distributed between different groups, the architecture is highly efficient for $N/2$ prime.

CHAPTER 5

2-D UMRT

In the parallel distributed architectures developed for the computation of 2-D DFT in chapter 4, there are only real additions involved till the penultimate layer. The number of complex multiplication for each coefficient in the last layer is $N/2$. Since only scaling by the twiddle factor is done, each complex multiplication is equivalent to two real multiplications. However multipliers are expensive components. They use large silicon area, consume more power and introduce long latencies into a circuit. In contrast, primitive operators such as adders, subtractors, and shifts are much cheaper in terms of power, area and delay.

The complex multiplications can be avoided, if the signals can be represented in terms of the MRT coefficients [140] and then the only computation required will be the real additions as in (1.9). MRT, in the raw form contains significant redundancy. The analysis of the visual representation explained in section 3.2.4 shows that there are three levels of redundancy in the coefficients. In [167] it was shown that the UMRT coefficients are unique, numerically compact and require only the same memory as required for the original image, when the image size is a power of 2. The number of MRT coefficients remains after removing the third level of redundancy is N^2 when N is a power of 2, as explained in section 3.2.4. But when N is not a power of 2, even after removing the third level of redundancy, the number of MRT coefficients remains is not N^2 . The visual representation is further analyzed to exploit a form of redundancy present between the MRT coefficients, namely derived redundancy [189], so as to obtain unique MRT with N^2 coefficients for any even N . Derived redundancy in 1-D was analyzed in [189].

5.1 Derived redundancy in MRT

The number of MRT coefficients left after eliminating the redundancy, as explained above, is N^2 when N is a power of 2. The transformed coefficients thus occupy the same memory as required for the original image. When N is not a power of two, the number of MRT coefficients after removing the redundancy is more than N^2 . Hence the visual representation of MRT coefficients are analyzed to explore the derived redundancy present between the MRT coefficients.

5.1.1 Analysis of the MRT coefficients for $N = 6$ ($N/2$ prime)

Fig. 5.1 shows that the visual representation of $Y_{0,2}^0$ & $Y_{0,2}^2$ are combined together along with the sign reversed form of $Y_{0,2}^1$, gives $Y_{0,0}^0$. So one of the MRT coefficients of $Y_{0,2}$ can be eliminated since the same can be derived from the other coefficients and $Y_{0,0}^0$. This type of redundancy present between the MRT coefficients is called derived redundancy.

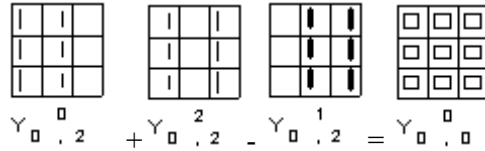


Fig. 5.1: Example 1 for derived redundancy for $N = 6$

Similarly, if the visual representation of $Y_{1,2}^0$ & $Y_{1,2}^2$ are combined along with the sign reversed form of $Y_{1,2}^1$, it will result in the visual representation of $Y_{3,0}^0$ as in fig. 5.2. If the MRT coefficients of $Y_{3,2}, Y_{5,2}, Y_{1,0}$ etc. are combined separately it will result in $Y_{3,0}$. Here the $\text{gcd}(k_1, k_2, M)$ of $Y_{3,2}, Y_{5,2}, Y_{1,0}$ etc. is 1 and that of $Y_{3,0}$ is M . On analysis of the basic DFT coefficients with $\text{gcd}(k_1, k_2, M) = 1$, it is found that one MRT coefficient each can be eliminated, as the same can be derived as shown above. When k_1 & k_2 are even, Y_{k_1, k_2}^p of such DFT coefficients can be combined to get $Y_{0,0}^0$, as shown in table 5.1. The table shows that when $N = 6$, the MRT coefficients are combined to form one of the group 1 coefficients depending on the nature of the frequency index (k_1, k_2) . This can be noticed for higher orders of N , where $N/2$ is prime and has a general relation, $\sum \pm Y_{\text{even}, \text{even}}^p$ will yield $Y_{0,0}^0$, $\sum \pm Y_{\text{odd}, \text{even}}^p$ will yield $Y_{M,0}^0$, $\sum \pm Y_{\text{even}, \text{odd}}^p$ will yield $Y_{0,M}^0$ and $\sum \pm Y_{\text{odd}, \text{odd}}^p$ will yield $Y_{M,M}^0$ when $N/2$ is prime, as shown in table 5.2.

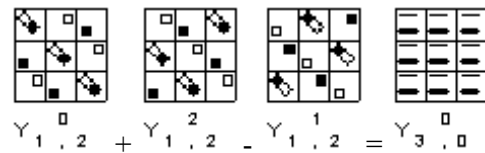


Fig. 5.2: Example 2 for derived redundancy for $N = 6$

Table 5.1: Index relation in derived redundancy - $N = 6$

Combination of MRT coefficients	Resulting MRT
$Y_{0,2}^0 + Y_{0,2}^2 - Y_{0,2}^1$	$Y_{0,0}^0$
$Y_{1,2}^0 + Y_{1,2}^2 - Y_{1,2}^1$	$Y_{3,0}^0$
$Y_{2,1}^0 + Y_{2,1}^2 - Y_{2,1}^1$	$Y_{0,3}^0$
$Y_{1,1}^0 + Y_{1,1}^2 - Y_{1,1}^1$	$Y_{3,3}^0$

For $N = 6$, as in table 5.3, the number of basic DFT coefficients (nb) is 20. These 20 basic DFT coefficients together have 52 MRT coefficients. 4 basic DFT coefficients, where $dm = M$, have one MRT coefficient (i.e., $p = 0$ only) each. One MRT coefficient each can be eliminated from

the remaining 16 basic DFT coefficients, where $dm = 1$. Then there will be $52 - 16 = 36$ UMRT coefficients, as shown in table 5.3, which is nothing but N^2 . Similar features can be seen when $N = 10, 14, 22$ etc. So when $N/2$ is prime, one MRT coefficient from each of the basic DFT coefficients, where $\gcd(k_1, k_2, M) = 1$ can be eliminated so as to obtain UMRT with N^2 coefficients.

Table 5.2: Index relation in derived redundancy - $N/2$ prime

Combination of MRT coefficients	Resulting MRT
$\sum \pm Y_{\text{even,even}}^p$	$Y_{0,0}^0$
$\sum \pm Y_{\text{odd,even}}^p$	$Y_{M,0}^0$
$\sum \pm Y_{\text{even,odd}}^p$	$Y_{0,M}^0$
$\sum \pm Y_{\text{odd,odd}}^p$	$Y_{M,M}^0$

Table 5.3: Derived redundancy when $N = 6$

dm	nb_{dm}	Total no. of MRT Coefficients $\frac{M}{dm} \times nb_{dm}$	No. of MRT coefficients that can be eliminated from each	Total coefficients that can be eliminated	No. of UMRT coefficients
1	16	48	1	16	32
$M=3$	4	4	0	0	4
Total	20	52		16	36

5.1.2 Analysis for $N = 12$ where $((N))_4 = 0$ and N not a power of 2

In fig. 5.3, for $N = 12$, the visual representation of $Y_{1,0}^0$ and $Y_{1,0}^4$ are combined together along with the sign reversed form of $Y_{1,0}^2$, where $\gcd(k_1, k_2, M) = 1$. The result of the combination is the visual representation of $Y_{3,0}^0$, where $\gcd(k_1, k_2, M) = 3$. Similarly the combination of $Y_{1,0}^1$ and $Y_{1,0}^5$ along with the sign reversed $Y_{1,0}^3$ will yield $Y_{3,0}^3$. The combination of visual representation of MRT coefficients with odd p i.e., $\sum \pm Y_{1,0}^{\text{odd}}$, will result in $Y_{3,0}^3$. Similarly the combination of MRT coefficients with even p i.e., $\sum \pm Y_{1,0}^{\text{even}}$ will yield $Y_{3,0}^0$.

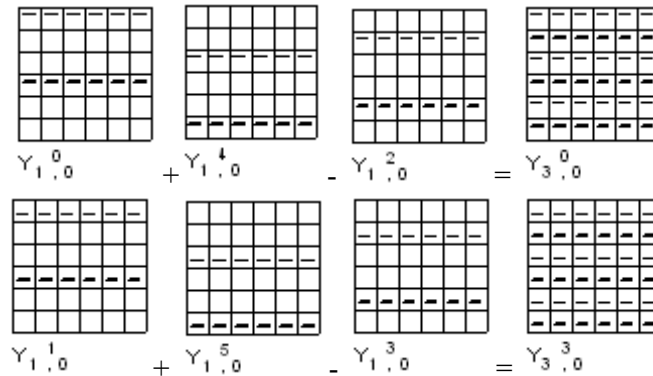


Fig. 5.3: Example for derived redundancy for $N = 12$

The combination of MRT coefficients with even phase index will yield an MRT coefficient with even phase, as seen in the 1st row of the table 5.4. Similarly, as seen in the 2nd row of table 5.4, the combination of MRT coefficients with odd phase index will yield an MRT coefficient with an odd phase index. In the third row of table 5.4, $\sum \pm Y_{2,0}^{even}$ yield $Y_{6,0}^0$. Hence the combination of MRT coefficients with k_1 & k_2 even, will result in an MRT coefficient with k_1 & k_2 even. This can be noticed for different combinations of (k_1, k_2, p) , as in table 5.5. These features can be seen for $N = 20, 24, 28$ etc. where $((N))_4 = 0$ and N not a power of 2. Thus the combination of MRT coefficients, as described above, will result in an MRT coefficient with the same index pattern in (k_1, k_2, p) as in the combination.

Table 5.4: Example of index relation in derived redundancy - $N = 12$

Combination of MRT coefficients	Resulting MRT
$Y_{1,0}^0 + Y_{1,0}^4 - Y_{1,0}^2$	$Y_{3,0}^0$
$Y_{1,0}^1 + Y_{1,0}^5 - Y_{1,0}^3$	$Y_{3,0}^3$
$Y_{2,0}^0 + Y_{2,0}^2 - Y_{2,0}^4$	$Y_{6,0}^0$

Table 5.5: Index relation in derived redundancy - $N = 12$

Combination of MRT coefficients		Resulting MRT	
$dm = 1$	$\sum \pm Y_{odd,even}^{even}$	$dm = 3$	$\sum \pm Y_{odd,even}^{even}$
	$\sum \pm Y_{odd,even}^{odd}$		$\sum \pm Y_{odd,even}^{odd}$
	$\sum \pm Y_{even,odd}^{even}$		$\sum \pm Y_{even,odd}^{even}$
	$\sum \pm Y_{even,odd}^{odd}$		$\sum \pm Y_{even,odd}^{odd}$
	$\sum \pm Y_{odd,odd}^{even}$		$\sum \pm Y_{odd,odd}^{even}$
	$\sum \pm Y_{odd,odd}^{odd}$		$\sum \pm Y_{odd,odd}^{odd}$
$dm = 2$	$\sum \pm Y_{even,even}^{even}$	$dm = M$	$\sum \pm Y_{even,even}^{even}$

On analysis of the visual representation for $N = 12$, there are 4 basic DFT coefficients for $\gcd(k_1, k_2, M) = M = 6$ with one MRT coefficient each, 6 basic DFT coefficients for $\gcd(k_1, k_2, M) = 3$ with two MRT coefficient each, 16 basic DFT coefficients for $\gcd(k_1, k_2, M) = 2$ with three MRT coefficients each and 24 basic DFT coefficients for $\gcd(k_1, k_2, M) = 1$ with six MRT coefficients each respectively as shown in table 5.6. Thus there are a total of 208 MRT coefficients for the 50 basic DFT coefficients for $N = 12$. One MRT coefficient each can be eliminated from the basic DFT coefficients having $\gcd(k_1, k_2, M) = 2$ since the same can be derived from the other MRT coefficients of the respective basic DFT coefficient and the corresponding MRT coefficients having $\gcd(k_1, k_2, M) = M$. Similarly the redundancy in the basic DFT coefficients with $\gcd(k_1, k_2, M) = 1$ in which two MRT coefficients can be eliminated as the same can be derived from the other MRT coefficients and the corresponding MRT coefficient having $\gcd(k_1, k_2, M) = 3$. Since there are 16

and 24 basic DFT coefficients with $\gcd(k_1, k_2, M) = 2$ and 1 respectively, a total of $16 \times 1 + 24 \times 2 = 64$ MRT coefficients are redundant and can be removed without any loss of information resulting in $208 - 64 = 144 = N^2$ MRT coefficients.

Table 5.6: Derived redundancy for $N = 12$

dm	nb_{dm}	Total no. of MRT coefficients $\frac{M}{dm} \times nb_{dm}$	No. of MRT coefficients that can be eliminated from each	Total coefficients that can be eliminated	No. of UMRT coefficients
1	24	144	2	48	96
2	16	48	1	16	32
3	6	12	0	0	12
$M=6$	4	4	0	0	4
Total	50	208		64	144

5.1.3 Analysis for $N = 18$ where $((N))_4 = 2$ and $N/2$ not prime

There are 68 basic DFT coefficients for $N = 18$, of which 4, 16 and 48 correspond to $\gcd(k_1, k_2, M) = M, 3$ and 1 respectively, with a total of 484, (i.e., $4 \times 1 + 16 \times 3 + 48 \times 9$), MRT coefficients. In fig. 5.4, the combination of visual representation of $Y_{1,0}^0, Y_{1,0}^6$ and the sign reversed $Y_{1,0}^3$ will result in $Y_{3,0}^0$. Similar combinations can be observed as illustrated in table 5.7. Thus there is derived redundancy in MRT coefficients for $N = 18$ and they can be eliminated.

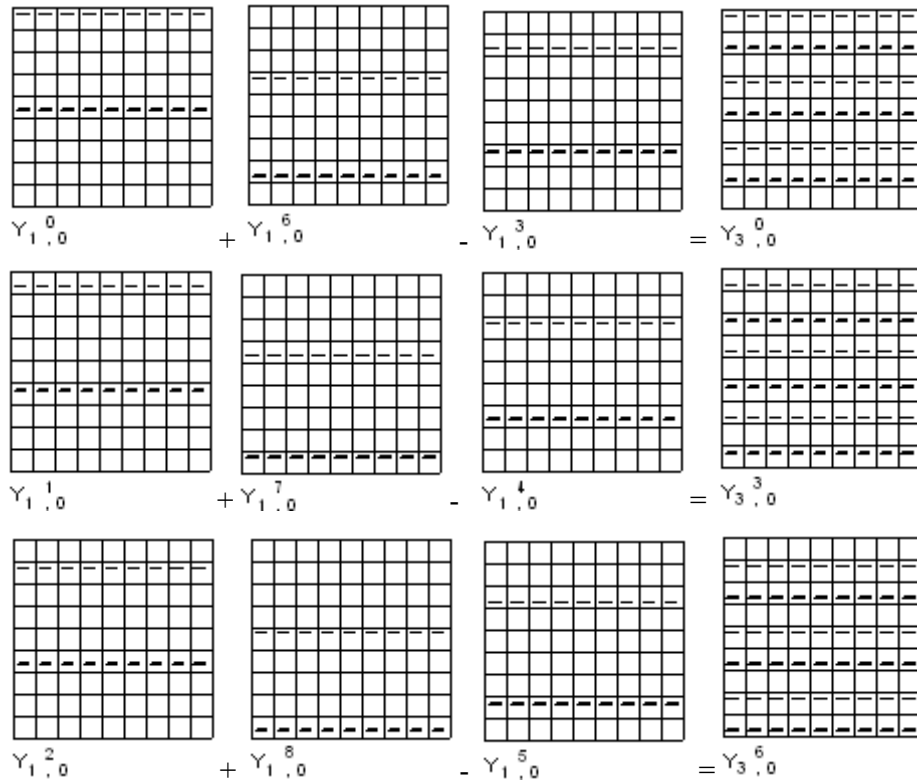


Fig. 5.4: Example for derived redundancy for $N = 18$

Table 5.7: Example of index relation in derived redundancy - $N = 18$

Combination of MRT coefficients	Resulting MRT
$Y_{1,0}^0 + Y_{1,0}^6 - Y_{1,0}^3$	$Y_{3,0}^0$
$Y_{1,0}^1 + Y_{1,0}^7 - Y_{1,0}^4$	$Y_{3,0}^3$
$Y_{1,0}^2 + Y_{1,0}^8 - Y_{1,0}^5$	$Y_{3,0}^6$
$Y_{6,0}^0 + Y_{6,0}^6 - Y_{6,0}^3$	$Y_{0,0}^0$

In table 5.7, the combination of MRT coefficients when k_1 & k_2 are odd & even respectively, will result in an MRT coefficient where k_1 & k_2 are odd & even respectively. Similarly the visual representation of MRT coefficients, having k_1 & k_2 even, are combined, the frequency indices of the resulting MRT coefficient will also be even, as in the 4th row of table 5.7. This can be noticed for different combinations of k_1 and k_2 . Similar feature is seen for $N = 30, 42$ etc. The similarity in the pattern of phase index as noticed in $((N))_4 = 0$ and N not a power of 2 is not seen when $N = 18$. Table 5.8 shows the pattern of frequency index, for different values of $\gcd(k_1, k_2, M) = dm$, when $N = 18$.

As in the first three rows of table 5.7, one MRT coefficient each can be eliminated, without loss of information, since the same can be derived from the other coefficients. Three MRT coefficients can thus be eliminated. The MRT coefficients of all the basic DFT coefficients with $\gcd(k_1, k_2, M) = 1$ can be combined, as shown in Table 5.8, to obtain the corresponding MRT coefficients having $\gcd(k_1, k_2, M) = 3$. So a total of $48 \times 3 = 144$ MRT coefficients can thus be eliminated.

Table 5.8: Index relation in derived redundancy - $N = 18$

Combination of MRT coefficients		Resulting MRT	
$dm = 1$	$\sum \pm Y_{odd, even}$ $\sum \pm Y_{even, odd}$ $\sum \pm Y_{odd, odd}$ $\sum \pm Y_{even, even}$	$dm = 3$	$Y_{odd, even}$ $Y_{even, odd}$ $Y_{odd, odd}$ $Y_{even, even}$
$dm = 3$	$\sum \pm Y_{even, even}$ $\sum \pm Y_{even, odd}$ $\sum \pm Y_{odd, even}$ $\sum \pm Y_{odd, odd}$	$dm = M$	$Y_{0,0}^0$ $Y_{0,M}^0$ $Y_{M,0}^0$ $Y_{M,M}^0$

Similarly the combination of the visual representation of $Y_{6,0}^0, Y_{6,0}^6$ and the sign reversed $Y_{6,0}^3$ will yield $Y_{0,0}^0$ as in the 4th row of table 5.7. Thus the combination of the MRT coefficients where $\gcd(k_1, k_2, M) = 3$ will result in the MRT coefficient where $\gcd(k_1, k_2, M) = M$. One MRT coefficient can be eliminated from the basic DFT coefficients with $\gcd(k_1, k_2, M) = 3$ and there are a

total of 16 such coefficients. Hence the number of MRT coefficients that can be eliminated from the basic DFT coefficients are 144 and 16 corresponding to $\gcd(k_1, k_2, M) = 1$ and 3 respectively. Thus only N^2 unique MRT coefficients are sufficient to represent the signal, as in table 5.9. The other MRT coefficients of the basic DFT coefficients can be reconstructed from the MRT coefficients, which are retained.

Table 5.9: Derived redundancy when $N = 18$

dm	nb_{dm}	Total no. of MRT Coefficients $\frac{M}{dm} \times nb_{dm}$	No. of MRT coefficients that can be eliminated from each	Total coefficients that can be eliminated	No. of UMRT coefficients
1	48	432	3	144	288
3	16	48	1	16	32
$M=9$	4	4	0	0	4
Total	68	484		160	324

5.1.4 Analysis of MRT coefficients when N is a power of 2

There are 22 basic DFT coefficients for $N = 8$, of which 4, 6 and 12 basic DFT coefficients correspond to $\gcd(k_1, k_2, M) = 4, 2$ and 1 respectively with a total of 64, (i.e., $4 + 6 \times 2 + 12 \times 4$), MRT coefficients. There are only N^2 MRT coefficients for the basic DFT coefficients which is unique and hence no elimination of MRT coefficients is required in the case of $N = 8$, as in table 5.10. Similar features can be noticed for $N = 16, 32, 64$ etc. Thus there is no derived redundancy in MRT coefficients when N is a power of 2.

Table 5.10: Derived redundancy when $N = 8$

dm	nb_{dm}	Total no. of MRT coefficients $\frac{M}{dm} \times nb_{dm}$	No. of MRT coefficients that can be eliminated from each	Total coefficients that can be eliminated	No. of UMRT coefficients
1	12	48	0	0	48
2	6	12	0	0	12
$M=4$	4	4	0	0	4
Total	22	64		0	64

5.2 Computation of redundant MRT coefficients

Analysis of derived redundancy has been done for different N , from 4 to 4620, selected suitably based on the peculiarity of its divisors and the results are summarized, for $((N))_4 = 0$ & N not a power of 2 and $((N))_4 = 2$ & $N/2$ not prime, in tables 5.11 and 5.12 respectively. In the tables nr is the number of redundant MRT coefficients corresponding to each dm , due to the derived redundancy. When $N/2$ prime, one MRT coefficient from each of the basic DFT coefficients can be eliminated, when $\gcd(k_1, k_2, M) = 1$, so as to obtain UMRT. There is no derived redundancy for N power of 2.

Table 5.11: nr corresponding to each dm for N , where $((N))_4 = 0$ & N not a power of 2.

12	dm	1	2													
	nr	2	1													
20	dm	1	2													
	nr	2	1													
24	dm	1	2	4												
	nr	4	2	1												
28	dm	1	2													
	nr	2	1													
36	dm	1	2	3	6											
	nr	6	3	2	1											
40	dm	1	2	4												
	nr	4	2	1												
44	dm	1	2													
	nr	2	1													
48	dm	1	2	4	8											
	nr	8	4	2	1											
52	dm	1	2													
	nr	2	1													
56	dm	1	2	4												
	nr	4	2	1												
60	dm	1	2	3	5	6	10									
	nr	14	7	2	2	1	1									
72	dm	1	2	3	4	6	12									
	nr	12	6	4	3	2	1									
84	dm	1	2	3	6	7	14									
	nr	18	9	2	1	2	1									
100	dm	1	2	5	10											
	nr	10	5	2	1											
108	dm	1	2	3	6	9	18									
	nr	18	9	6	3	2	1									
120	dm	1	2	3	4	5	6	10	12	20						
	nr	28	14	4	7	4	2	2	1	1						
132	dm	1	2	3	6	11	22									
	nr	26	13	2	1	1	1									
144	dm	1	2	3	4	6	8	12	24							
	nr	24	12	8	6	4	3	2	1							
168	dm	1	2	3	4	6	7	12	14	28						
	nr	36	18	4	9	2	4	1	2	1						
180	dm	1	2	3	5	6	9	10	15	18	30					
	nr	42	21	14	6	7	2	3	2	1	1					
200	dm	1	2	4	5	10	20									
	nr	20	10	5	4	2	1									
216	dm	1	2	3	4	6	9	12	18	36						
	nr	36	18	12	9	6	4	3	2	1						
280	dm	1	2	4	5	7	10	14	20	28						
	nr	44	22	11	4	4	2	2	1	1						
336	dm	1	2	3	4	6	7	8	12	14	24	28	56			
	nr	72	36	8	18	4	8	9	2	4	1	2	1			
360	dm	1	2	3	4	5	6	9	10	12	15	18	20	30	36	60
	nr	84	42	28	21	12	14	4	6	7	4	2	3	2	1	1
400	dm	1	2	4	5	8	10	20	40							
	nr	40	20	10	8	5	4	2	1							
420	dm	1	2	3	5	7	6	10	14	15	21	30	35	42	70	
	nr	114	57	22	18	14	11	9	7	2	2	1	2	1	1	
660	dm	1	2	3	5	6	10	11	15	22	30	33	55	66	110	
	nr	170	85	30	26	15	13	14	2	7	1	2	2	1	1	

780	<i>dm</i>	1	2	3	5	6	10	13	15	26	30	39	65	78	130																
	<i>nr</i>	198	99	34	30	17	15	14	2	7	1	2	2	1	1																
900	<i>dm</i>	1	2	3	5	6	9	10	15	18	25	30	45	50	75	90	150														
	<i>nr</i>	210	105	70	42	35	10	21	14	5	6	7	2	3	2	1	1														
1000	<i>dm</i>	1	2	4	5	10	20	25	50	100																					
	<i>nr</i>	100	50	25	20	10	5	4	2	1																					
1540	<i>dm</i>	1	2	5	7	10	11	14	22	35	55	70	77	110	154																
	<i>nr</i>	290	145	34	30	17	22	15	11	2	2	1	2	1	1																
1800	<i>dm</i>	1	2	3	4	5	6	9	10	12	15	18	20	25	30	36	45	50	60	75	90	100	150	180	300						
	<i>nr</i>	420	210	140	105	84	70	20	42	35	28	10	21	12	14	5	4	6	7	4	2	3	2	1	1						
4620	<i>dm</i>	1	2	3	5	6	7	10	11	14	15	21	22	30	33	35	42	55	66	70	77	105	110	154	165	210	231	330	385	462	770
	<i>nr</i>	1350	675	290	222	145	170	111	114	85	34	30	57	17	22	26	15	18	11	13	14	2	9	7	2	1	2	1	2	1	1

Table 5.12: *nr* corresponding to each *dm* for *N*, where $((N))_4 = 2$ and *N*/2 not prime

<i>N</i>									
18	<i>dm</i>	1	3						
	<i>nr</i>	3	1						
30	<i>dm</i>	1	3	5					
	<i>nr</i>	7	1	1					
42	<i>dm</i>	1	3	7					
	<i>nr</i>	9	1	1					
50	<i>dm</i>	1	5						
	<i>nr</i>	5	1						
54	<i>dm</i>	1	3	9					
	<i>nr</i>	9	3	1					
66	<i>dm</i>	1	3	11					
	<i>nr</i>	13	1	1					
70	<i>dm</i>	1	5	7					
	<i>nr</i>	11	1	1					
78	<i>dm</i>	1	3	13					
	<i>nr</i>	15	1	1					
90	<i>dm</i>	1	3	5	9	15			
	<i>nr</i>	21	7	3	1	1			
98	<i>dm</i>	1	7						
	<i>nr</i>	7	1						
102	<i>dm</i>	1	3	17					
	<i>nr</i>	19	1	1					
210	<i>dm</i>	1	3	5	7	15	21	35	
	<i>nr</i>	57	11	9	7	1	1	1	
250	<i>dm</i>	1	5	25					
	<i>nr</i>	25	5	1					
450	<i>dm</i>	1	3	5	9	15	25	45	75
	<i>nr</i>	105	35	21	5	7	3	1	1

The total number of redundant MRT coefficients can be obtained from the table 5.11 and 5.12 using the formula $\sum_{dm_e} nb_{dm_e} .nr$ where $dm_e = \gcd(k_1, k_2, M)$ having redundant MRT coefficients.

Table 5.13 shows the total number of MRT coefficients for the entire basic DFT coefficients, the number of MRT coefficients that can be derived and the percentage reduction for $((N))_4 = 0$ & *N* not a power of 2, $((N))_4 = 2$ & *N*/2 not prime and *N*/2 prime. In the table, for *N* = 630, 1050, 1470, 1890, 2310, 3990 and 4620, the percentage reduction is more than fifty percentage since the number of *dm* is more. Derived redundancy is only $2N + 4$ for *N*/2 prime and hence the percentage

of reduction is low. Thus it can be inferred that in general, derived redundancy is high when the number of dm is more. However there is no derived redundancy for N power of 2, the reason for which is explained below.

Table 5.13: Reduction in computation due to derived redundancy

$((N))_i=0$ & N not a power of 2				$((N))_i=0$ & $N/2$ not prime				$N/2$ prime			
N	$nmrt_{nb}$	nr	% reduction	N	$nmrt_{nb}$	nr	% reduction	N	$nmrt_{nb}$ (N^2+2N+4)	nr $(2N+4)$	% reduction
56	3648	512	14.03	18	484	160	33.06	6	52	16	30.77
60	6448	2848	44.17	30	1612	712	44.17	34	1228	72	5.86
72	7744	2560	33.06	42	2964	1200	40.49	62	3972	128	3.22
84	11856	4800	40.49	50	3124	624	19.97	106	11452	216	1.89
96	13312	4096	30.77	54	4372	1456	33.30	202	41212	408	0.99
100	12496	2496	19.97	66	6916	2560	37.02	298	89404	600	0.67
108	17488	5824	33.30	70	7068	2168	30.67	394	156028	792	0.51
120	25792	11392	44.17	78	9516	3432	36.07	502	253012	1008	0.39
132	27664	10240	37.02	90	15004	6904	46.01	586	344572	1176	0.34
144	30976	10240	33.06	98	11204	1600	14.28	718	516964	1440	0.28
168	47424	19200	40.49	102	15964	5560	34.83	818	670764	1640	0.24
200	49984	9984	19.97	210	367536	191136	52.00	922	851932	1848	0.22
216	69952	23296	33.30	250	78124	15624	19.99	1018	1038364	2040	0.20
280	113088	34688	30.67	450	378004	175504	46.43	1142	1306452	2288	0.18
336	189896	76800	40.44	494	278892	34856	12.49	1502	2259012	3008	0.13
360	240064	11464	47.76	630	855228	458328	53.59	1574	2480628	3158	0.13
400	199956	39936	19.97	858	1265628	529464	41.83	1642	2699452	3288	0.12
660	857584	421984	49.21	1050	2314884	1212384	52.37	1706	2913852	3416	0.12
780	1179984	571584	48.44	1470	4515212	2354312	52.14	1774	3150628	3552	0.11
900	1512016	702016	46.43	1890	7725324	4153224	53.76	1814	3294228	3632	0.11
1000	1249984	249984	19.99	2310	12220572	6884472	56.34	1858	3455884	3720	0.11
1540	3760176	1388576	36.93	3718	16460612	2637088	16.02	1906	3636652	3816	0.10
1800	6048064	2808064	46.43	3990	35007804	19087704	54.52	1994	3980028	3992	0.10
4620	48882288	27537888	56.34	4598	24544020	3402416	13.86	4622	21372132	9248	0.04

In the first row of table 5.11, for $N = 12$, the derived redundancy is noticed in the basic DFT coefficients where $dm = 1$ and 2 whereas derived Redundancy is not present when $dm = 6$ and 3, i.e., when $dm = M$ and $M/2$. Redundant MRT coefficients are not present when $dm = 10$ and 5, for $N = 20$, i.e., when $dm = M$ and $M/2$, as seen in the 2nd row of table 5.11. Similar features are noticed for all N listed in table 5.11 and 5.12. It can be inferred that the derived redundancy in MRT coefficients is present in all the basic DFT coefficients except when $\gcd(k_1, k_2, M)$ falls in the series $M, M/2, M/2^2, \dots, M/2^{n-1}$ where, $2^n \parallel N$. For N power of 2, all dm falls in the above series and hence there is no derived redundancy. Redundancy present can be determined as follows:

$$\text{Number of MRT coefficients in a basic DFT coefficient, } n_p = M / \gcd(k_1, k_2, M) = M/dm \quad (5.1)$$

$$(\text{since } \gcd(k_1, k_2, M) = dm)$$

Table 5.14 shows the number of basic DFT coefficients (nb_{dm}) and the number of UMRT coefficients (nu) corresponding to each dm for $N = 12, 20, 24$ and 30. From the analysis of the table, the number of UMRT coefficient in a basic DFT coefficient is given by,

$$nu = \varphi\left(\frac{N}{\gcd(k_1, k_2, M)}\right) = \varphi\left(\frac{N}{dm}\right) \tag{5.2}$$

Then from (5.1) and (5.2), the number of redundant MRT coefficients in a basic DFT coefficient is given by

$$nr = M/dm - \varphi\left(\frac{N}{dm}\right). \tag{5.3}$$

From the analysis of table 5.11 & 5.12 and by trial and error, the total number of MRT coefficients that are redundant in all the basic DFT coefficients when $\gcd(k_1, k_2, M) = dm$ is given by $2^\alpha ((-1)^{n+1} \prod_{i=1}^n (p_i^{\beta_i-1} - p_i^{\beta_i}) + \prod_{i=1}^n p_i^{\beta_i})$ where p_i is the odd prime divisors of M/dm , n is the number of odd prime divisors of N , α is the power of 2 in the prime factorization of M/dm and β_i is the power of odd prime divisors p_i in the prime factorization of M/dm .

From the table 5.14, it can be seen that the total number of UMRT coefficients is N^2 . This can be extended to any even value of N . From (5.2), total number of UMRT coefficients = $\sum_{dm} nb_{dm} \cdot \varphi\left(\frac{N}{dm}\right) = N^2$.

Table 5.14: nb_{dm} and nu for different N

N								Total UMRT coefficients
12	dm	1	2	3	6			144
	nb_{dm}	24	16	6	4			
	nu	4	2	2	1			
20	dm	1	2	5	10			400
	nb_{dm}	36	24	6	4			
	nu	8	4	2	1			
24	dm	1	2	3	4	6	12	576
	nb_{dm}	48	24	12	16	6	4	
	nu	8	4	4	2	2	1	
30	dm	1	3	5	15			900
	nb_{dm}	96	24	16	4			
	nu	8	4	2	1			

5.3 MRT coefficients in basic DFT coefficients for N , power of 2

Let $N = 2^n$, then $dm(i) = 2^0, 2^1, 2^2, \dots, 2^{n-1}$ are the n divisors of M .

Total no. of basic DFT coefficients where $\gcd(k_1, k_2, M) = M$ is given by, $nb_M = 4$

Total no. of Y_{k_1, k_2}^p where $\gcd(k_1, k_2, M) = M$ is given by, $nmrt_M = 4$

From table 5.15, total no. of basic DFT coefficients where $\gcd(k_1, k_2, M) = dm$ (except for $dm = M$),

$$nb_{dm} = 3.M/dm \tag{5.4}$$

From (5.1), no. of Y_{k_1, k_2}^p for a basic DFT coefficient where $\gcd(k_1, k_2, M)$ is dm ,

$$n_p = M/dm \tag{5.5}$$

From (5.4 & 5.5), total no. of Y_{k_1, k_2}^p corresponding to the basic DFT coefficients where $\gcd(k_1, k_2, M)$ is dm (except for $dm = M$),

$$nmrt_{dm} = 3.M^2/dm^2 \quad (5.6)$$

$$\begin{aligned} \text{Total no. of } Y_{k_1, k_2}^p &= 4 + \sum_{i=0}^{n-2} 3.M^2 / dm(i)^2 = 4 + \sum_{i=0}^{n-2} 3.dm(n-1-i)^2 \\ &(\because dm(i) = M / dm(n-1-i)) \end{aligned}$$

E.g. when $N = 8 = 2^3$, $dm(i) = 1, 2$, and 4 .

$$\text{Total no. of } Y_{k_1, k_2}^p = 4 + \sum_{i=0}^{3-2} 3.dm(3-1-i)^2 = 4 + 3.4^2 + 3.2^2 = 64$$

So in general when N is a power of 2, the MRT coefficients of the basic DFT coefficients are unique and form the UMRT.

Table 5.15: nb for each dm when N power of 2

dm	nb_{dm}						Total ($3N-2$)
	1	2	4	8	16	32	
N							
4	6	4	-	-	-	-	10
8	12	6	4	-	-	-	22
16	24	12	6	4	-	-	46
32	48	24	12	6	4	-	94
64	96	48	24	12	6	4	190

5.4 Selection of UMRT coefficients

Section 5.2 shows that there is no derived redundancy in MRT coefficients when $\gcd(k_1, k_2, M)$ falls in the series $M, M/2, M/2^2, \dots, M/2^{n-1}$ where, $2^n \parallel N$. When N is a power of 2, all the coefficients fall within the above series i.e., $\gcd(k_1, k_2, M)$ of the basic DFT coefficients is either $M, M/2, M/2^2, \dots$ or $M/2^{n-1}$. It is also verified that, when N is a power of 2, for every dm , $M/dm = \varphi(N/dm)$. So from (5.3), there is no derived redundancy when N is a power of 2.

It was also seen in section 5.1 and 5.2 that the total number of UMRT coefficients is N^2 for any even N . But the total number of MRT coefficients of the basic DFT coefficients is more than that due to the derived redundancy present, except when N is a power of 2. We have to select the MRT coefficients when $N/2$ prime, $((N))_4 = 2$ & $N/2$ not prime, and $((N))_4 = 0$ and N not a power of 2 by discarding redundancy. When N is not a power of 2, there are MRT coefficients whose $\gcd(k_1, k_2, M)$ does not fall in the above series i.e., when $M/dm \neq \varphi(N/dm)$. The MRT coefficients of such basic DFT coefficients have to be selected so as to eliminate the redundant one. For each basic DFT coefficient, depending on $\gcd(k_1, k_2, M)$, the number of redundant MRT coefficients can be found, as in section 5.2. These redundant MRT coefficients can be removed in three ways: 1) from the

beginning 2) from the middle or 3) from the end. If there are ‘ nr ’ redundant MRT coefficients, then either the first ‘ nr ’ MRT coefficients, or the middle most ‘ nr ’ coefficients or the last ‘ nr ’ coefficients can be removed. If the first nr MRT coefficients are to be removed where $\text{gcd}(k_1, k_2, M) = dm$, then $p = 0, dm, 2.dm, \dots (nr-1)dm$ are removed retaining the remaining MRT coefficients, i.e., $p = nr.dm, (nr + 1)dm, (nr + 2)dm, \dots, M/dm - 1$. E.g., when $N = 24$ and $(k_1, k_2) = (2, 0)$ where $dm = 2$, then $p = 0$ and 2 can be removed retaining $p = 4, 6, 8$ and 10.

5.5 Placement/Matrix representation of UMRT coefficients

In [167], the positional details of 8×8 MRT matrix has been shown, which places the MRT coefficients corresponding to $k_2 = 0, M, 1$ and 2 in that order from left to right in a 8×8 matrix. The above scheme even though places the MRT coefficients corresponding to a (k_1, k_2) in a row, a generalized placement is not possible. Here a new placement method of UMRT coefficients for an $N \times N$ matrix is proposed.

The number of UMRT coefficients is N^2 . When N is a power of 2, the MRT coefficients of basic DFT coefficients form the UMRT. E.g. when $N = 8$, the MRT coefficients corresponding to the following (k_1, k_2) are unique: $(0, 0), (1, 0), (2, 0), (4, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (6, 2), (0, 4), (1, 4), (2, 4)$ and $(4, 4)$. In the placement scheme, the nonnegative integers (kcp_n) co-prime to N/dm and less than N/dm are computed, where $dm = \text{gcd}(k_1, k_2, M)$ is the divisor of M . Then the MRT coefficients corresponding to (k_1, k_2) are placed in $((k_1.kcp_n)_N, ((k_2.kcp_n)_N)$. E.g. when $(k_1, k_2) = (0, 0)$, $\text{gcd}(k_1, k_2, M) = 4$ and $kcp_1 = 1$. Hence $p = 0$ corresponding to $(0, 0)$ is placed at $(0, 0)$. When $(k_1, k_2) = (1, 0)$, $\text{gcd}(k_1, k_2, M) = 1$ and $kcp_n = 1, 3, 5$ and 7. Hence $p = 0, 1, 2,$ and 3 are placed at location $(1, 0), (3, 0), (5, 0),$ and $(7, 0)$ respectively. When $(k_1, k_2) = (2, 0)$, $\text{gcd}(k_1, k_2, M) = 2$ and $kcp_n = 1,$ and 3. Hence $p = 0,$ and 2 are placed respectively at location $(2, 0),$ and $(6, 0)$. Placement of all the MRT coefficients for $N = 8$ is shown in fig. 5.5 in which entries ‘102’ indicates that the coefficient with index $(k_1, k_2, p) = (1, 0, 2)$ is placed in that position of the 2-D array.

000	010	020	011	040	012	022	013
100	110	120	311	140	512	321	713
200	210	220	611	240	212	622	613
101	310	320	111	141	712	121	513
400	410	420	411	440	412	422	413
102	510	122	711	142	112	323	313
202	610	620	211	242	612	222	213
103	710	322	511	143	312	123	113

Fig. 5.5: Placement details of 8×8 MRT

The above scheme of placement can be extended to any even N . As a special case when N is a power of 2, the number of nonnegative integers co-prime to N/dm and less than N/dm , (i.e., $\phi(N/dm)$, where ϕ is the Euler Totient function) is M/dm . Therefore the nonnegative integers which are co-prime to N/dm and less than N/dm are the odd numbers from 1 to N/dm .

Number of unique MRT coefficients present in a basic DFT coefficient is equal to the redundancy factor of the corresponding basic DFT coefficient. The UMRT coefficients of a basic DFT coefficient are placed in the positions corresponding to the redundant DFT coefficients. Algorithm for placement of one of the MRT coefficient $Y(k_1, k_2, p)$ is illustrated below. The placement scheme satisfies the linearity property. But it does not satisfy other matrix properties such as transpose etc.

5.5.1 Algorithm for placement of a UMRT coefficient

1. Given k_1, k_2, p and the MRT coefficient $Y(k_1, k_2, p)$
2. Compute $dm = \text{gcd}(k_1, k_2, M)$
3. Compute $\varphi(N/dm)$ and the nonnegative integers co-prime to N/dm and less than N/dm defined as $\text{coprime.Nbydm}(r)$
4. $kcp = \text{coprime.Nbydm}(p/dm)$
5. $Y(((k_1.kcp))_N, ((k_2.kcp))_N) = Y(k_1, k_2, p)$

5.6 Development of algorithms for the computation of 2-D UMRT

In 2-D UMRT only unique MRT coefficients are present. Unique MRT coefficients are the Y_{k_1, k_2}^p of the basic DFT coefficients after removing those Y_{k_1, k_2}^p which can be derived. Hence UMRT can be computed using any of three methods described below.

5.6.1 Three layer M spacing method

The four layer M spacing method, for the computation of 2-D DFT described in section 4.2.5.4, can be suitably modified for computing UMRT. The computations in layer 4 are fully avoided. There is no need to compute the $N/2$ twiddle factors as required for the 2-D DFT computation. In layer 3, only unique MRT coefficients need be computed.

The number of MRT coefficients to be eliminated for each dm has to be pre-computed using (5.3). These MRT coefficients can be derived from the unique MRT coefficients and hence need not be computed. For each divisor ' dm ' of M , $\varphi(N/dm)$ and the co-prime integers of every N/dm and upto N/dm has to be pre-computed, as discussed in section 5.5, for placement of the coefficients. Table 5.16 shows the number of UMRT coefficients required to be computed in each group C, D, E and F for different N . In the table Tnu is the total number of UMRT coefficients corresponding to all the basic DFT coefficients where $\text{gcd}(k_1, k_2, M) = dm$. It can be seen that the number of UMRT coefficients corresponding to C, D, E and F are same. Hence the number of UMRT coefficients is equally distributed between C, D, E and F. This inference is helpful in developing scalable architecture. But the number of computations in each group is not same for

$((N))_4 = 0$. This is due to the uneven distribution of MRT coefficients with the same dm . The number of computation is evenly distributed between the four groups when $((N))_4 = 2$, since the MRT coefficients with the same dm is equally distributed between the four groups, as in table 5.16.

Table 5.16: nu to be computed by each group for different N

N		C (k_1 =even, k_2 =even)				D (k_1 =odd, k_2 =even)				E (k_1 =even, k_2 =odd)				F (k_1 =odd, k_2 =odd)											
4	dm	2				Total	1				Total	1				Total	1				Total	1			
	<i>nb</i>	4				4	2				2	2				2	2				2	2			
	<i>Tnu</i>	4				4	4				4	4				4	4				4	4			
6	dm	3	1			3	1			3	1			3	1			3	1			3	1		
	<i>nb</i>	1	4			5	1	4		5	1	4		5	1	4		5	1	4		5	1	4	
	<i>Tnu</i>	1	8			9	1	8		9	1	8		9	1	8		9	1	8		9	1	8	
8	dm	4	2			1				1				1				1				1			
	<i>nb</i>	4	6			10	4			4	4			4	4			4	4			4	4		
	<i>Tnu</i>	4	12			16	16			16	16			16	16			16	16			16	16		
10	dm	5	1			5	1			5	1			5	1			5	1			5	1		
	<i>nb</i>	1	6			7	1	6		7	1	6		7	1	6		7	1	6		7	1	6	
	<i>Tnu</i>	1	24			25	1	24		25	1	24		25	1	24		25	1	24		25	1	24	
12	dm	6	2			1	3			1	3			1	3			1	3			1	3		
	<i>nb</i>	4	16			20	8	2		10	8	2		10	8	2		10	8	2		10	8	2	
	<i>Tnu</i>	4	32			36	32	4		36	32	4		36	32	4		36	32	4		36	32	4	
14	dm	7	1			7	1			7	1			7	1			7	1			7	1		
	<i>nb</i>	1	8			9	1	8		9	1	8		9	1	8		9	1	8		9	1	8	
	<i>Tnu</i>	1	48			49	1	48		49	1	48		49	1	48		49	1	48		49	1	48	
16	dm	8	4	2		1				1				1				1				1			
	<i>nb</i>	4	6	12		22	8			8	8			8	8			8	8			8	8		
	<i>Tnu</i>	4	12	48		64	64			64	64			64	64			64	64			64	64		
18	dm	9	1	3		9	1	3		9	1	3		9	1	3		9	1	3		9	1	3	
	<i>nb</i>	1	12	4		17	1	12	4	17	1	12	4	17	1	12	4	17	1	12	4	17	1	12	4
	<i>Tnu</i>	1	72	8		81	1	72	8	81	1	72	8	81	1	72	8	81	1	72	8	81	1	72	8
20	dm	10	2			5	1			5	1			5	1			5	1			5	1		
	<i>nb</i>	4	24			28	2	12		14	2	12		14	2	12		14	2	12		14	2	12	
	<i>Tnu</i>	4	96			100	4	96		100	4	96		100	4	96		100	4	96		100	4	96	
22	dm	11	1			11	1			11	1			11	1			11	1			11	1		
	<i>nb</i>	1	12			13	1	12		13	1	12		13	1	12		13	1	12		13	1	12	
	<i>Tnu</i>	1	120			121	1	120		121	1	120		121	1	120		121	1	120		121	1	120	
24	dm	12	6	4	2	3	1			3	1			3	1			3	1			3	1		
	<i>nb</i>	4	6	16	24	50	4	16		20	4	16		20	4	16		20	4	16		20	4	16	
	<i>Tnu</i>	4	12	32	96	144	16	128		144	16	128		144	16	128		144	16	128		144	16	128	
28	dm	14	2			7	1			7	1			7	1			7	1			7	1		
	<i>nb</i>	4	32			36	2	16		18	2	16		18	2	16		18	2	16		18	2	16	
	<i>Tnu</i>	4	192			196	4	192		196	4	192		196	4	192		196	4	192		196	4	192	
30	dm	15	5	3	1	15	5	3	1	15	5	3	1	15	5	3	1	15	5	3	1	15	5	3	1
	<i>nb</i>	1	4	6	24	35	1	4	6	24	35	1	4	6	24	35	1	4	6	24	35	1	4	6	24
	<i>Tnu</i>	1	8	24	192	225	1	8	24	192	225	1	8	24	192	225	1	8	24	192	225	1	8	24	192
32	dm	16	8	4	2	1				1				1				1				1			
	<i>nb</i>	4	6	12	24	46	16			16	16			16	16			16	16			16	16		
	<i>Tnu</i>	4	12	48	192	256	256			256	256			256	256			256	256			256	256		

The fig. 5.6 shows the M spacing based parallel distributed architecture for the computation of $N \times N$ UMRT. The important steps of M spacing algorithm for UMRT computation are as follows.

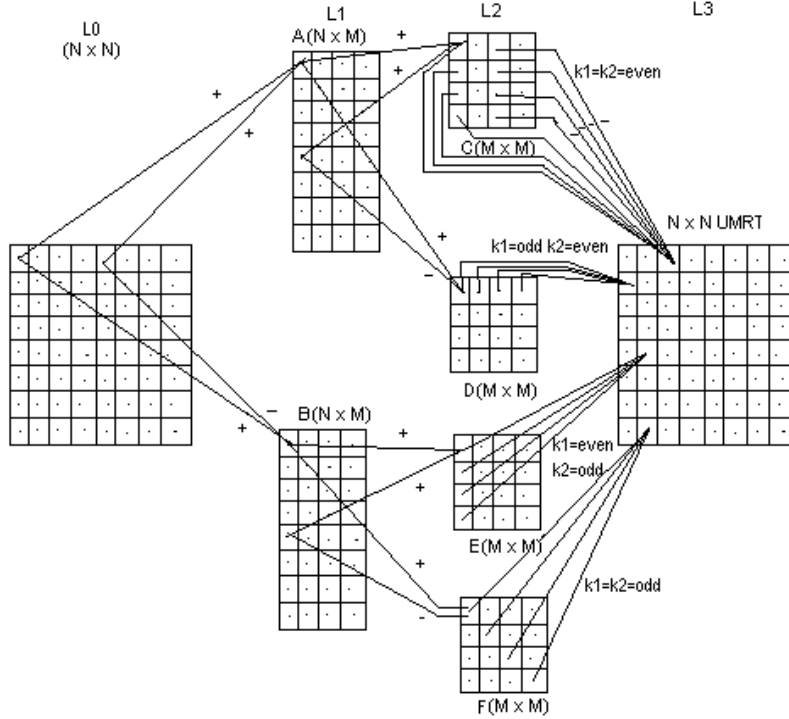


Fig. 5.6: M spacing parallel distributed architecture for $N \times N$ UMRT

5.6.1.1 Algorithm

1. Compute divisors of $M = dm(i)$, number of divisors of $M = no_of_divisors_of_M$, indices (k_1, k_2) of all the basic DFT coefficients and the no. of basic DFT = $no_of_basicDFT$ as in section 3.2.5 and 3.2.6.
2. For each divisor of M , $dm(i)$ calculated in step 1, compute $\varphi(N/dm(i))$, the co-prime integers of every $N/dm(i)$ up to $N/dm(i)$ and the number of MRT coefficients to be eliminated corresponding to each dm .

For $i = 1$ to $no_of_divisors_of_M$

$Nbydm = N/dm(i)$, $r = dm(i)$, $s = 1$

$phy_ofNbydm(r) = 1$, $coprime(r, s) = 1$, $s = 2$

For $t = 2$ to $Nbydm$

If $\gcd(t, Nbydm) = 1$

$coprime(r, s) = t$, $s = s + 1$, $phy_ofNbydm(r) = phy_ofNbydm(r) + 1$

$MRT_eliminate(r) = M/r - phy_ofNbydm(r)$ --number of redundant MRT coefficients for each dm are computed and stored

3. Algorithm for computing UMRT coefficients

Layer 1 & 2

Layer 1 & 2 are same as in section 4.2.3.

Layer 3

(Unique MRT coefficients corresponding to every basic DFT coefficients are computed in layer 3)

For $q = 1$ to $no_of_basicDFT$

Compute $dm = \gcd(k1(q), k2(q), M)$, $h = \gcd(k1(q), M)$, $v = \gcd(k2(q), M)$, $z = \gcd(k2(q), N)$

If $((k1(q)))_2 = 0$ ---depending on the nature of (k_1, k_2) C, D, E or F block is selected for computation

if $((k2(q)))_2 = 0$

U = C

else U = E

elseif $((k2(q)))_2 = 0$

U = D

else U = F

For $p = 0$ to $M - 1 - MRT_eliminate(dm).dm$ in steps of dm ---redundant MRT coefficients are removed from the last

$kcp = coprime(dm, p/dm + 1)$

$fk_1 = ((kcp.k1(q)))_N$, $fk_2 = ((kcp.k2(q)))_N$ --- placement of coefficients are computed here

compute particular solution $(n1, n2)$ using modified trial and error algorithm as in section 4.2.5.3.4

For $r = 0$ to dm

For $s = 0$ to dm

For $t = 0$ to $M/dm - 1$

$next_n1 = ((n1 + r.v/dm + k2(q).t))_M$ ---computation for index of next element of U

If $((M))_{k2(q)} = 0$

$next_n2 = ((n2+r((M - k1(q)))_{M/dm} + s.M/v + (M - k1(q).t))_M$ --index when $k_2|M$

else

$next_n2 = ((n2+((r[N - z].k1(q)/[2.z.dm]))_{M/dm} + s.M/v + [M - k1(q).t])_M$ --index when $k_2 \nmid M$

If $((next_n1.k1(q) + next_n2.k2(q)))_N \geq M$ ---testing for element to be added or subtracted

$next_term = -U(next_n1, next_n2)$

else

$next_term = U(next_n1, next_n2)$

$Y(fk_1, fk_2) = Y(fk_1, fk_2) + next_term$

5.6.2 Visual method

Visual method, as discussed in section 3.3, can be suitably modified to compute the UMRT coefficients. The modifications are similar to that done in the M spacing method. In the first step compute the divisors of M , number of divisors of M , indices of all the basic DFT coefficients and the number of basic DFT. In the second step compute the number of MRT coefficients to be eliminated for each dm , $\varphi(N/dm)$, and the co-prime integers of every N/dm up to N/dm . In the final step compute the unique MRT coefficients using the visual method.

5.6.3 Modified direct method for UMRT computation

UMRT coefficients are computed using the modified DFT method using (1.6). The MRT coefficients that can be derived are not computed using the elimination algorithm discussed in the other two methods. The placement of unique MRT coefficients is also done using the algorithm developed in section 5.5.1. The important steps of the algorithm are as follows.

1. Steps 1 and 2 are same as that of the algorithm in section 5.6.1.1
2. UMRT coefficients corresponding to every basic DFT coefficients are computed next

For $q = 1$ to $no_of_basicDFT$

 Compute $dm = \gcd(k1(q), k2(q), M)$

 For $p = 0$ to $M - 1 - MRT_eliminate(dm).dm$ in steps of dm ---redundant MRT coefficients are removed from the last

$kcp = coprime(dm, p/dm + 1)$

$fk_1 = ((kcp.k1(q)))_N, fk_2 = ((kcp.k2(q)))_N$ --- placement of coefficients are computed here

 For $n1 = 0$ to $N - 1$

 For $n2 = 0$ to $N - 1$

$z = ((n1.k1 + n2.k2))_N$

 If $z = p$

$Y(fk_1, fk_2) = Y(fk_1, fk_2) + A(n1, n2)$

 elseif $z = p + M$

$Y(fk_1, fk_2) = Y(fk_1, fk_2) - A(n1, n2)$

5.7 Conclusion

The 2-D UMRT coefficients are unique and fit in the $N \times N$ memory space, for any even N . The 2-D UMRT maps a 2-D array of real data in time/spatial domain into another 2-D array of real data in frequency domain using real additions only. Different algorithms are designed and developed for the computation of 2-D UMRT. The proposed placement scheme satisfies linearity property, but does not satisfy other matrix properties such as transpose.

CHAPTER 6

IMPLEMENTATION OF PARALLEL DISTRIBUTED ARCHITECTURE FOR THE COMPUTATION OF 2-D DFT & UMRT

The algorithms developed in chapters 3, 4, and 5 are simulated in Matlab®. The algorithm developed in software provides maximum flexibility but lacks performance. To meet the tight throughput constraint of the real time processing, a very high speed, dedicated special purpose hardware processor is required. Nowadays, more complex DSP and image/video processing algorithms are implemented on single chip. Hardware implementations in ASICs were proposed to speed up the algorithm. But this is an inflexible solution and lacks cost efficiency. Another approach suitable for implementation is the use of reconfigurable hardware using FPGA. The performance of FPGA is quickly nearing that of ASICs.

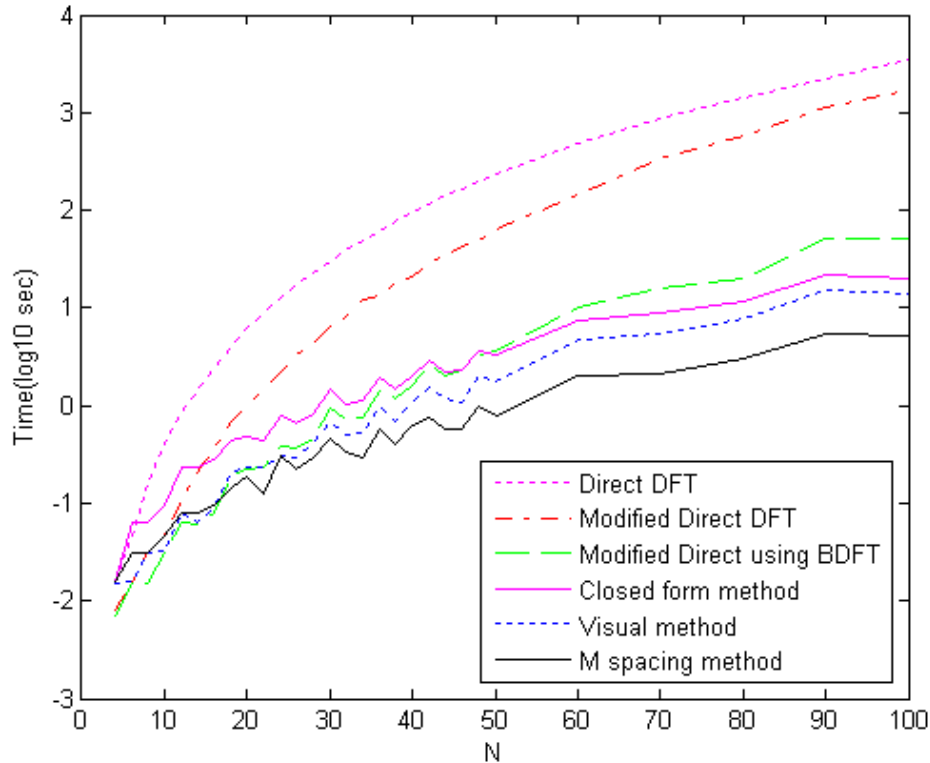
6.1 Matlab simulation

6.1.1 2-D DFT

The 2-D DFT computation, using visual method, M spacing method and modified DFT using basic DFT coefficients, is compared with three other methods viz. a viz., direct DFT, modified DFT and closed form DFT [189]. Direct DFT computation uses the basic 2-D DFT equation to compute the coefficients. Modified DFT computation is used to compute the DFT coefficients using (1.5) in the second method. In another method named as “Modified DFT using basic DFT coefficients”, Y_{k_1, k_2}^p of only basic DFT coefficients are computed using (1.6) and (1.7). Then the entire DFT coefficients are computed by the permutations over p as done in visual method in section 3.3. Visual method is also involved in the comparison. Closed form method [189] is used to compute the MRT of basic DFT coefficients and then the entire DFT coefficients are computed by the permutations over p as above. Six methods described above are simulated using Matlab 7.0 on Intel® Pentium®4 CPU, 1.5 GHZ, machine. The computation time of all the methods are shown in table 6.1 and plotted in fig. 6.1. The fig. shows that M spacing method performs better when compared to all other methods in terms of speed, for $N > 16$. However, modified DFT using basic DFT coefficients performs slightly better for small values of N . This is due to the overhead in computing the particular solution, in M spacing method, which is significant for small values of N .

Table 6.1: Execution time (in sec.) of 2-D DFT computational schemes for N

N	Direct DFT (s)	Closed form DFT(s)	Modified DFT		Visual DFT(s)	M spacing DFT(s)
			Direct(s)	using basic DFT(s)		
4	0.016	0.016	0.008	0.007	0.015	0.016
6	0.047	0.063	0.016	0.015	0.016	0.031
8	0.157	0.063	0.031	0.015	0.031	0.031
10	0.422	0.094	0.046	0.031	0.032	0.047
12	0.844	0.234	0.109	0.063	0.078	0.078
14	1.437	0.234	0.204	0.062	0.063	0.078
16	2.469	0.282	0.36	0.078	0.093	0.094
18	3.984	0.438	0.672	0.187	0.203	0.14
20	6.125	0.484	0.984	0.219	0.234	0.188
22	8.781	0.438	1.484	0.234	0.234	0.125
24	12.672	0.797	2.235	0.375	0.312	0.312
26	17.062	0.671	3.266	0.359	0.297	0.219
28	23.265	0.813	4.422	0.468	0.422	0.297
30	30.047	1.453	6.438	0.922	0.687	0.453
32	39.578	1.031	8	0.719	0.500	0.328
34	49.719	1.125	11.984	0.765	0.532	0.297
36	62.172	1.937	13.407	1.391	1.000	0.578
38	76.984	1.500	17.89	1.187	0.687	0.406
40	96.156	2.047	20.766	1.594	1.063	0.625
42	115.14	2.906	28.266	2.594	1.516	0.766
44	141.55	2.203	33.406	1.985	1.172	0.578
46	166.01	2.328	43.344	2.281	1.094	0.578
48	200.09	3.609	48.453	3.297	2.016	1
50	231.76	3.359	64.015	3.64	1.750	0.797
60	486.97	7.453	145.16	10.016	4.781	1.985
70	881.42	8.843	336.38	15.703	5.359	2.125
80	1407.1	11.375	575.94	20.156	7.641	3.078
90	2252	21.438	1131.6	52	15.297	5.438
100	3435.1	20.047	1735.1	51.187	14.125	5.11

**Fig. 6.1:** Comparison of execution time of different 2-D DFT computational schemes.

6.1.1.1 Computational complexity

In the visual method, Y_{k_1, k_2}^p of the basic 2-D DFT coefficients are computed either using the formula derived from the analysis of visual representation or by using the visual representation as a look up table rather than doing the computations in (1.6) & (1.7). These Y_{k_1, k_2}^p can be used to compute all the DFT coefficients by permutation over p . When N is the size of the data matrix and is even, from (4.1) the number of real additions for a $Y_{k_1, k_2}^p = \frac{N^2 \cdot dm}{M} - 1$,

$$(6.1)$$

where $\text{gcd}(k_1, k_2, M) = dm$.

$$\text{No. of real additions for a basic DFT coefficient} = \frac{M}{dm} \cdot \left(\frac{N^2 \cdot dm}{M} - 1 \right). \quad (6.2)$$

\therefore total number of real additions (A_R) required for the 2-D DFT computation is given by,

$$A_R = \sum_{dm} nb_{dm} \cdot \frac{M}{dm} \left(\frac{N^2 \cdot dm}{M} - 1 \right) = \sum_{dm} nb_{dm} \left(N^2 - \frac{M}{dm} \right), \quad (6.3)$$

where $M = N/2$, & nb_{dm} is the number of basic DFT coefficients for a given dm .

Total number of complex multiplications for a basic DFT coefficient = $M/dm - 1$.

Redundancy factor of a basic DFT coefficient = $nd_{dm} + 1 = \varphi(N/dm)$,

where nd_{dm} is the number of DFT coefficients that can be derived from the basic DFT coefficients by permutation.

Total number of complex multiplications (M_C) is same as the total number of complex additions (A_C) and is given by

$$M_C = A_C = \sum_{dm} nb_{dm} (nd_{dm} + 1) \left(\frac{M}{dm} - 1 \right) = \sum_{dm} nb_{dm} \cdot \varphi \left(\frac{N}{dm} \right) \cdot \left(\frac{M}{dm} - 1 \right), \quad (6.4)$$

Here one complex multiplication involves two real multiplications and no real additions since the data is real which is multiplied with the twiddle factor.

Difference between the M spacing method and the visual method of 2-D DFT is in the computation of Y_{k_1, k_2}^p and hence the number of complex multiplications and additions are same in both the cases, whereas the number of real additions differs. Total number of real additions (A_R) required for the 2-D DFT computation using M spacing method is given by,

$A_R = \text{Number of additions in layer 1} + \text{layer 2} + \text{layer 3}$.

Number of additions in layer 1 = $2 \cdot N \cdot M = N^2$

Number of additions in layer 2 = $4(M \cdot M) = N^2$

The number of additions in layer 3 can be calculated as follows:

$$\text{From (6.1), number of additions for a } Y_{k_1, k_2}^p \text{ in layer 3} = \frac{N^2 \cdot dm}{4 \cdot M} - 1 = M \cdot dm - 1. \quad (6.5)$$

$$\text{Number of additions for a basic DFT coefficient in layer 3} = M/dm(M \cdot dm - 1). \quad (6.6)$$

$$\text{Number of additions in layer 3} = \sum_{dm} nb_{dm} \cdot \left(M^2 - \frac{M}{dm} \right) \quad (6.7)$$

$$\therefore A_R = 2.N^2 + \sum_{dm} nb_{dm} \left(M^2 - \frac{M}{dm} \right) \quad (6.8)$$

Table 6.2 shows the number of real multiplications (M_R) and additions (A_R) required for 2-D DFT computation based on the visual approach and M spacing based method along with the number of computations required for the Direct 2-D DFT [130], row/column DFT [21], vector radix FFT [8] and row/column FFT [100] for different N . One complex multiplication in the above cases is equivalent to four real multiplications and two real additions. Even though the number of computations for vector radix FFT and r/c FFT is less, they require that the size of the data matrix to be of powers of 2. One must be willing to constrain oneself to sizes like 1024 & 2048 or zero pad one's data to the next power of 2 and thereby lowering the algorithm's efficiency. For many of the applications, this limitation on N cannot be tolerated. The visual approach and M spacing method, which might be less efficient in particular case, but provide much better coverage of allowed DFT size. Both methods allow the size of the data matrix to be any even N . So it is a trade off between flexibility and efficiency that make the visual approach and M spacing methods attractive. The results are also shown as a plot in fig. 6.2 and 6.3 as a logarithmic function of number of real multiplications and real additions respectively for different data size. The spike visible when $N = 30$ in the plot of visual/M spacing DFT, in fig. 6.2, is due to the fact that $N = 32$ is having more dm and hence requires low number of multiplications. The small spikes in the plot are also due to the same reason i.e., the next higher N is having more dm . The spike visible in the plots of visual and M spacing DFT in fig. 6.3 is for N having more number of divisors where the number of real additions is more. E.g., $N = 60$ is having more number of divisors when compared to $N = 62$. The spike is predominant in the plots of visual DFT.

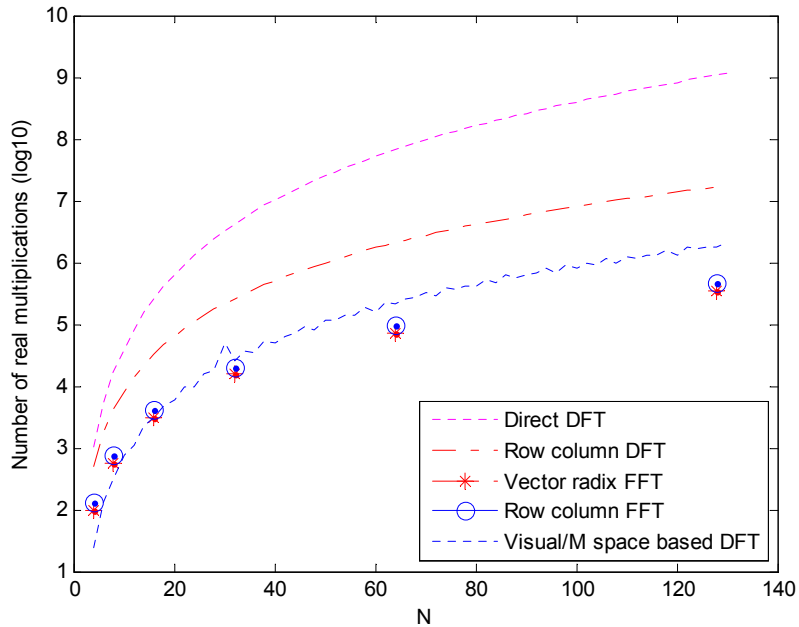


Fig. 6.2: M_R of different 2-D DFT computational scheme for N

Table 6.2: Comparison of computational complexity of 2-D DFT computation

N	Direct DFT		Row column DFT		Vector radix FFT		Row column FFT		Visual DFT		M space DFT	
	M_R	A_R	M_R	A_R	M_R	A_R	M_R	A_R	M_R	A_R	M_R	A_R
4	1024	896	512	512	96	96	128	192	24	168	24	80
6	5184	4752	1728	1728					128	796	128	328
8	16384	15360	4096	4096	576	576	768	1152	312	1656	312	728
10	40000	38000	8000	8000					768	3444	768	1544
12	82944	79488	13824	13824					1112	8104	1112	2992
14	153664	148176	21952	21952					2304	9132	2304	4232
16	262144	253952	32768	32768	3072	3072	4096	6144	3000	14520	3000	6200
18	419904	408240	46656	46656					4736	26284	4736	10408
20	640000	624000	64000	64000					5976	33480	5976	13280
22	937024	915728	85184	85184					9600	34236	9600	16328
24	1327104	1299456	110592	110592					9848	72376	9848	26008
26	1827904	1792752	140608	140608					16128	55956	16128	26888
28	2458624	2414720	175616	175616					17304	86952	17304	35600
30	3240000	3186000	216000	216000					49280	173668	49280	80968
32	4194304	4128768	262144	262144	15360	15360	20480	30720	26040	121272	26040	51128
34	5345344	5266736	314432	314432					36864	123492	36864	59912
36	6718464	6625152	373248	373248					35096	253480	35096	90832
38	8340544	8230800	438976	438976					51840	171612	51840	83528
40	10240000	10112000	512000	512000					50040	294456	50040	112856
42	12446784	12298608	592704	592704					63872	378428	63872	143816
44	14992384	14822016	681472	681472					70104	319656	70104	134768
46	17909824	17715152	778688	778688					92928	302316	92928	147848
48	21233664	21012480	884736	884736					83192	609784	83192	216952
50	25000000	24750000	1000000	1000000					115968	482844	115968	210344
52	29246464	28965248	1124864	1124864					116952	519624	116952	220832
54	34012224	33697296	1259712	1259712					139520	753340	139520	295528
56	39337984	38986752	1404928	1404928					142008	759288	142008	299864
58	45265984	44875760	1560896	1560896					188160	601812	188160	295688
60	51840000	51408000	1728000	1728000					162200	1415752	162200	477952
62	59105344	58628688	1906624	1906624					230400	733836	230400	360968
64	67108864	66584576	2097152	2097152	73728	73728	98304	147456	216504	990648	216504	415160
66	75898944	75323952	2299968	2299968					255488	1381132	255488	540424
68	85525504	84896640	2515456	2515456					264984	1138632	264984	488960
70	96040000	95354000	2744000	2744000					316416	1544148	316416	627848
72	107495424	106748928	2985984	2985984					285752	2216824	285752	773080
74	119946304	119135856	3241792	3241792					393984	1242612	393984	612872
76	133448704	132570752	3511808	3511808					371544	1578408	371544	680240
78	148060224	147111120	3796416	3796416					424832	2240516	424832	883784
80	163840000	162816000	4096000	4096000					412152	2465016	412152	932216
82	180848704	179745968	4410944	4410944					537600	1687236	537600	833288
84	199148544	197963136	4741632	4741632					457688	3621032	457688	1253744
86	218803264	217531152	5088448	5088448					620928	1944636	620928	960968
88	239878144	238515200	5451776	5451776					565752	2772024	565752	1126424
90	262440000	260982000	5832000	5832000					635264	4475860	635264	1635100
92	286557184	284999808	6229504	6229504					663192	2770344	663192	1200272
94	312299584	310638416	6644672	6644672					812544	2535372	812544	1254152
96	339738624	337969152	7077888	7077888					683768	5001976	683768	1771768
98	368947264	367064880	7529536	7529536					905472	3391308	905472	1537736
100	400000000	398000000	8000000	8000000					826776	4514280	826776	1759280
102	432972864	430850448	8489664	8489664					958592	4896148	958592	1951816
104	467943424	465693696	8998912	8998912					939768	4497336	939768	1842008
106	504990784	502608752	9528128	9528128					1168128	3628596	1168128	1797128
108	544195584	541676160	10077696	10077696					994136	7158568	994136	2545456
110	585640000	582978000	10648000	10648000					1254528	5642436	1254528	2363336
112	629407744	626597888	11239424	11239424					1158456	6337080	1158456	2467256
114	675584064	672620976	11852352	11852352					1342208	6780716	1342208	2712968
116	724255744	721133952	12487168	12487168					1337304	5494728	1337304	2393120
118	775511104	772225040	13144256	13144256					1614720	4998012	1614720	2477768
120	829440000	825984000	13824000	13824000					1302200	12364408	1302200	4077208
122	886133824	882502128	14526784	14526784					1785600	5521236	1785600	2737928
124	945685504	941872256	15252992	15252992					1635864	6694056	1635864	2919248
126	1008189504	1004188752	16003008	16003008					1782656	11471180	1782656	4215848
128	1073741824	1069547520	16777216	16777216	344064	344064	458752	688128	1764792	8007096	1764792	3345848
130	1142440000	1138046000	17576000	17576000					2081280	9156588	2081280	3866888

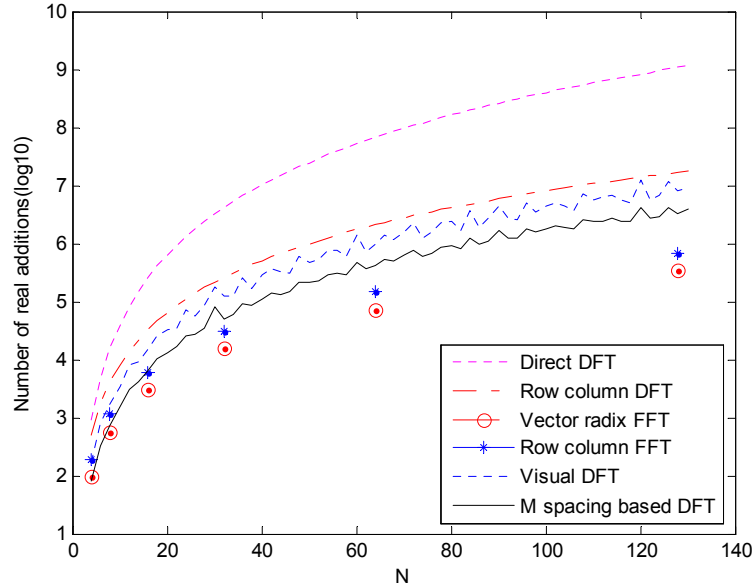


Fig. 6.3: A_R of different 2-D DFT computational scheme for N

6.1.2 2-D UMRT

The 2-D UMRT computation by three methods developed in section 5.6 namely, modified direct method, visual method and three layer M spacing based method are simulated using Matlab 7.0 on Intel® Pentium®4 CPU, 1.5 GHZ, machine. Table 6.3 shows the time taken (in seconds) for the computation of 2-D UMRT by the above schemes for different sizes of data matrix and fig. 6.4 shows the corresponding plot. It is seen that for smaller values of N up to 12, the modified Direct UMRT method performs slightly better. This is due to the overheads in the computation of particular solution for each UMRT coefficient, which is significant. However for higher values of N , the M spacing based UMRT out performs the other methods in terms of speed.

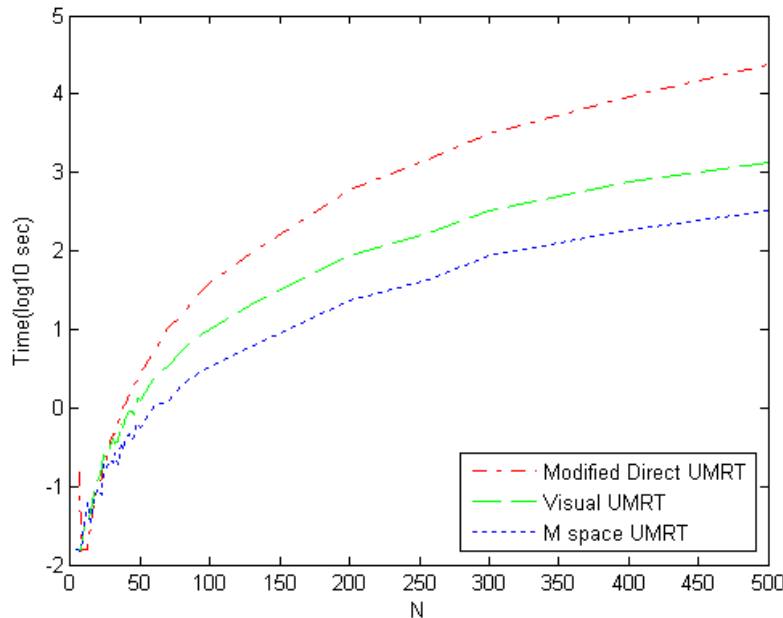


Fig. 6.4: Comparison of execution time for different 2-D UMRT computational schemes

Table 6.3: Execution time (in sec.) of 2-D UMRT computational schemes for N

N	Modified Direct UMRT (s)	Visual UMRT (s)	M spacing UMRT (s)
4	0	0.016	0.016
6	0.15	0.016	0.015
8	0.015	0.016	0.016
10	0.016	0.031	0.031
12	0.016	0.031	0.062
14	0.031	0.047	0.031
16	0.047	0.062	0.047
18	0.079	0.109	0.078
20	0.11	0.125	0.094
22	0.125	0.172	0.078
24	0.218	0.25	0.204
26	0.219	0.235	0.172
28	0.36	0.313	0.219
30	0.469	0.406	0.203
32	0.469	0.344	0.265
34	0.657	0.36	0.188
36	0.829	0.641	0.375
38	1.032	0.562	0.297
40	1.156	0.781	0.422
42	1.5	0.89	0.468
44	1.672	0.891	0.391
46	2.016	0.829	0.438
48	2.204	1.344	0.656
50	2.797	1.266	0.562
60	5.438	2.594	1.078
70	10.453	3.406	1.219
80	15.547	5.516	2.047
90	27.438	7.89	2.688
100	39.828	10.203	3.297
128	92.812	20.391	6.062
200	602.063	87.937	23.578
256	1455.297	169.25	42.484
300	3186.578	333.343	86.36
400	9197.656	757.359	185.047
500	23817.031	1324.5	321.625

6.1.2.1 Computational complexity

No complex operations are involved in 2-D UMRT computation. The number of real additions involved in the computation of 2-D UMRT can be calculated by subtracting the total number of real additions in layer 3 for the redundant MRT coefficients from that required for M spacing based 2-D DFT.

From (6.5), number of real additions for the redundant MRT coefficients corresponding to the basic

$$\text{DFT coefficients in layer 3} = \sum_{dm_e} nb_{dm_e} \cdot (M \cdot dm_e - 1) \cdot nr, \quad (6.9)$$

where dm_e is the gcd(k_1, k_2, M) when $M/dm \neq \varphi(N/dm)$.

$$i.e., A_{R(UMRT)} = 2 \cdot N^2 + \sum_{dm} nb_{dm} \left(M^2 - \frac{M}{dm} \right) - \sum_{dm_e} nb_{dm_e} (M \cdot dm_e - 1) nr. \quad (6.10)$$

The number of complex multiplications is a significant component in the 2-D DFT computation using M spacing based method, as can be seen in Table 6.4. Fewer number of MRT coefficients are computed in UMRT, when N is not a power of 2, which reduces the number of real additions required. When N is a power of 2, the number of real additions required for 2-D DFT and UMRT are same, since same number of MRT coefficients need be computed. From the table, it can be observed that for 2-D DFT computation, N having more number of divisors is suitable as it require less number of complex operations, whereas for 2-D UMRT, $N/2$ prime is suitable.

Table 6.4: Computational complexity of 2-D DFT & UMRT for M spacing method

N	M spacing based method			
	2-D DFT			2-D UMRT
	M_C	A_C	A_R	A_R
4	12	12	56	56
6	64	64	200	168
8	156	156	416	416
10	384	384	776	680
12	556	556	1880	1464
14	1152	1152	1928	1736
16	1500	1500	3200	3200
18	2368	2368	5672	4104
20	2988	2988	7304	6200
22	4800	4800	6728	6248
24	4924	4924	16160	14304
26	8064	8064	10760	10088
28	8652	8652	18296	16184
30	11200	11200	31688	20040
32	13020	13020	25088	25088
34	18432	18432	23048	21896
36	17548	17548	55736	39096
38	25920	25920	31688	24568
40	25020	25020	62816	52640
42	31936	31936	79944	52584
44	35052	35052	64664	59576
46	46464	46464	54920	52808
48	41596	41596	133760	99456
50	57984	57984	94376	77000
52	58476	58476	103880	96824
54	69760	69760	156008	106920
56	71004	71004	157856	138656
58	94080	94080	107528	104168
60	81100	81100	315752	195000
62	115200	115200	130568	126728
64	108252	108252	198656	198656
210	4033600	4033600	13887816	7413000
660	111453100	111453100	495508616	276243000
780	184461100	184461100	798561816	450972600
1540	1505888076	1505888076	4856834024	3220124600
4620	37685147500	37685147500	218507101512	106207193400
4622	49348252800	49348252800	49433704328	49412341448

6.1.3 Parallel distributed architectures and other methods for 8×8 point DFT

Three parallel distributed architectures and other methods for 2-D DFT computation are simulated in Matlab 7.4 and their time of execution for 8×8 point DFT, in AMD Athlon™ 64 processor running at 2.39 GHZ with a memory of 768 MB of RAM, is shown in table 6.5. As seen in section 6.1.1, for small values of N , modified DFT perform slightly better when compared to Modified DFT using basic DFT, M spacing and visual methods. This is due to the overhead due to the requirement of computing indices of basic DFT coefficients. This overhead is negligible for higher values of N when compared to the computation gain obtained by the new algorithms. In M spacing based method and visual method there is additional overhead due to the computation of particular solution, which is significant for small values of N . In M spacing based method, it is required to classify the basic DFT coefficients based on frequency index. This also contribute to the computation time and is significant for small values of N . M spacing based method is simulated in three different ways, one using the general algorithm developed in section 4.2.5.4, second and third one by using equations developed in section 4.2.4 with and without calculating the indices of basic DFT coefficients respectively. Column no. 6 of table 6.5 shows the timing of M spacing method, where layer 3 is computed directly using equations without the requirement of computing the particular solutions and hence the improvement in timing. The timing of M spacing method in column 7 is obtained by computing layer 3 directly using equations and by using pre-computed values for the indices of basic DFT coefficients. The performance improvement is noticed as the software overhead reduces. The timing in column 7 is comparable with the simulation results of version I and II architectures. Among the parallel distributed architectures, version I performs slightly better than the other two.

Table 6.5: Computation time for 8×8 point DFT by different methods

Direct DFT	Modified Direct	Modified Direct using basic DFT	Visual method	M spacing (using algorithm)	M spacing (using equations)	M spacing (using equations and without calculating Basic DFT coefficients)	Version I Architecture	Version II Architecture
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
40.25ms	6.893ms	12.123ms	14.417ms	24.215ms	6.995ms	6.733ms	6.544ms	6.569ms

6.1.4 Parallel distributed architectures and other methods for 8×8 point UMRT

Even though MRT computation [140] does not involve complex multiplications, redundancy is too high. In section 5.1 it is shown that UMRT coefficients can be obtained by removing all types of redundancy so that it require only the same memory as required for the original data. Table 6.6

shows the time of execution, in AMD Athlon™ 64 processor running at 2.39 GHZ with a memory of 768 MB of RAM, for both non architecture methods and the architecture methods for the computation of 8×8 point UMRT. The discussion in section 6.1.3 holds true here also. Version I and II architectures perform better when compared to all other methods.

Table 6.6: Computation time for 8×8 point UMRT by different methods

Modified Direct method	Visual method	M spacing (using algorithm)	M spacing (using equations)	M spacing (using equations and without calculating Basic DFT coefficients)	Version I	Version II
4.998ms	9.563ms	12.623ms	1.483ms	1.323ms	0.211ms	0.221ms

6.2 FPGA implementation of the architectures for 2-D UMRT

In the architecture models for the computation of 8×8 point DFT, UMRT coefficients are obtained at the output of the penultimate layer. All the three parallel distributed architectures are simulated and synthesized using Xilinx® 10.1 ISE where optimization selected is ‘balanced’ with a speed grade of -2. VHDL using behavioral modeling is used to describe the model. Target device selected is the Xilinx’s xc5v1x330-2-ff1760 of Virtex V family of FPGA. The device is so selected to have maximum number of I/O pins for fully parallel implementation of the architecture where all the 8×8 data will be available in parallel at the input and the UMRT coefficients will be available in parallel at the output pins at the end of computation. Test benches are created and the results of computation are verified with the actual values and found to be exact.

6.2.1 Fully parallel implementation of 2-D UMRT

In the fully parallel implementation of 8×8 UMRT it is assumed that the data size is 7 bit and all the 8×8 data arrives at the input in parallel, as shown in fig. 6.5. The 64 UMRT coefficients, each of 8 bit size, are available in parallel at the output after computation. The massive parallelism available in FPGA is exploited. The implementation uses fully combinational logic.

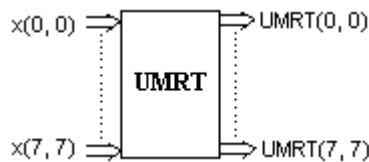


Fig. 6.5: Fully parallel 8×8 UMRT

The results of the synthesis of the three parallel distributed architectures are shown in table 6.7. It is noticed that the M spacing model performs slightly better when compared to the other two in terms of speed, where as it outperforms the other two in terms of the area. Moreover the version I and version II architectures are designed for $N = 8$ only and the algorithm cannot be extended to other values of N . The M spacing based architecture can be used for any even value of N . Table 6.7 also shows the synthesis results of 10×10 point UMRT. Two devices are required to implement the same due to the limitation in the maximum number of I/O buffers (1200) available in the target device selected.

Table 6.7: Synthesis results of fully parallel version of 2-D UMRT

Performance factors	8 × 8 UMRT			10 × 10 UMRT
	Version I	Version II	M Spacing	M spacing
No. of adders/subtractors	594	527	371	673
Cell usage- BELs	12276	10544	8027	14436
No. of SLICE LUTs	4584	4032	2920	
Max. combinational path delay	10.848ns	10.848ns	10.618ns	25.211ns
Logic delay	6.984ns	6.984ns	7.007ns	12.102ns
Routing delay	3.864ns	3.864ns	3.611ns	13.109ns

6.2.2 Different schemes of M spacing based architecture for 8 × 8 point UMRT

DSP algorithms are used in various real time applications with different sampling rate requirements. The different sample rate and computation requirements necessitate different architecture considerations for implementation of DSP algorithms. Different schemes for M spacing based 8×8 point UMRT implemented in FPGA are illustrated below.

6.2.2.1 Fully parallel implementation of four layer architecture

In the M spacing based architecture synthesized in section 6.2.1, the number of computations in layer 3 is not same for all coefficients. The number of terms to be added, for UMRT coefficients in layer 3, when $\gcd(k_1, k_2, M) > 1$ is $\gcd(k_1, k_2, M)$ times more than that required for the computation of UMRT coefficients with $\gcd(k_1, k_2, M) = 1$. In order to make the number of computations of each UMRT coefficient in layer 3 the same, i.e., $M - 1$ additions, it is split into two as in section 4.2.3. The number of additions in the newly formed layer (i.e. layer 4) is $\gcd(k_1, k_2, M) - 1$. The number of computations in layer 4 is maximum for UMRT coefficients when $\gcd(k_1, k_2, M) = M$ and minimum when $\gcd(k_1, k_2, M) = 1$. The synthesis results are as shown in table 6.8. Due to the increase in number of layers, both the hardware requirement and time of execution has increased when compared to the three layer architecture. But the model is quite suitable for pipelined

computation in which the computation of a new block of data starts before completing the computation of the previous data set.

Table 6.8: Synthesis results of fully parallel, 3 & 4 layer M spacing 8×8 point UMRT

Performance factors	M Spacing (three layer)	M spacing (four layer)
No. of adders/subtractors	371	393
Cell usage- BELs	8027	8228
No. of SLICE LUTs	2920	3047
Max. combinational path delay	10.618ns	11.073ns
Logic delay	7.007ns	7.007ns
Routing delay	3.611ns	4.066ns

6.2.2.2 Semi parallel implementation

In this type of implementation data are read in parallel. 8×8 UMRT coefficients are available in parallel at the output and the computation of first two layers is also done in parallel. But in layer 3 all UMRT coefficients are computed concurrently, while computation of each UMRT coefficient is done sequentially. The table 6.9 shows synthesis result of the maximum combinational path delay (MCPD) in different scenarios.

Table 6.9: MCPD for the computation of different combinations of UMRT coefficients

Computation of UMRT coefficients involved	Max. comb. path delay (ns)
All N^2 coefficients	18.645
Coefficients when $\gcd(k_1, k_2, M) = 4$ alone	18.590
Coefficients when $\gcd(k_1, k_2, M) = 2$ alone	13.285
Coefficients when $\gcd(k_1, k_2, M) = 1$ alone	10.215
Coefficients when $\gcd(k_1, k_2, M) = 4$ and 2	18.613
Coefficients when $\gcd(k_1, k_2, M) = 2$ and 1	13.432
Coefficients when $\gcd(k_1, k_2, M) = 1$ and UMRT(4, 2) & UMRT(4,6)	13.105
Coefficients when $\gcd(k_1, k_2, M) = 1$ and UMRT(0, 0)	14.929
Coefficients when $\gcd(k_1, k_2, M) = 1$ and UMRT(2, 0)	12.765

In table 6.9, the UMRT coefficients when $\gcd(k_1, k_2, M) = 4$ takes maximum time due to the $M^2 - 1$ computations required, while the computation of UMRT coefficients when $\gcd(k_1, k_2, M) = 1$ takes minimum time as there are only $M - 1$ computations for each UMRT coefficient. First two rows of the table 6.9 shows that apart from the logic delay, as the number of UMRT coefficients increases, routing delay also increases.

6.2.2.3 Parallel distributed architecture with data in/out serially

a) Single data in and single coefficient out

In this type of implementation data are read in serially one per clock cycle. This requires 64 clock cycles. After all the data are available, the computations are done in parallel. The results of

the computations, i.e. the UMRT coefficients are send out one per clock cycle. The UMRT chip is shown in fig. 6.6 and the synthesis results are shown in table 6.10. The entire process requires $64 + 64 = 128$ clock cycles.

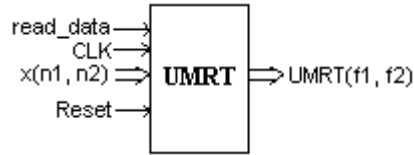


Fig. 6.6: 8×8 UMRT chip with one data in (parallel implementation)

b) *Two data in and single coefficient out*

Two data at M spacing as in fig. 6.7 are read, during each clock cycle. The computations in parallel commence only after all the 8×8 data are available. The results are sent out serially one per clock cycle. Total number of clock cycles required for the entire operation is $32 + 64 = 96$. The synthesis results are depicted in table 6.10.

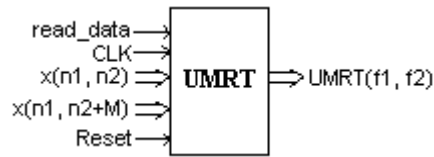


Fig. 6.7: 8×8 UMRT chip with two data in (parallel implementation)

c) *Four data in and single coefficient out*

In fig. 6.8, four data at M spacing are read in one clock cycle. The number of clock cycle to read in all the data reduces to 16 at the expense of increase in the number of input pins. After all the data are available the computations are carried out in parallel and the results are sent out serially one per clock cycle. Total number of clock cycles required is $16 + 64 = 80$. In the block diagram shown in fig. 6.9, the memory block is used to store the input data, which is a fully parallel one. The synthesis results are in table 6.10.

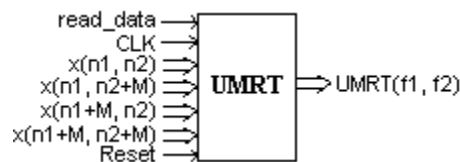


Fig. 6.8: 8×8 UMRT chip with four data in (parallel implementation)

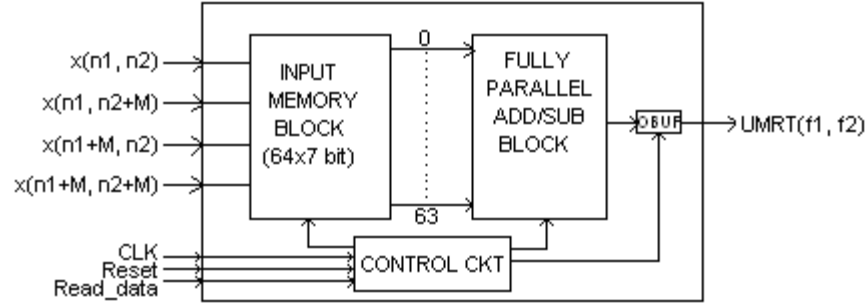


Fig. 6.9: Block diagram of the 8×8 UMRT chip with four data in

Table 6.10: Synthesis results of fully parallel, three layer M spacing 8×8 point UMRT

Performance factors	Single Data in, single coefficient out	Two Data in, single coefficient out	Four Data in, single coefficient out
Max. frequency	620.848 MHZ	510.347 MHZ	629.743 MHZ
Cell usage: BELS	8317	8278	8260
Flip flops	538	536	534
IO Buffers	17	24	39
Clock buffers	1	1	1

6.2.2.4 Sequential implementation

a) *Four input serially and one result out sequentially*

In fig. 6.10, four data at M spacing are read in parallel per clock cycle and the computation of the pattern C, D, E and F corresponding to the above data are also done and stored in the registers. 16 clock cycles are required to read in the 8×8 data. After all the data are read in, the computation of UMRT is carried out. First UMRT coefficient is send out in the 17th clock cycle. 64 UMRT coefficients are sent out sequentially which takes 64 clock cycles. 80 clock cycles are required for the entire results to be available. The maximum frequency of operation is 581.311 MHZ. This frequency mainly depends on the maximum computational delay in any of the clock cycles. The maximum time of computation required for UMRT coefficients is when $\gcd(k_1, k_2, M) = 4$ i.e., in the computation of UMRT(0, 0), UMRT(4, 0), UMRT(0, 4) and UMRT(4, 4) where there are 7 additions. The synthesis results are shown in table 6.11. This architecture requires that the data should be given in a definite order.

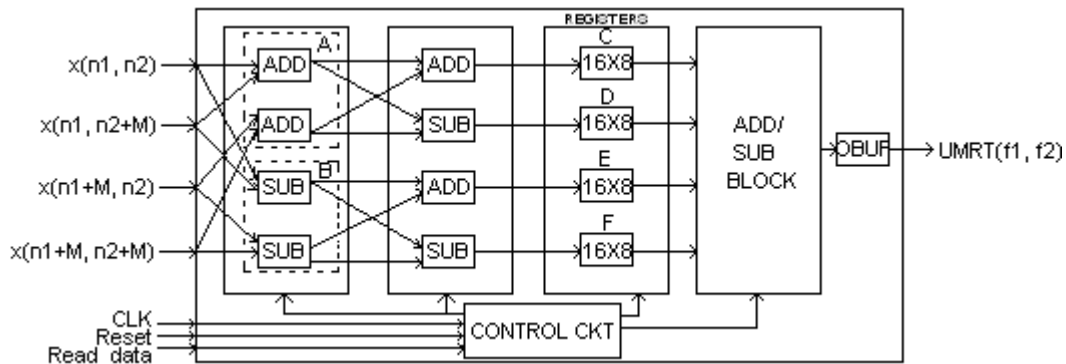


Fig. 6.10: 8×8 UMRT chip with four Data in (sequential implementation)

b) *Two input serially and one UMRT out sequentially*

In this scheme two data are read at a time in each clock cycle and the corresponding A and B values are computed, as in fig. 6.11. After reading all the data and computing A and B, the computation of C, D, E and F commences. One value each of C, D, E and F are computed per clock cycles and hence require 16 clock cycles to compute. In the 49th clock cycle first UMRT coefficient is computed and the result is sent out through the output pins. In the subsequent clock cycle, UMRT coefficients are computed and sent out sequentially. So the entire process requires $32 + 16 + 64 = 112$ clock cycles.

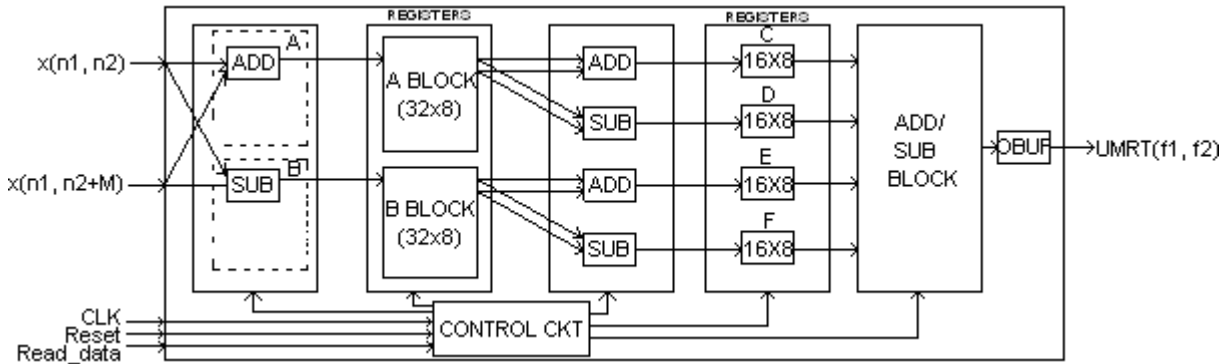


Fig. 6.11: 8×8 UMRT chip with two Data in and single UMRT out serially

Table 6.11: Synthesis results of 8×8 UMRT chips shown in fig. 6.10 and 6.11

Performance factors	Four Data in, single coefficient out	Two Data in, single coefficient out
Max. frequency	581.311 MHZ	631.532 MHZ
Cell usage: BELS	2375	3725
Flip flops	530	1044
IO Buffers	39	25
Clock buffers	16	16

6.2.2 Comparison of different FPGA implementations

FPGA implementation of version I, II and three layer M spacing based method for 8×8 UMRT computation, as in section 6.2.1, shows that the M spacing method is better in terms of time and space as it require less number of adders.

Routing delay of 8×8 UMRT and 10×10 UMRT using M spacing architectures in section 6.2.1 is 3.611ns and 13.109ns respectively. The increase in routing delay for 10×10 UMRT is due to the increase in hardware complexity as it requires an additional 302 adders. Moreover the generated hardware is a highly interconnected one as each output of layer 2 in fig. 5.6 is required for computing several UMRT coefficients. This also contributes to the routing delay.

The cell usage of fully parallel implementation of 8×8 UMRT using four layer M spacing parallel distributed architecture in section 6.2.1 is 8027, whereas that of sequential implementation

with four data input, as in section 6.2.2.4, requires only 2375. The number of external I/O data pins required for the fully parallel version is 960 whereas that of the sequential one is 36. However, the sequential version requires data storage and there will be problems connected with clocks. The entire computation in fully parallel version takes only 10.618ns, whereas the sequential version require 80 clock cycles with a clock frequency of 581 MHZ. Hence the selection of either of the above architecture is a trade off between time and space. Both the time and space is high in the implementation developed in 6.2.2.2, since the input/output are sequential and the computations are in parallel.

6.3 Conclusion

The 2-D DFT/UMRT computation using M spacing based method performs better when compared to the existing methods as well as the other methods developed namely visual method and Modified direct method. Table 6.4 suggests that the M spacing based method is more suitable for 2-D DFT computation when N has more number of divisors, whereas 2-D UMRT computes faster for $N/2$ prime. FPGA implementations of 8×8 UMRT using different methods show that the M spacing based method has the potential to be used for high speed real time applications. The different schemes implemented for M spacing based method show its flexibility in implementation to suit different applications. It is the trade off between time and space, which determines the selection of proper scheme for the application.

CHAPTER 7

DISCUSSIONS AND CONCLUSIONS

Visual representation of 2-D DFT coefficients in terms of 2×2 data is presented in chapter 3 and the same is analyzed to develop an algorithm to compute 2-D DFT.

7.1 Visual representation

A visual representation of DFT coefficients based on 2×2 DFT was developed in [88] for ease of analysis of 2-D signals in the frequency domain. The DFT coefficients were represented visually using a set of primitive symbols derived from the relation between 2×2 data and 2×2 DFT coefficients. This gives a visual representation of the relation between $N \times N$ point DFT and 2×2 point DFTs. A visual representation based on 2×2 data, presented in chapter 3 on the other hand, gives a direct relationship between time domain data and the frequency domain representation in terms of visuals. These visuals are the representatives of data points involved in the computation of DFT coefficients. Thus the frequency domain analysis of 2-D signals using these visuals is possible without computing the DFT coefficients, there by reducing the computational requirement significantly.

The important features of the visual representation based on 2×2 data are discussed below.

7.1.1 The software

The visual representation of 2-D DFT coefficients in terms of 2×2 data for any even value of N can be constructed using the software developed in VC++. If the visual representation of a few selected DFT coefficients is only required, the software allows the user to indicate the desired frequency index and obtain the visual output. Similarly if only a selected few Y_{k_1, k_2}^p are required, by indicating the desired frequency and phase, the user will obtain the desired visual representation.

7.1.2 Computation

The visual representation is data independent and hence can be used as a lookup table to compute the DFT coefficient of any index (k_1, k_2) . The position and nature of the data involved in

computing Y_{k_1, k_2}^p of any DFT coefficient can be analyzed. The patterns available in the selected few DFT coefficients can be analyzed easily and can be used to generate simple and efficient schemes for its computation.

7.1.3 Exploitation of redundancy

Redundancy in computation present at different levels has been analyzed using the visual representation, which enables to reduce the computational complexity. E.g., Y_{k_1, k_2}^p of many DFT coefficients are either same or sign reversed form of some other DFT coefficients. Hence it is enough to compute Y_{k_1, k_2}^p of one among such DFT coefficients named basic DFT coefficient and other coefficients could be derived. The fig. 3.5 shows that the number of basic DFT coefficient is high when the number of dm is more. It is more efficient when $N/2$ is prime, where nb is only $2N + 8$.

Section 5.1 shows that, when N is not a power of 2, many Y_{k_1, k_2}^p of several DFT coefficients need not be computed due to derived redundancy. There is no derived redundancy when N is a power of 2. The derived redundancy is eliminated to obtain UMRT, which can be used as an alternate way of representing signals. From table 5.13, the derived redundancy is high when N has more number of divisors. The number of real additions is low when $N/2$ is prime as in table 6.4. Hence UMRT will be most efficient when $N/2$ is prime.

In section 3.2.3.1, it is observed that one cell is enough to represent the coefficients in group 1 and one row/column for group 2 and 3 coefficients. Also deduced that the visual representation of Y_{k_1, k_2}^p for $p = 0$ if available, the visual representation for other values of p could be obtained by circular shift of the pattern. Hence the visual representation of Y_{k_1, k_2}^p for any one value of p is enough to represent the DFT coefficient. So flexibility in the visual representation scheme can be provided to effectively utilize the memory without affecting the speed. E.g., if there is memory constraint, storing of the visual representation can be done by eliminating the redundancy at all levels, so that at the time of retrieval, it can be derived by proper circular shift of the identical row/column. Thus the visual representation of unique cells of the unique Y_{k_1, k_2}^p need be stored and others could be derived.

It is seen in section 3.2.4 that the visual representation of Y_{k_1, k_2}^p corresponding to lower orders of N are contained in higher orders. The redundancy in visual representation of Y_{k_1, k_2}^p between different N 's is illustrated using theorem 3.4 and 3.5. Thus it is enough to store the visual

representation of few higher orders of N so that the visual representation of all Y_{k_1, k_2}^p of lower orders of N if required can be obtained by visual manipulation, using the relation stated in the theorems.

7.2 Computation of 2-D DFT

Theorem 3.1 shows that the existence of Y_{k_1, k_2}^p depends on the divisors, dm , of M and that the number of complex multiplications for a DFT coefficient is not exactly $N/2$, but less than that depending on the divisors. The number of complex multiplications for a DFT coefficient is $M/dm - 1$. Thus the number of complex multiplications is less for N with more number of divisors.

Due to the redundancy present among Y_{k_1, k_2}^p of several DFT coefficients, Y_{k_1, k_2}^p of a basic set of DFT coefficients need be computed and others could be derived. The mathematical relation developed in section 3.2.5 for the number of basic DFT coefficients show that, it is low when $N/2$ is prime and high when N is having more number of divisors. The algorithm presented in section 3.2.6 gives a procedure for computing the index values of the basic set of DFT coefficients. The 2-D DFT computation can be simplified using the basic DFT coefficients identified.

The patterns in the basic DFT coefficients have been analyzed to derive an algorithm for its computation. Using the visual approach, entire 2-D DFT coefficients, for any even N , could be computed by permutation over p . Computation of selected few DFT coefficients is also possible. The visual approach for 2-D DFT computation outperforms the conventional DFT computation, modified DFT and closed form method in terms of speed as seen in figure 6.1.

7.3 Architecture

Parallel Distributed architecture developed in [71] is suitable for 2-D DFT for a particular order N such that $((N))_4 = 2$. Using a similar approach, Version I and Version II parallel distributed architecture for the computation of 8×8 point DFT is developed based on the analysis of visual representation in terms of 2×2 DFT. A parallel distributed approach is employed in which the computations are in terms of real additions. There are 4 layers of computational units. The last layer alone involves complex operations that too are scaling by the pre-computed twiddle factor values. The architecture has a highly parallel structure and can be employed, since most of the image processing applications use the standard 8×8 size. The architectures are developed as a step towards extending the model to implement 2-D DFT for $((N))_4 = 0$. A generalized architecture can then be derived for an even N . But the primitive symbols combination for group 2 and 3 coefficients differs for $((N))_4 = 2$ and $N = 8$. There is significant difference in the architecture developed with that of [71] and hence a generalized architecture based on the above approach is not feasible.

The analysis of visual representation based on 2×2 data lead to the development of M spaced architecture for 2-D DFT computation. The model is capable of implementing 2-D DFT of any even N . 2-D DFT computation can be done for odd values of N using the M spacing method, by padding one row and column with zero to make it even.

There are 4 layers of computational units in the M spacing based architecture. The computations in the processing layers are in terms of real additions and hence the speed of operations is high. Scaling by the twiddle factors in the last layer is the only complex operations involved in this architecture. Redundancy of computation at various levels has been eliminated. The number of complex multiplication and addition are same for version I, II and M spacing based architectures, for an 8×8 data matrix. However the number of real addition are 592, 504 and 416 for version I, II and M spacing based 8×8 point DFT computation respectively.

7.4 2-D UMRT

The 2-D UMRT coefficients for any even N are unique, numerically compact and require only the same memory space as required for the original image. If the signal can be represented in terms of 2-D UMRT coefficients, then the only computation required will be the real additions. In versions I and II architectures shown in fig. 4.2 & 4.4, for the computation of 8×8 point DFT, UMRT coefficients are available at the output of layer 3, except for the group 1 coefficients. One real addition performed for the group 1 coefficients in the fourth layer has to be accommodated in the third layer. So the layer 3 has to be modified slightly. The computations are in terms of real addition only. So the speed of operation is high and memory requirement is low. In the four layer M spacing based architecture shown in fig. 4.8, which compute the $N \times N$ point DFT, where N is even, MRT coefficients corresponding to the basic DFT coefficients are available at the output of layer 3. For the computation of 2-D UMRT, the fourth layer can be fully avoided. The third layer is modified in fig. 5.6 so that we need compute only UMRT coefficients rather than computing all Y_{k_1, k_2}^p corresponding to the basic DFT coefficients. There is reduction in computation for layer 3, as shown in table 6.4. Simulation results in MALAB®, section 6.1.2, shows that M spacing based 2-D UMRT is better in terms of speed when compared to the existing methods and the visual approach.

7.5 FPGA implementation

The three fully parallel distributed architecture for the computation of 8×8 point UMRT, namely version I, version II and M spacing method, are implemented in FPGA using Xilinx®. The synthesized results show that the M spacing based architecture performs better when compared to the other two architectures in terms of area and speed. Moreover it is a generalized architecture.

FPGA can be reconfigured to compute 2-D UMRT for the data matrix of any even N . Different schemes of M spacing based architectures synthesized in FPGA reveal that it can be modified to meet the different constraints such as area and speed.

7.6 Suggestions for further work in the field

7.6.1 Visual representation

1. Visual representation of DFT coefficients can be made more interactive by incorporating the result of the analysis. E.g., the software may be modified to display the basic DFT coefficients, of a data matrix of size N , having a particular value of $\gcd(k_1, k_2, M)$. Thus by integrating the visual and analytical methods, only the relevant visuals need be handled for further analysis.
2. The software can be modified to highlight the patterns or features available in the selected coefficients. E.g., in applications like texture analysis where a few DFT coefficients need be computed, the identification of similar patterns in such coefficients can be used to derive simple computational techniques for hardware implementation.
3. Instead of using the primitive symbols based on 2×2 data, the data in an MRT coefficient can be represented using “o” for data to be added and “•” if the corresponding data is to be subtracted. Absence of a symbol indicates that the data in the corresponding position is not involved in the computation. This type of representation scheme if employed can be used to obtain the visual representation of a DFT/MRT coefficient from that of a known coefficient by visual manipulation. In cases where memory is a constraint, the visual representation of a selected few coefficients need be stored, in the form of one cell or one row/column of cells while that of the other coefficients, if required, can be obtained by visual manipulation.
4. In section 3.2.4, it is shown that the visual representation of Y_{k_1, k_2}^p corresponding to lower orders is contained in higher orders. Similarly many of the visual representation of Y_{k_1, k_2}^p of higher orders can be constructed using the visual representation of lower orders. Analysis of the visual representation of different N can be carried out to obtain all Y_{k_1, k_2}^p of higher orders from lower orders by visual manipulation. This will enable to compute 2-D DFT of higher orders using the hardware, which compute lower orders.

7.6.2 Algorithm

1. 2-D signal representation using basic DFT coefficients is possible and hence can be used to form a variable length transform. The length of the transform is $2N + 8$ for $N/2$ prime and $3N - 2$ when N is a power of 2. The properties of the variable length transform can be analyzed.

2. Algorithm for computing the inverse of the transform formed from the basic DFT coefficients can be developed.
3. A suitable algorithm is to be developed which will compute 2-D DFT of higher orders, if the visual representation of lower orders are available and vice-versa.
4. 2-D IUMRT algorithm is to be developed.
5. The placement scheme satisfies the linearity property. It does not satisfy the matrix properties such as transpose and hence the possibility of developing a suitable scheme for placement of UMRT coefficients can be explored.
6. The possibility of obtaining a direct mapping between data and UMRT coefficients is to be explored.
7. During the analysis of derived redundancy in section 5.1, Y_{k_1, k_2}^p of a DFT coefficient is combined to obtain Y_{k_1, k_2}^p of some other DFT coefficient. Relation between the frequency indices of these DFT coefficients can be analyzed.

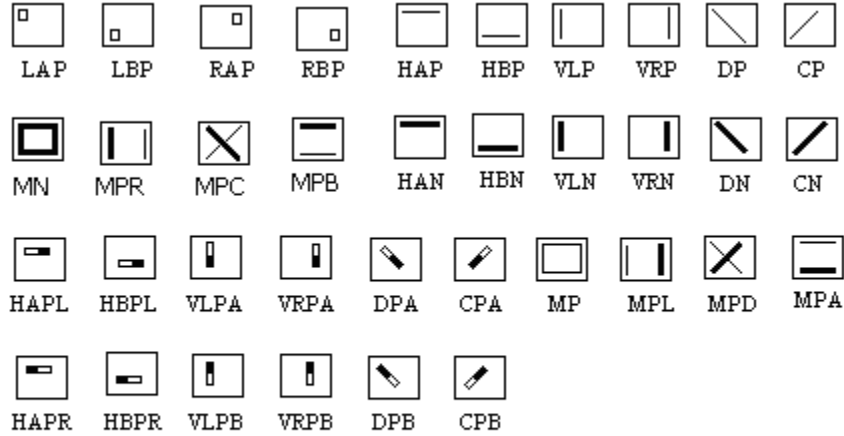
7.6.3 Architecture

1. Pipelined techniques can be incorporated in the architecture to improve performance. E.g., the number of computations for UMRT coefficients depends on $\gcd(k_1, k_2, M)$. In a fully parallel implementation, this is a bottle neck for coefficients with higher value of $\gcd(k_1, k_2, M)$. This can be minimized by employing parallel/pipeline technique.
2. 8×8 point DFT computation using version I and II parallel distributed architecture can be implemented in FPGA.
3. $N \times N$ DFT computation using four layer and five layer M spacing parallel distributed architectures can be implemented in FPGA.
4. An architecture is to be developed which will compute 2-D DFT/UMRT of higher orders using that of lower orders and vice-versa.
5. Parallel distributed architecture for the computation of 2-D IDFT/IUMRT can be developed.
6. Parallel distributed architecture incorporating the duality property of 2-D UMRT/IUMRT is to be developed.
7. A suitable scheme to reconfigure FPGA, multiple times during computation, for different size of the data matrix as per the application demands.

APPENDIX A

VISUAL REPRESENTATION OF DFT COEFFICIENTS BASED ON 2×2 DFT

A.1 Primitive symbols and its Mnemonics



Legend: V-Vertical H-Horizontal R-Right L-Left A-Above B-Below

P-Positive N-Negative D-Diagonal C-Cross Diagonal M-Matrix

Fig. A.1: List of primitive symbols based on 2×2 DFT

In order to understand picture in general the following rule may be applied:

- A white rectangle implies that the DFT coefficient at that point is to be added.
- A black rectangle implies that the DFT coefficient at that point is to be subtracted.
- Thin lines indicate that the nodes involved are to be added.
- Bold lines indicate that the nodes involved are to be subtracted.

The meaning of a few primitive symbols and the corresponding mnemonics used in the visual representation are as below:

1. Symbol named LAP (Left Above Positive) indicates that the DFT coefficients on the $(0, 0)^{\text{th}}$ position of the 2×2 DFT is taken with a positive sign and rest of the data are not considered. Similarly, RAP (right above positive), RBP (right below positive), LBP (left below positive) indicates consideration of DFT coefficients at positions $(0, 1)$, $(1, 1)$ and $(1, 0)$ respectively. It is to be noted that in the visual representation a hollow square symbol represents single positive DFT point.
2. Symbol named DP (diagonal positive) indicates that the two DFT coefficients on the diagonal of the 2×2 DFT matrix i.e. at position $(0, 0)$ and $(1, 1)$ are taken with a positive sign and the rest are not considered. A thin diagonal line is shown in the visual representation. Similarly CP (cross-diagonal positive) considers DFT points on the cross-diagonal i.e. at position $(0, 1)$ and $(1, 0)$. A thin cross-diagonal line shows it.

A.2 Visual representation of 8×8 DFT based on 2×2 DFT

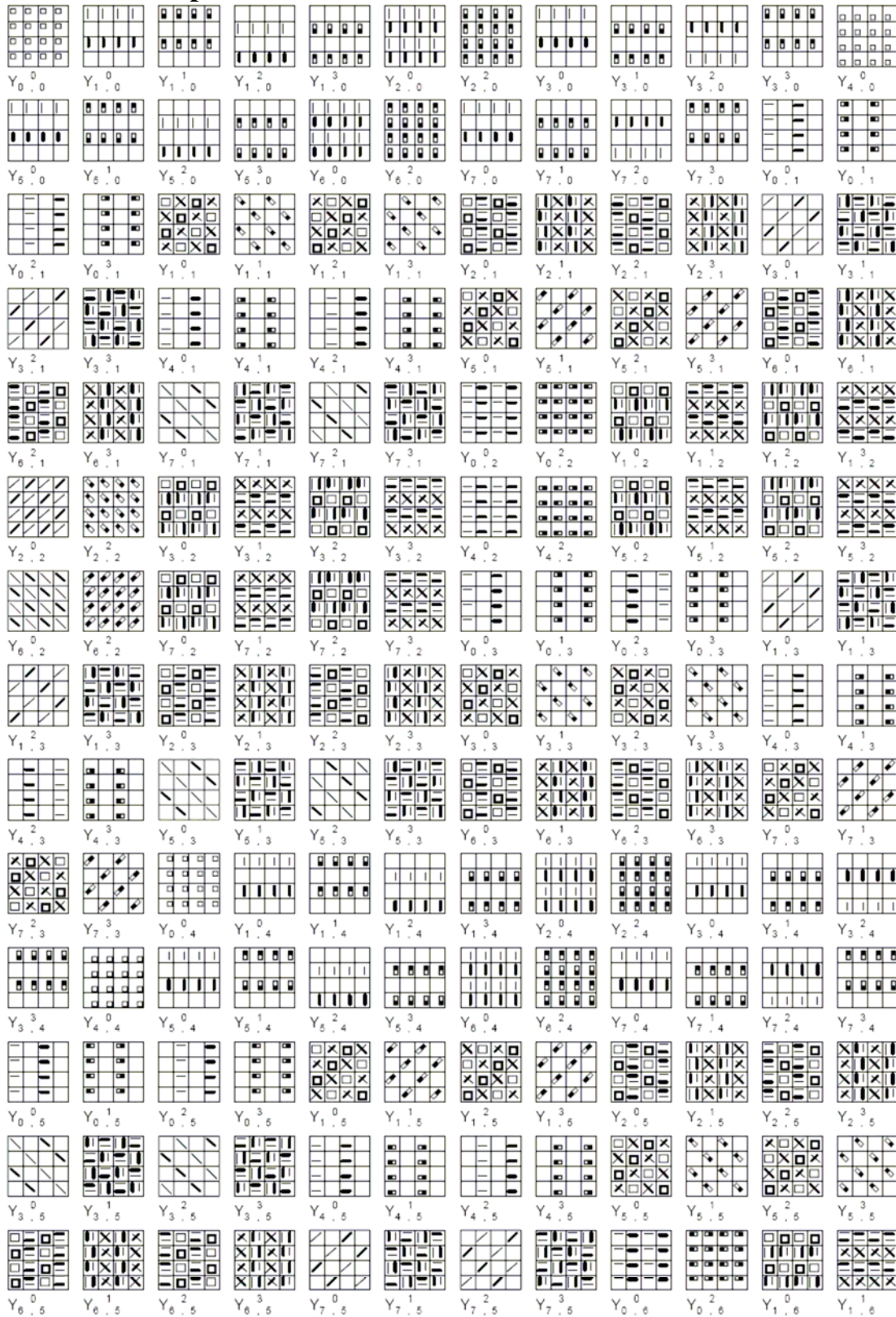


Fig. A.2: Visual representation based on 2×2 DFT for $N = 8$

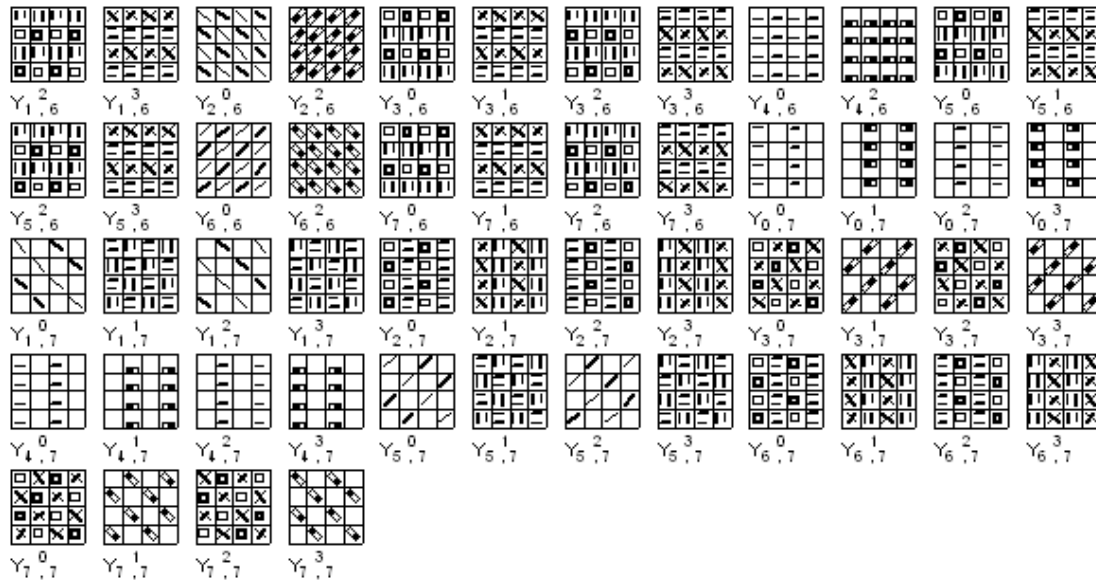


Fig. A.2: Visual representation of DFT coefficients for $N = 8$ (contd...)

A.3 Matrix showing the grouping of DFT coefficients for $N = 8$

$$\mathbf{Y} = \begin{bmatrix}
 \textcircled{Y_{0,0}} & Y_{0,1} & Y_{0,2} & Y_{0,3} & \textcircled{Y_{0,4}} & Y_{0,5} & Y_{0,6} & Y_{0,7} \\
 Y_{1,0} & Y_{1,1} & Y_{1,2} & Y_{1,3} & Y_{1,4} & Y_{1,5} & Y_{1,6} & Y_{1,7} \\
 Y_{2,0} & Y_{2,1} & Y_{2,2} & Y_{2,3} & Y_{2,4} & Y_{2,5} & Y_{2,6} & Y_{2,7} \\
 Y_{3,0} & Y_{3,1} & Y_{3,2} & Y_{3,3} & Y_{3,4} & Y_{3,5} & Y_{3,6} & Y_{3,7} \\
 \textcircled{Y_{4,0}} & Y_{4,1} & Y_{4,2} & Y_{4,3} & \textcircled{Y_{4,4}} & Y_{4,5} & Y_{4,6} & Y_{4,7} \\
 Y_{5,0} & Y_{5,1} & Y_{5,2} & Y_{5,3} & Y_{5,4} & Y_{5,5} & Y_{5,6} & Y_{5,7} \\
 Y_{6,0} & Y_{6,1} & Y_{6,2} & Y_{6,3} & Y_{6,4} & Y_{6,5} & Y_{6,6} & Y_{6,7} \\
 Y_{7,0} & Y_{7,1} & Y_{7,2} & Y_{7,3} & Y_{7,4} & Y_{7,5} & Y_{7,6} & Y_{7,7}
 \end{bmatrix}$$

Fig. A.3: Matrix showing the grouping of DFT coefficients for $N = 8$

APPENDIX B

B.1 Greatest Common Divisor (gcd)

Definition: Let $a, b, c \in \mathbb{Z}$. If $a \neq 0$ or $b \neq 0$, $\gcd(a, b)$ is defined to be the largest integer d such that $d \mid a$ and $d \mid b$ and is denoted as $g(a, b)$.

gcd properties:

1. If $e \mid a$ then $-e \mid a$.
2. If $a \neq 0$, then the largest positive integer that divides a is $|a|$.
3. $\gcd(a, b) = \gcd(|a|, |b|)$.
4. $\gcd(a, b) = \gcd(b, a)$.
5. If $a \neq 0$ or $b \neq 0$, then $\gcd(a, b)$ exists and satisfies

$$0 < \gcd(a, b) \leq \min\{|a|, |b|\}.$$

6. $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$.

B.2 Linear Diophantine Equation

Diophantine equations are equations that require integer solutions. The linear Diophantine equation $ax + by = c$ has a solution only if $\gcd(a, b)$ divides c . In that case, there are infinite number of solutions given by: $x = x_0 + (b / \gcd(a, b))t$, $y = y_0 - (a / \gcd(a, b))t$, where (x_0, y_0) is a solution, $t \in \mathbb{Z}$.

B.3 Bezout's Lemma:

For all integers a and b there exist integers s and t such that

$$\gcd(a, b) = s.a + t.b$$

B.4 Theorems on Linear Congruence

Theorem B.4.1:

If $a.c \equiv ((b.c))_m$ and $\gcd(c, m) = g$, then $a \equiv ((b))_{m/g}$

Theorem B.4.2:

The linear congruence $a.x + b.y \equiv ((c))_m$ has solutions if and only if $g \mid c$ where $g = \gcd(a, b, m)$.

Theorem B.4.3:

If $\gcd(a, m) = 1$ or $\gcd(b, m) = 1$, then the linear congruence $a.x + b.y \equiv ((c))_m$ has exactly m incongruent solutions.

Theorem B.4.4:

The linear congruence $a.x + b.y \equiv ((c))_m$ has exactly gm incongruent solutions, where $g = \gcd(a, b, m)$, provided $g \mid c$.

B.5 Principle of Inclusion Exclusion

If A_1, A_2, \dots, A_n are finite set, then

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| - \sum_{i,j:1 \leq i < j \leq n} |A_i \cap A_j| + \sum_{i,j,k:1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \dots + (-1)^{n-1} |A_1 \cap \dots \cap A_n|,$$

where $|A|$ denotes the cardinality of the set A .

B.6 Euclidean Algorithm, Extended Euclidean Algorithm.

The Euclidean algorithm is used to determine the gcd of any two integers.

Let $a, b \in \mathbb{Z}$ be such that $a \geq b > 0$. Set $r_0 = a$ and $r_1 = b$. Suppose that

$$\begin{aligned} r_0 &= r_1 q_1 + r_2, 0 \leq r_2 < r_1 \\ r_1 &= r_2 q_2 + r_3, 0 \leq r_3 < r_2 \\ &\dots \\ r_{n-2} &= r_{n-1} q_{n-1} + r_n, 0 \leq r_n < r_{n-1} \\ r_{n-1} &= r_n q_n. \end{aligned}$$

Then, $\gcd(a, b) = r_n =$ (the last non-zero remainder).

B.7 Co-prime integer.

Definition: A co-prime integer of N is a positive integer less than or equal to a number N which is also relatively prime to N , where 1 is counted as being relatively prime to all numbers.

B.8 Euler Totient Function.

Definition: The Euler totient function $\phi(N)$, also called totient function, is defined as the number of positive integers less than N , that are co-prime to (i.e., do not contain any factor other than 1 in common with) N , where 1 is counted as being co-prime to all numbers. The totient function $\phi(N)$ can be simply defined as the number of co-prime integers of N . E.g., $\phi(24) = 8$.

It is mathematically expressed as

$$\phi(N) = N \prod_{r|N} \left(1 - \frac{1}{r}\right)$$

APPENDIX C

Computation of layer 3 for $N = 4, 6, 10$ and 12 . Y_{k_1, k_2}^p /MRT coefficients of basic DFT coefficients are computed.

C.1 Layer 3 computations for $N = 4$

$$\begin{aligned}
 Y_{0,0}^0 &= C(0,0)+C(0,1)+C(1,0)+C(1,1) & Y_{2,0}^0 &= C(0,0)+C(0,1)-C(1,0)-C(1,1) \\
 Y_{0,2}^0 &= C(0,0)-C(0,1)+C(1,0)-C(1,1) & Y_{2,2}^0 &= C(0,0)-C(0,1)-C(1,0)+C(1,1) \\
 Y_{1,0}^0 &= D(0,0)+D(0,1) & Y_{1,0}^1 &= D(1,0)+D(1,1) \\
 Y_{1,2}^0 &= D(0,0)-D(0,1) & Y_{1,2}^1 &= D(1,0)-D(1,1) \\
 Y_{0,1}^0 &= E(0,0)+E(1,0) & Y_{0,1}^1 &= E(0,1)+E(1,1) \\
 Y_{2,1}^0 &= E(0,0)-E(1,0) & Y_{2,1}^1 &= E(0,1)-E(1,1) \\
 Y_{1,1}^0 &= F(0,0)-F(1,1) & Y_{1,1}^1 &= F(0,1)+F(1,0) \\
 Y_{3,1}^0 &= F(0,0)+F(1,1) & Y_{3,1}^1 &= F(0,1)-F(1,0)
 \end{aligned}$$

C.2 Layer 3 computations for $N = 6$

$$\begin{aligned}
 Y_{0,0}^0 &= C(0,0)+C(0,1)+C(0,2)+C(1,0)+C(1,1)+C(1,2)+C(2,0)+C(2,1)+C(2,2) \\
 Y_{3,0}^0 &= D(0,0)+D(0,1)+D(0,2)-D(1,0)-D(1,1)-D(1,2)+D(2,0)+D(2,1)+D(2,2) \\
 Y_{0,3}^0 &= E(0,0)-E(0,1)+E(0,2)+E(1,0)-E(1,1)+E(1,2)+E(2,0)-E(2,1)+E(2,2) \\
 Y_{3,3}^0 &= F(0,0)-F(0,1)+F(0,2)-F(1,0)+F(1,1)-F(1,2)+F(2,0)-F(2,1)+F(2,2) \\
 \\
 Y_{1,0}^0 &= D(0,0)+D(0,1)+D(0,2) & Y_{1,0}^1 &= D(1,0)+D(1,1)+D(1,2) \\
 Y_{1,0}^2 &= D(2,0)+D(2,1)+D(2,2) & Y_{2,0}^0 &= C(0,0)+C(0,1)+C(0,2) \\
 Y_{2,0}^1 &= -C(2,0)-C(2,1)-C(2,2) & Y_{2,0}^2 &= C(1,0)+C(1,1)+C(1,2) \\
 Y_{0,1}^0 &= E(0,0)+E(1,0)+E(2,0) & Y_{0,1}^1 &= E(0,1)+E(1,1)+E(2,1) \\
 Y_{0,1}^2 &= E(0,2)+E(1,2)+E(2,2) & Y_{1,1}^0 &= F(0,0)-F(1,2)-F(2,1) \\
 Y_{1,1}^1 &= F(0,1)+F(1,0)-F(2,2) & Y_{1,1}^2 &= F(0,2)+F(1,1)+F(2,0)
 \end{aligned}$$

$$\begin{aligned}
Y_{2,1}^0 &= E(0,0)-E(1,1)+E(2,2) & Y_{2,1}^1 &= E(0,1)-E(1,2)-E(2,0) \\
Y_{2,1}^2 &= E(0,2)+E(1,0)-E(2,1) & Y_{3,1}^0 &= F(0,0)-F(1,0)+F(2,0) \\
Y_{3,1}^1 &= F(0,1)-F(1,1)+F(2,1) & Y_{3,1}^2 &= F(0,2)-F(1,2)+F(2,2) \\
Y_{4,1}^0 &= E(0,0)+E(1,2)-E(2,1) & Y_{4,1}^1 &= E(0,1)-E(1,0)-E(2,2) \\
Y_{4,1}^2 &= E(0,2)-E(1,1)+E(2,0) & Y_{5,1}^0 &= F(0,0)+F(1,1)+F(2,2) \\
Y_{5,1}^1 &= F(0,1)+F(1,2)-F(2,0) & Y_{5,1}^2 &= F(0,2)-F(1,0)-F(2,1) \\
Y_{0,2}^0 &= C(0,0)+C(1,0)+C(2,0) & Y_{0,2}^1 &= -C(0,2)-C(1,2)-C(2,2) \\
Y_{0,2}^2 &= C(0,1)+C(1,1)+C(2,1) & Y_{1,2}^0 &= D(0,0)-D(1,1)+D(2,2) \\
Y_{1,2}^1 &= -D(0,2)+D(1,0)-D(2,1) & Y_{1,2}^2 &= D(0,1)-D(1,2)+D(2,0) \\
Y_{2,2}^0 &= C(0,0)+C(1,2)+C(2,1) & Y_{2,2}^1 &= -C(0,2)-C(1,1)-C(2,0) \\
Y_{2,2}^2 &= C(0,1)+C(1,0)+C(2,2) & Y_{3,2}^0 &= D(0,0)-D(1,0)+D(2,0) \\
Y_{3,2}^1 &= -D(0,2)+D(1,2)-D(2,2) & Y_{3,2}^2 &= D(0,1)-D(1,1)+D(2,1) \\
Y_{4,2}^0 &= C(0,0)+C(1,1)+C(2,2) & Y_{4,2}^1 &= -C(0,2)-C(1,0)-C(2,1) \\
Y_{4,2}^2 &= C(0,1)+C(1,2)+C(2,0) & Y_{5,2}^0 &= D(0,0)-D(1,2)+D(2,1) \\
Y_{5,2}^1 &= -D(0,2)+D(1,1)-D(2,0) & Y_{5,2}^2 &= D(0,1)-D(1,0)+D(2,2) \\
Y_{1,3}^0 &= F(0,0)-F(0,1)+F(0,2) & Y_{1,3}^1 &= F(1,0)-F(1,1)+F(1,2) \\
Y_{1,3}^2 &= F(2,0)-F(2,1)+F(2,2) & Y_{2,3}^0 &= E(0,0)-E(0,1)+E(0,2) \\
Y_{2,3}^1 &= -E(2,0)+E(2,1)-E(2,2) & Y_{2,3}^2 &= E(1,0)-E(1,1)+E(1,2)
\end{aligned}$$

C.3 Layer 3 computations for $N = 10$

$$\begin{aligned}
Y_{0,0}^0 &= C(0,0)+C(0,1)+C(0,2)+C(0,3)+C(0,4)+C(1,0)+C(1,1)+C(1,2)+C(1,3)+C(1,4)+C(2,0)+ \\
&\quad C(2,1)+C(2,2)+C(2,3)+C(2,4)+C(3,0)+C(3,1)+C(3,2)+C(3,3)+C(3,4)+C(4,0)+C(4,1)+C(4, \\
&\quad 2)+C(4,3)+C(4,4) \\
Y_{5,0}^0 &= D(0,0)+D(0,1)+D(0,2)+D(0,3)+D(0,4)-D(1,0)-D(1,1)-D(1,2)-D(1,3)-D(1,4)+D(2,0)+ \\
&\quad D(2,1)+D(2,2)+D(2,3)+D(2,4)-D(3,0)-D(3,1)-D(3,2)-D(3,3)-D(3,4)+D(4,0)+D(4,1)+ \\
&\quad D(4,2)+D(4,3)+D(4,4) \\
Y_{0,5}^0 &= E(0,0)-E(0,1)+E(0,2)-E(0,3)+E(0,4)+E(1,0)-E(1,1)+E(1,2)-E(1,3)+E(1,4)+E(2,0)-E(2,1) \\
&\quad +E(2,2)-E(2,3)+E(2,4)+E(3,0)-E(3,1)+E(3,2)-E(3,3)+E(3,4)+E(4,0)-E(4,1)+E(4,2)- \\
&\quad E(4,3)+E(4,4)
\end{aligned}$$

$$Y_{5,5}^0 = F(0,0)-F(0,1)+F(0,2)-F(0,3)+F(0,4)-F(1,0)+F(1,1)-F(1,2)+F(1,3)-F(1,4)+F(2,0)-F(2,1) \\ +F(2,2)-F(2,3)+F(2,4)-F(3,0)+F(3,1)-F(3,2)+F(3,3)-F(3,4)+F(4,0)-F(4,1)+F(4,2)- \\ F(4,3)+F(4,4)$$

$$Y_{1,0}^0 = D(0,0)+D(0,1)+D(0,2)+D(0,3)+D(0,4) \quad Y_{1,0}^1 = D(1,0)+D(1,1)+D(1,2)+D(1,3)+D(1,4)$$

$$Y_{1,0}^2 = D(2,0)+D(2,1)+D(2,2)+D(2,3)+D(2,4) \quad Y_{1,0}^3 = D(3,0)+D(3,1)+D(3,2)+D(3,3)+D(3,4)$$

$$Y_{1,0}^4 = D(4,0)+D(4,1)+D(4,2)+D(4,3)+D(4,4) \quad Y_{2,0}^0 = C(0,0)+C(0,1)+C(0,2)+C(0,3)+C(0,4)$$

$$Y_{2,0}^1 = -C(3,0)-C(3,1)-C(3,2)-C(3,3)-C(3,4) \quad Y_{2,0}^2 = C(1,0)+C(1,1)+C(1,2)+C(1,3)+C(1,4)$$

$$Y_{2,0}^3 = -C(4,0)-C(4,1)-C(4,2)-C(4,3)-C(4,4) \quad Y_{2,0}^4 = C(2,0)+C(2,1)+C(2,2)+C(2,3)+C(2,4)$$

$$Y_{0,1}^0 = E(0,0)+E(1,0)+E(2,0)+E(3,0)+E(4,0) \quad Y_{0,1}^1 = E(0,1)+E(1,1)+E(2,1)+E(3,1)+E(4,1)$$

$$Y_{0,1}^2 = E(0,2)+E(1,2)+E(2,2)+E(3,2)+E(4,2) \quad Y_{0,1}^3 = E(0,3)+E(1,3)+E(2,3)+E(3,3)+E(4,3)$$

$$Y_{0,1}^4 = E(0,4)+E(1,4)+E(2,4)+E(3,4)+E(4,4) \quad Y_{1,1}^0 = F(0,0)-F(1,4)-F(2,3)-F(3,2)-F(4,1)$$

$$Y_{1,1}^1 = F(0,1)+F(1,0)-F(2,4)-F(3,3)-F(4,2) \quad Y_{1,1}^2 = F(0,2)+F(1,1)+F(2,0)-F(3,4)-F(4,3)$$

$$Y_{1,1}^3 = F(0,3)+F(1,2)+F(2,1)+F(3,0)-F(4,4) \quad Y_{1,1}^4 = F(0,4)+F(1,3)+F(2,2)+F(3,1)+F(4,0)$$

$$Y_{2,1}^0 = E(0,0)-E(1,3)-E(2,1)+E(3,4)+E(4,2) \quad Y_{2,1}^1 = E(0,1)-E(1,4)-E(2,2)-E(3,0)+E(4,3)$$

$$Y_{2,1}^2 = E(0,2)+E(1,0)-E(2,3)-E(3,1)+E(4,4) \quad Y_{2,1}^3 = E(0,4)+E(1,2)+E(2,0)-E(3,3)-E(4,1)$$

$$Y_{2,1}^4 = E(0,4)+E(1,2)+E(2,0)-E(3,3)-E(4,1) \quad Y_{3,1}^0 = F(0,0)-F(1,2)+F(2,4)+F(3,1)-F(4,3)$$

$$Y_{3,1}^1 = F(0,1)-F(1,3)-F(2,0)+F(3,2)-F(4,4) \quad Y_{3,1}^2 = F(0,2)-F(1,4)-F(2,1)+F(3,3)+F(4,0)$$

$$Y_{3,1}^3 = F(0,3)+F(1,0)-F(2,2)+F(3,4)+F(4,1) \quad Y_{3,1}^4 = F(0,4)+F(1,1)-F(2,3)-F(3,0)+F(4,2)$$

$$Y_{4,1}^0 = E(0,0)-E(1,1)+E(2,2)-E(3,3)+E(4,4) \quad Y_{4,1}^1 = E(0,1)-E(1,2)+E(2,3)-E(3,4)-E(4,0)$$

$$Y_{4,1}^2 = E(0,2)-E(1,3)+E(2,4)+E(3,0)-E(4,1) \quad Y_{4,1}^3 = E(0,3)-E(1,4)-E(2,0)+E(3,1)-E(4,2)$$

$$Y_{4,1}^4 = E(0,4)+E(1,0)-E(2,1)+E(3,2)-E(4,3) \quad Y_{5,1}^0 = F(0,0)-F(1,0)+F(2,0)-F(3,0)+F(4,0)$$

$$Y_{5,1}^1 = F(0,1)-F(1,1)+F(2,1)-F(3,1)+F(4,1) \quad Y_{5,1}^2 = F(0,2)-F(1,2)+F(2,2)-F(3,2)+F(4,2)$$

$$Y_{5,1}^3 = F(0,3)-F(1,3)+F(2,3)-F(3,3)+F(4,3) \quad Y_{5,1}^4 = F(0,4)-F(1,4)+F(2,4)-F(3,4)+F(4,4)$$

$$Y_{6,1}^0 = E(0,0)+E(1,4)-E(2,3)+E(3,2)-E(4,1) \quad Y_{6,1}^1 = E(0,1)-E(1,0)-E(2,4)-E(3,3)+E(4,2)$$

$$Y_{6,1}^2 = E(0,2)-E(1,1)-E(2,0)+E(3,4)-E(4,3) \quad Y_{6,1}^3 = E(0,3)-E(1,2)+E(2,1)-E(3,0)-E(4,4)$$

$$Y_{6,1}^4 = E(0,4)-E(1,3)+E(2,2)-E(3,1)+E(4,0) \quad Y_{7,1}^0 = F(0,0)+F(1,3)-F(2,1)-F(3,4)+F(4,2)$$

$$Y_{7,1}^1 = F(0,1)+F(1,4)-F(2,2)+F(3,0)+F(4,3) \quad Y_{7,1}^2 = F(0,2)-F(1,0)-F(2,3)+F(3,1)+F(4,4)$$

$$\begin{aligned}
Y_{7,1}^3 &= F(0,3)-F(1,1)-F(2,4)+F(3,2)-F(4,0) & Y_{7,1}^4 &= F(0,4)-F(1,2)+F(2,0)+F(3,3)-F(4,1) \\
Y_{8,1}^0 &= E(0,0)+E(1,2)+E(2,4)-E(3,1)-E(4,3) & Y_{8,1}^1 &= E(0,1)+E(1,3)-E(2,0)-E(3,2)-E(4,4) \\
Y_{8,1}^2 &= E(0,2)+E(1,4)-E(2,1)-E(3,3)+E(4,0) & Y_{8,1}^3 &= E(0,3)-E(1,0)-E(2,2)-E(3,4)+E(4,1) \\
Y_{8,1}^4 &= E(0,4)-E(1,1)-E(2,3)+E(3,0)+E(4,2) & Y_{9,1}^0 &= F(0,0)+F(1,1)+F(2,2)+F(3,3)+F(4,4) \\
Y_{9,1}^1 &= F(0,1)+F(1,2)+F(2,3)+F(3,4)-F(4,0) & Y_{9,1}^2 &= F(0,2)+F(1,3)+F(2,4)-F(3,0)-F(4,1) \\
Y_{9,1}^3 &= F(0,3)+F(1,4)-F(2,0)-F(3,1)-F(4,2) & Y_{9,1}^4 &= F(0,4)-F(1,0)-F(2,1)-F(3,2)-F(4,3) \\
Y_{0,2}^0 &= C(0,0)+C(1,0)+C(2,0)+C(3,0)+C(4,0) & Y_{0,2}^1 &= -C(0,3)-C(1,3)-C(2,3)-C(3,3)-C(4,0) \\
Y_{0,2}^2 &= C(0,1)+C(1,1)+C(2,1)+C(3,1)+C(4,1) & Y_{0,2}^3 &= -C(0,4)-C(1,4)-C(2,4)-C(3,4)-C(4,4) \\
Y_{0,2}^4 &= C(0,2)+C(1,2)+C(2,2)+C(3,2)+C(4,2) & Y_{1,2}^0 &= D(0,0)-D(1,2)+D(2,4)-D(3,1)+D(4,3) \\
Y_{1,2}^1 &= -D(0,3)+D(1,0)-D(2,2)+D(3,4)-D(4,1) & Y_{1,2}^2 &= D(0,1)-D(1,3)+D(2,0)-D(3,2)+D(4,4) \\
Y_{1,2}^3 &= -D(0,4)+D(1,1)-D(2,3)+D(3,0)-D(4,2) & Y_{1,2}^4 &= D(0,2)-D(1,4)+D(2,1)-D(3,3)+D(4,0) \\
Y_{2,2}^0 &= C(0,0)+C(1,4)+C(2,3)+C(3,2)+C(4,1) & Y_{2,2}^1 &= -C(0,3)-C(1,2)-C(2,1)-C(3,0)-C(4,4) \\
Y_{2,2}^2 &= C(0,1)+C(1,0)+C(2,4)+C(3,3)+C(4,2) & Y_{2,2}^3 &= -C(0,4)-C(1,3)-C(2,2)-C(3,1)-C(4,0) \\
Y_{2,2}^4 &= C(0,2)+C(1,1)+C(2,0)+C(3,4)+C(4,3) & Y_{3,2}^0 &= D(0,0)-D(1,1)+D(2,2)-D(3,3)+D(4,4) \\
Y_{3,2}^1 &= -D(0,3)+D(1,4)-D(2,0)+D(3,1)-D(4,2) & Y_{3,2}^2 &= D(0,1)-D(1,2)+D(2,3)-D(3,4)+D(4,0) \\
Y_{3,2}^3 &= -D(0,4)+D(1,0)-D(2,1)+D(3,2)-D(4,3) & Y_{3,2}^4 &= D(0,2)-D(1,3)+D(2,4)-D(3,0)+D(4,1) \\
Y_{4,2}^0 &= C(0,0)+C(1,3)+C(2,1)+C(3,4)+C(4,2) & Y_{4,2}^1 &= -C(0,3)-C(1,1)-C(2,4)-C(3,2)-C(4,0) \\
Y_{4,2}^2 &= C(0,1)+C(1,4)+C(2,2)+C(3,0)+C(4,3) & Y_{4,2}^3 &= -C(0,4)-C(1,2)-C(2,0)-C(3,3)-C(4,1) \\
Y_{4,2}^4 &= C(0,2)+C(1,0)+C(2,3)+C(3,1)+C(4,4) & Y_{5,2}^0 &= D(0,0)-D(1,0)+D(2,0)-D(3,0)+D(4,0) \\
Y_{5,2}^1 &= -D(0,3)+D(1,3)-D(2,3)+D(3,3)-D(4,3) & Y_{5,2}^2 &= D(0,1)-D(1,1)+D(2,1)-D(3,1)+D(4,1) \\
Y_{5,2}^3 &= -D(0,4)+D(1,4)-D(2,4)+D(3,4)-D(4,4) & Y_{5,2}^4 &= D(0,2)-D(1,2)+D(2,2)-D(3,2)+D(4,2) \\
Y_{6,2}^0 &= C(0,0)+C(1,2)+C(2,4)+C(3,1)+C(4,3) & Y_{6,2}^1 &= -C(0,3)-C(1,0)-C(2,2)-C(3,4)-C(4,1) \\
Y_{6,2}^2 &= C(0,1)+C(1,3)+C(2,0)+C(3,2)+C(4,4) & Y_{6,2}^3 &= -C(0,4)-C(1,1)-C(2,3)-C(3,0)-C(4,2) \\
Y_{6,2}^4 &= C(0,2)+C(1,4)+C(2,1)+C(3,3)+C(4,0) & Y_{7,2}^0 &= D(0,0)-D(1,4)+D(2,3)-D(3,2)+D(4,1) \\
Y_{7,2}^1 &= -D(0,3)+D(1,2)-D(2,1)+D(3,0)-D(4,4) & Y_{7,2}^2 &= D(0,1)-D(1,0)+D(2,4)-D(3,3)+D(4,2) \\
Y_{7,2}^3 &= -D(0,4)+D(1,3)-D(2,2)+D(3,1)-D(4,0) & Y_{7,2}^4 &= D(0,2)-D(1,1)+D(2,0)-D(3,4)+D(4,3) \\
Y_{8,2}^0 &= C(0,0)+C(1,1)+C(2,2)+C(3,3)+C(4,4) & Y_{8,2}^1 &= -C(0,3)-C(1,4)-C(2,0)-C(3,1)-C(4,2) \\
Y_{8,2}^2 &= C(0,1)+C(1,2)+C(2,3)+C(3,4)+C(4,0) & Y_{8,2}^3 &= -C(0,4)-C(1,0)-C(2,1)-C(3,2)-C(4,3)
\end{aligned}$$

$$\begin{aligned}
Y_{8,2}^4 &= C(0,2)+C(1,3)+C(2,4)+C(3,0)+C(4,1) & Y_{9,2}^0 &= D(0,0)-D(1,3)+D(2,1)-D(3,4)+D(4,2) \\
Y_{9,2}^1 &= -D(0,3)+D(1,1)-D(2,4)+D(3,2)-D(4,0) & Y_{9,2}^2 &= D(0,1)-D(1,4)+D(2,2)-D(3,0)+D(4,3) \\
Y_{9,2}^3 &= -D(0,4)+D(1,2)-D(2,0)+D(3,3)-D(4,1) & Y_{9,2}^4 &= D(0,2)-D(1,0)+D(2,3)-D(3,1)+D(4,4) \\
Y_{1,5}^0 &= F(0,0)+F(0,1)+F(0,2)+F(0,3)+F(0,4) & Y_{1,5}^1 &= F(1,0)+F(1,1)+F(1,2)+F(1,3)+F(1,4) \\
Y_{1,5}^2 &= F(2,0)+F(2,1)+F(2,2)+F(2,3)+F(2,4) & Y_{1,5}^3 &= F(3,0)+F(3,1)+F(3,2)+F(3,3)+F(3,4) \\
Y_{1,5}^4 &= F(4,0)+F(4,1)+F(4,2)+F(4,3)+F(4,4) & Y_{2,5}^0 &= E(0,0)-E(0,1)+E(0,2)-E(0,3)+E(0,4) \\
Y_{2,5}^1 &= -E(3,0)+E(3,1)-E(3,2)+E(3,3)-E(3,4) & Y_{2,5}^2 &= E(1,0)-E(1,1)+E(1,2)-E(1,3)+E(1,4) \\
Y_{2,5}^3 &= -E(4,0)+E(4,1)-E(4,2)+E(4,3)-E(4,4) & Y_{2,5}^4 &= E(2,0)-E(2,1)+E(2,2)-E(2,3)+E(2,4)
\end{aligned}$$

C.4 Layer 3 computations for $N = 12$

$$\begin{aligned}
Y_{0,0}^0 &= C(0,0)+C(0,1)+C(0,2)+C(0,3)+C(0,4)+C(0,5)+C(1,0)+C(1,1)+C(1,2)+C(1,3)+C(1,4)+ \\
&\quad C(1,5)+C(2,0)+C(2,1)+C(2,2)+C(2,3)+C(2,4)+C(2,5)+C(3,0)+C(3,1)+C(3,2)+C(3,3)+C(3, \\
&\quad 4)+C(3,5)+C(4,0)+C(4,1)+C(4,2)+C(4,3)+C(4,4)+C(4,5)+C(5,0)+C(5,1)+C(5,2)+C(5,3)+ \\
&\quad C(5,4)+C(5,5) \\
Y_{6,0}^0 &= C(0,0)+C(0,1)+C(0,2)+C(0,3)+C(0,4)+C(0,5)-C(1,0)-C(1,1)-C(1,2)-C(1,3)-C(1,4)-C(1,5) \\
&\quad +C(2,0)+C(2,1)+C(2,2)+C(2,3)+C(2,4)+C(2,5)-C(3,0)-C(3,1)-C(3,2)-C(3,3)-C(3,4)-C(3,5) \\
&\quad +C(4,0)+C(4,1)+C(4,2)+C(4,3)+C(4,4)+C(4,5)-C(5,0)-C(5,1)-C(5,2)-C(5,3)-C(5,4)-C(5,5) \\
Y_{0,6}^0 &= C(0,0)+C(1,0)+C(2,0)+C(3,0)+C(4,0)+C(5,0)-C(0,1)-C(1,1)-C(2,1)-C(3,1)-C(4,1)- \\
&\quad C(5,1)+C(0,2)+C(1,2)+C(2,2)+C(3,2)+C(4,2)+C(5,2)-C(0,3)-C(1,3)-C(2,3)-C(3,3)-C(4,3)- \\
&\quad C(5,3)+C(0,4)+C(1,4)+C(2,4)+C(3,4)+C(4,4)+C(5,4)-C(0,5)-C(1,5)-C(2,5)-C(3,5)- \\
&\quad C(4,5)-C(5,5) \\
Y_{6,6}^0 &= C(0,0)-C(0,1)+C(0,2)-C(0,3)+C(0,4)-C(0,5)+C(1,0)-C(1,1)+C(1,2)-C(1,3)+C(1,4)-C(1,5)+ \\
&\quad C(2,0)-C(2,1)+C(2,2)-C(2,3)+C(2,4)-C(2,5)+C(3,0)-C(3,1)+C(3,2)-C(3,3)+C(3,4)-C(3,5)+ \\
&\quad C(4,0)-C(4,1)+C(4,2)-C(4,3)+C(4,4)-C(4,5)+C(5,0)-C(5,1)+C(5,2)-C(5,3)+C(5,4)-C(5,5) \\
Y_{1,0}^0 &= D(0,0)+D(0,1)+D(0,2)+D(0,3)+D(0,4)+D(0,5) \\
Y_{1,0}^1 &= D(1,0)+D(1,1)+D(1,2)+D(1,3)+D(1,4)+D(1,5) \\
Y_{1,0}^2 &= D(2,0)+D(2,1)+D(2,2)+D(2,3)+D(2,4)+D(2,5) \\
Y_{1,0}^3 &= D(3,0)+D(3,1)+D(3,2)+D(3,3)+D(3,4)+D(3,5) \\
Y_{1,0}^4 &= D(4,0)+D(4,1)+D(4,2)+D(4,3)+D(4,4)+D(4,5) \\
Y_{1,0}^5 &= D(5,0)+D(5,1)+D(5,2)+D(5,3)+D(5,4)+D(5,5)
\end{aligned}$$

$$Y_{2,0}^0 = C(0,0)+C(0,1)+C(0,2)+C(0,3)+C(0,4)+C(0,5)-C(2,0)-C(2,1)-C(2,2)-C(2,3)-C(2,4)-C(2,5)$$

$$Y_{2,0}^2 = C(1,0)+C(1,1)+C(1,2)+C(1,3)+C(1,4)+C(1,5)-C(3,0)-C(3,1)-C(3,2)-C(3,3)-C(3,4)-C(3,5)$$

$$Y_{2,0}^4 = C(2,0)+C(2,1)+C(2,2)+C(2,3)+C(2,4)+C(2,5)-C(4,0)-C(4,1)-C(4,2)-C(4,3)-C(4,4)-C(4,5)$$

$$Y_{3,0}^0 = D(0,0)+D(0,1)+D(0,2)+D(0,3)+D(0,4)+D(0,5)- D(2,0)-D(2,1)-D(2,2)-D(2,3)-D(2,4)-$$

$$D(2,5)+ D(4,0)+D(4,1)+D(4,2)+D(4,3)+D(4,4)+D(4,5)$$

$$Y_{3,0}^3 = D(1,0)+D(1,1)+D(1,2)+D(1,3)+D(1,4)+D(1,5)- D(3,0)-D(3,1)-D(3,2)-D(3,3)-D(3,4)-$$

$$D(3,5)+ D(5,0)+D(5,1)+D(5,2)+D(5,3)+D(5,4)+D(5,5)$$

$$Y_{4,0}^0 = C(0,0)+C(0,1)+C(0,2)+C(0,3)+C(0,4)+C(0,5)+C(3,0)+C(3,1)+C(3,2)+C(3,3)+C(3,4)+C(3,5)$$

$$Y_{4,0}^2 = -C(2,0)-C(2,1)-C(2,2)-C(2,3)-C(2,4)-C(2,5)- C(5,0)-C(5,1)-C(5,2)-C(5,3)-C(5,4)-C(5,5)$$

$$Y_{4,0}^4 = C(1,0)+C(1,1)+C(1,2)+C(1,3)+C(1,4)+C(1,5)+C(4,0)+C(4,1)+C(4,2)+C(4,3)+C(4,4)+C(4,5)$$

$$Y_{0,1}^0 = E(0,0)+E(1,0)+E(2,0)+E(3,0)+E(4,0)+E(5,0)$$

$$Y_{0,1}^1 = E(0,1)+E(1,1)+E(2,1)+E(3,1)+E(4,1)+E(5,1)$$

$$Y_{0,1}^2 = E(0,2)+E(1,2)+E(2,2)+E(3,2)+E(4,2)+E(5,2)$$

$$Y_{0,1}^3 = E(0,3)+E(1,3)+E(2,3)+E(3,3)+E(4,3)+E(5,3)$$

$$Y_{0,1}^4 = E(0,4)+E(1,4)+E(2,4)+E(3,4)+E(4,4)+E(5,4)$$

$$Y_{0,1}^5 = E(0,5)+E(1,5)+E(2,5)+E(3,5)+E(4,5)+E(5,5)$$

$$Y_{1,1}^0 = F(0,0)-F(1,5)-F(2,4)-F(3,3)-F(4,2)-F(5,1)$$

$$Y_{1,1}^1 = F(0,1)+F(1,0)-F(2,5)-F(3,4)-F(4,3)-F(5,2)$$

$$Y_{1,1}^2 = F(0,2)+F(1,1)+F(2,0)-F(3,5)-F(4,4)-F(5,3)$$

$$Y_{1,1}^3 = F(0,3)+F(1,2)+F(2,1)+F(3,0)-F(4,5)-F(5,4)$$

$$Y_{1,1}^4 = F(0,4)+F(1,3)+F(2,2)+F(3,1)+F(4,0)-F(5,5)$$

$$Y_{1,1}^5 = F(0,5)+F(1,4)+F(2,3)+F(3,2)+F(4,1)+F(5,0)$$

$$Y_{2,1}^0 = E(0,0)-E(1,4)-E(2,2)-E(3,0)+E(4,4)+E(5,2)$$

$$Y_{2,1}^1 = E(0,1)-E(1,5)-E(2,3)-E(3,1)+E(4,5)+E(5,3)$$

$$Y_{2,1}^2 = E(0,2)+E(1,0)-E(2,4)-E(3,2)-E(4,0)+E(5,4)$$

$$Y_{2,1}^3 = E(0,3)+E(1,1)-E(2,5)-E(3,3)-E(4,1)+E(5,5)$$

$$Y_{2,1}^4 = E(0,4)+E(1,2)+E(2,0)-E(3,4)-E(4,2)-E(5,0)$$

$$Y_{2,1}^5 = E(0,5)+E(1,3)+E(2,1)-E(3,5)-E(4,3)-E(5,1)$$

$$\begin{aligned}
Y_{3,1}^0 &= F(0,0)-F(1,3)-F(2,0)+F(3,3)+F(4,0)-F(5,3) \\
Y_{3,1}^1 &= F(0,1)-F(1,4)-F(2,1)+F(3,4)+F(4,1)-F(5,4) \\
Y_{3,1}^2 &= F(0,2)-F(1,5)-F(2,2)+F(3,5)+F(4,2)-F(5,5) \\
Y_{3,1}^3 &= F(0,3)+F(1,0)-F(2,3)-F(3,0)+F(4,3)+F(5,0) \\
Y_{3,1}^4 &= F(0,4)+F(1,1)-F(2,4)-F(3,1)+F(4,4)+F(5,1) \\
Y_{3,1}^5 &= F(0,5)+F(1,2)-F(2,5)-F(3,2)+F(4,5)+F(5,2) \\
Y_{4,1}^0 &= E(0,0)-E(1,2)+E(2,4)+E(3,0)-E(4,2)+E(5,4) \\
Y_{4,1}^1 &= E(0,1)-E(1,3)+E(2,5)+E(3,1)-E(4,3)+E(5,5) \\
Y_{4,1}^2 &= E(0,2)-E(1,4)-E(2,0)+E(3,2)-E(4,4)-E(5,0) \\
Y_{4,1}^3 &= E(0,3)-E(1,5)-E(2,1)+E(3,3)-E(4,5)-E(5,1) \\
Y_{4,1}^4 &= E(0,4)+E(1,0)-E(2,2)+E(3,4)+E(4,0)-E(5,2) \\
Y_{4,1}^5 &= E(0,5)+E(1,1)-E(2,3)+E(3,5)+E(4,1)-E(5,3) \\
Y_{5,1}^0 &= F(0,0)-F(1,1)+F(2,2)-F(3,3)+F(4,4)-F(5,5) \\
Y_{5,1}^1 &= F(0,1)-F(1,2)+F(2,3)-F(3,4)+F(4,5)+F(5,0) \\
Y_{5,1}^2 &= F(0,2)-F(1,3)+F(2,4)-F(3,5)-F(4,0)+F(5,1) \\
Y_{5,1}^3 &= F(0,3)-F(1,4)+F(2,5)+F(3,0)-F(4,1)+F(5,2) \\
Y_{5,1}^4 &= F(0,4)-F(1,5)-F(2,0)+F(3,1)-F(4,2)+F(5,3) \\
Y_{5,1}^5 &= F(0,5)+F(1,0)-F(2,1)+F(3,2)-F(4,3)+F(5,4) \\
Y_{6,1}^0 &= E(0,0)-E(1,0)+E(2,0)-E(3,0)+E(4,0)-E(5,0) \\
Y_{6,1}^1 &= -E(0,1)+E(1,1)-E(2,1)+E(3,1)-E(4,1)+E(5,1) \\
Y_{6,1}^2 &= E(0,2)-E(1,2)+E(2,2)-E(3,2)+E(4,2)-E(5,2) \\
Y_{6,1}^3 &= -E(0,3)+E(1,3)-E(2,3)+E(3,3)-E(4,3)+E(5,3) \\
Y_{6,1}^4 &= E(0,4)-E(1,4)+E(2,4)-E(3,4)+E(4,4)-E(5,4) \\
Y_{6,1}^5 &= -E(0,5)+E(1,5)-E(2,5)+E(3,5)-E(4,5)+E(5,5) \\
Y_{7,1}^0 &= F(0,0)+F(1,5)-F(2,4)+F(3,3)-F(4,2)+F(5,1) \\
Y_{7,1}^1 &= F(0,1)-F(1,0)-F(2,5)+F(3,4)-F(4,3)+F(5,2) \\
Y_{7,1}^2 &= F(0,2)-F(1,1)+F(2,0)+F(3,5)-F(4,4)+F(5,3) \\
Y_{7,1}^3 &= F(0,3)-F(1,2)+F(2,1)-F(3,0)-F(4,5)+F(5,4)
\end{aligned}$$

$$Y_{7,1}^4 = F(0,4)-F(1,3)+F(2,2)-F(3,1)+F(4,0)+F(5,5)$$

$$Y_{7,1}^5 = F(0,5)-F(1,4)+F(2,3)-F(3,2)+F(4,1)-F(5,0)$$

$$Y_{8,1}^0 = E(0,0)+E(1,4)-E(2,2)+E(3,0)+E(4,4)-E(5,2)$$

$$Y_{8,1}^1 = E(0,1)+E(1,5)-E(2,3)+E(3,1)+E(4,5)-E(5,3)$$

$$Y_{8,1}^2 = E(0,2)-E(1,0)-E(2,4)+E(3,2)-E(4,0)-E(5,4)$$

$$Y_{8,1}^3 = E(0,3)-E(1,1)-E(2,5)+E(3,3)-E(4,1)-E(5,5)$$

$$Y_{8,1}^4 = E(0,4)-E(1,2)+E(2,0)+E(3,4)-E(4,2)+E(5,0)$$

$$Y_{8,1}^5 = E(0,5)-E(1,3)+E(2,1)+E(3,5)-E(4,3)+E(5,1)$$

$$Y_{9,1}^0 = F(0,0)+F(1,3)-F(2,0)-F(3,3)+F(4,0)+F(5,3)$$

$$Y_{9,1}^1 = F(0,1)+F(1,4)-F(2,1)-F(3,4)+F(4,1)+F(5,4)$$

$$Y_{9,1}^2 = F(0,2)+F(1,5)-F(2,2)-F(3,5)+F(4,2)+F(5,5)$$

$$Y_{9,1}^3 = F(0,3)-F(1,0)-F(2,3)+F(3,0)+F(4,3)-F(5,0)$$

$$Y_{9,1}^4 = F(0,4)-F(1,1)-F(2,4)+F(3,1)+F(4,4)-F(5,1)$$

$$Y_{9,1}^5 = F(0,5)-F(1,2)-F(2,5)+F(3,2)+F(4,5)-F(5,2)$$

$$Y_{10,1}^0 = E(0,0)+E(1,2)+E(2,4)-E(3,0)-E(4,2)-E(5,4)$$

$$Y_{10,1}^1 = E(0,1)+E(1,3)+E(2,5)-E(3,1)-E(4,3)-E(5,5)$$

$$Y_{10,1}^2 = E(0,2)+E(1,4)-E(2,0)-E(3,2)-E(4,4)+E(5,0)$$

$$Y_{10,1}^3 = E(0,3)+E(1,5)-E(2,1)-E(3,3)-E(4,5)+E(5,1)$$

$$Y_{10,1}^4 = E(0,4)-E(1,0)-E(2,2)-E(3,4)+E(4,0)+E(5,2)$$

$$Y_{10,1}^5 = E(0,5)-E(1,1)-E(2,3)-E(3,5)+E(4,1)+E(5,3)$$

$$Y_{11,1}^0 = F(0,0)+F(1,1)+F(2,2)+F(3,3)+F(4,4)+F(5,5)$$

$$Y_{11,1}^1 = F(0,1)+F(1,2)+F(2,3)+F(3,4)+F(4,5)-F(5,0)$$

$$Y_{11,1}^2 = F(0,2)+F(1,3)+F(2,4)+F(3,5)-F(4,0)-F(5,1)$$

$$Y_{11,1}^3 = F(0,3)+F(1,4)+F(2,5)-F(3,0)-F(4,1)-F(5,2)$$

$$Y_{11,1}^4 = F(0,4)+F(1,5)-F(2,0)-F(3,1)-F(4,2)-F(5,3)$$

$$Y_{11,1}^5 = F(0,5)-F(1,0)-F(2,1)-F(3,2)-F(4,3)-F(5,4)$$

$$Y_{0,2}^0 = C(0,0)+C(1,0)+C(2,0)+C(3,0)+C(4,0)+C(5,0)-C(0,3)-C(1,3)-C(2,3)-C(3,3)-C(4,3)-C(5,3)$$

$$Y_{0,2}^2 = C(0,1)+C(1,1)+C(2,1)+C(3,1)+C(4,1)+C(5,1)-C(0,4)-C(1,4)-C(2,4)-C(3,4)-C(4,4)-C(5,4)$$

$$Y_{0,2}^4 = C(0,2)+C(1,2)+C(2,2)+C(3,2)+C(4,2)+C(5,2) -C(0,5)-C(1,5)-C(2,5)-C(3,5)-C(4,5)-C(5,5)$$

$$Y_{1,2}^0 = D(0,0)-D(0,3)-D(2,2)+D(2,5)-D(4,1)+D(4,4)$$

$$Y_{1,2}^1 = D(1,0)-D(1,3)-D(3,2)+D(3,5)-D(5,1)+D(5,4)$$

$$Y_{1,2}^2 = D(0,1)-D(0,4)+D(2,0)-D(2,3)-D(4,2)+D(4,5)$$

$$Y_{1,2}^3 = D(1,1)-D(1,4)+D(3,0)-D(3,3)-D(5,2)+D(5,5)$$

$$Y_{1,2}^4 = D(0,2)-D(0,5)+D(2,1)-D(2,4)+D(4,0)-D(4,3)$$

$$Y_{1,2}^5 = D(1,2)-D(1,5)+D(3,1)-D(3,4)+D(5,0)-D(5,3)$$

$$Y_{2,2}^0 = C(0,0)+C(1,5)+C(2,4)+C(3,3)+C(4,2)+C(5,1)-C(0,3)-C(1,2)-C(2,1)-C(3,0)-C(4,5)-C(5,4)$$

$$Y_{2,2}^2 = C(0,1)+C(1,0)+C(2,5)+C(3,4)+C(4,3)+C(5,2)-C(0,4)-C(1,3)-C(2,2)-C(3,1)-C(4,0)-C(5,5)$$

$$Y_{2,2}^4 = C(0,2)+C(1,1)+C(2,0)+C(3,5)+C(4,4)+C(5,3)-C(0,5)-C(1,4)-C(2,3)-C(3,2)-C(4,1)-C(5,0)$$

$$Y_{3,2}^0 = D(0,0)-D(0,3)-D(2,0)+D(2,3)+D(4,0)-D(4,3)$$

$$Y_{3,2}^1 = -D(1,2)+D(1,5)+D(3,2)-D(3,5)-D(5,2)+D(5,5)$$

$$Y_{3,2}^2 = D(0,1)-D(0,4)-D(2,1)+D(2,4)+D(4,1)-D(4,4)$$

$$Y_{3,2}^3 = D(1,0)-D(1,3)-D(3,0)+D(3,3)+D(5,0)-D(5,3)$$

$$Y_{3,2}^4 = D(0,2)-D(0,5)-D(2,2)+D(2,5)+D(4,2)-D(4,5)$$

$$Y_{3,2}^5 = D(1,1)-D(1,4)-D(3,1)+D(3,4)+D(5,1)-D(5,4)$$

$$Y_{4,2}^0 = C(0,0)+C(1,4)+C(2,2)+C(3,0)+C(4,4)+C(5,2)-C(0,3)-C(1,1)-C(2,5)-C(3,3)-C(4,1)-C(5,5)$$

$$Y_{4,2}^2 = C(0,1)+C(1,5)+C(2,3)+C(3,1)+C(4,5)+C(5,3)-C(0,4)-C(1,2)-C(2,0)-C(3,4)-C(4,2)-C(5,0)$$

$$Y_{4,2}^4 = C(0,2)+C(1,0)+C(2,4)+C(3,2)+C(4,0)+C(5,4)-C(0,5)-C(1,3)-C(2,1)-C(3,5)-C(4,3)-C(5,1)$$

$$Y_{5,2}^0 = D(0,0)-D(0,3)+D(2,1)-D(2,4)+D(4,2)-D(4,5)$$

$$Y_{5,2}^1 = -D(1,1)+D(1,4)-D(3,2)+D(3,5)+D(5,0)-D(5,3)$$

$$Y_{5,2}^2 = D(0,1)-D(0,4)+D(2,2)-D(2,5)-D(4,0)+D(4,3)$$

$$Y_{5,2}^3 = -D(1,2)+D(1,5)+D(3,0)-D(3,3)+D(5,1)-D(5,4)$$

$$Y_{5,2}^4 = D(0,2)-D(0,5)-D(2,0)+D(2,3)-D(4,1)+D(4,4)$$

$$Y_{5,2}^5 = D(1,0)-D(1,3)+D(3,1)-D(3,4)+D(5,2)-D(5,5)$$

$$Y_{6,2}^0 = C(0,0)-C(1,0)+C(2,0)-C(3,0)+C(4,0)-C(5,0)-C(0,3)+C(1,3)-C(2,3)+C(3,3)-C(4,3)+C(5,3)$$

$$Y_{6,2}^2 = C(0,1)-C(1,1)+C(2,1)-C(3,1)+C(4,1)-C(5,1)-C(0,4)+C(1,4)-C(2,4)+C(3,4)-C(4,4)+C(5,4)$$

$$Y_{6,2}^4 = C(0,2)-C(1,2)+C(2,2)-C(3,2)+C(4,2)-C(5,2)-C(0,5)+C(1,5)-C(2,5)+C(3,5)-C(4,5)+C(5,5)$$

$$Y_{8,2}^0 = C(0,0)+C(1,2)+C(2,4)+C(3,0)+C(4,2)+C(5,4)-C(0,3)-C(1,5)-C(2,1)-C(3,3)-C(4,3)-C(5,1)$$

$$Y_{8,2}^2 = C(0,1)+C(1,3)+C(2,5)+C(3,1)+C(4,3)+C(5,5)-C(0,4)-C(1,0)-C(2,2)-C(3,4)-C(4,0)-C(5,2)$$

$$Y_{8,2}^4 = C(0,2)+C(1,4)+C(2,0)+C(3,2)+C(4,4)+C(5,0)-C(0,5)-C(1,1)-C(2,3)-C(3,5)-C(4,1)-C(5,3)$$

$$Y_{10,2}^0 = C(0,0)+C(1,1)+C(2,2)+C(3,3)+C(4,4)+C(5,5)-C(0,3)-C(1,4)-C(2,5)-C(3,0)-C(4,1)-C(5,2)$$

$$Y_{10,2}^2 = C(0,1)+C(1,2)+C(2,3)+C(3,4)+C(4,5)+C(5,0)-C(0,4)-C(1,5)-C(2,0)-C(3,1)-C(4,2)-C(5,3)$$

$$Y_{10,2}^4 = C(0,2)+C(1,3)+C(2,4)+C(3,5)+C(4,0)+C(5,1)-C(0,5)-C(1,0)-C(2,1)-C(3,2)-C(4,3)-C(5,4)$$

$$Y_{0,3}^0 = E(0,0)+E(1,0)+E(2,0)+E(3,0)+E(4,0)+E(5,0)-E(0,2)-E(1,2)-E(2,2)-E(3,2)-E(4,2)-E(5,2) + \\ E(0,4)+E(1,4)+E(2,4)+E(3,4)+E(4,4)+E(5,4)$$

$$Y_{0,3}^3 = E(0,1)+E(1,1)+E(2,1)+E(3,1)+E(4,1)+E(5,1)-E(0,3)-E(1,3)-E(2,3)-E(3,3)-E(4,3)-E(5,3) + \\ E(0,5)+E(1,5)+E(2,5)+E(3,5)+E(4,5)+E(5,5)$$

$$Y_{1,3}^0 = F(0,0)-F(0,2)+F(0,4)-F(3,1)+F(3,3)-F(3,5)$$

$$Y_{1,3}^1 = F(1,0)-F(1,2)+F(1,4)-F(4,1)+F(4,3)-F(4,5)$$

$$Y_{1,3}^2 = F(2,0)-F(2,2)+F(2,4)-F(5,1)+F(5,3)-F(5,5)$$

$$Y_{1,3}^3 = F(0,1)-F(0,3)+F(0,5)+F(3,0)-F(3,2)+F(3,4)$$

$$Y_{1,3}^4 = F(1,1)-F(1,3)+F(1,5)+F(4,0)-F(4,2)+F(4,4)$$

$$Y_{1,3}^5 = F(2,1)-F(2,3)+F(2,5)+F(5,0)-F(5,2)+F(5,4)$$

$$Y_{2,3}^0 = E(0,0)-E(0,2)+E(0,4)-E(3,0)+E(3,2)-E(3,4)$$

$$Y_{2,3}^1 = -E(2,1)+E(2,3)-E(2,5)+E(5,1)-E(5,3)+E(5,5)$$

$$Y_{2,3}^2 = E(1,0)-E(1,2)+E(1,4)-E(4,0)+E(4,2)-E(4,4)$$

$$Y_{2,3}^3 = E(0,1)-E(0,3)+E(0,5)-E(3,1)+E(3,3)-E(3,5)$$

$$Y_{2,3}^4 = E(2,0)-E(2,2)+E(2,4)-E(5,0)+E(5,2)-E(5,4)$$

$$Y_{2,3}^5 = E(1,1)-E(1,3)+E(1,5)-E(4,1)+E(4,3)-E(4,5)$$

$$Y_{3,3}^0 = F(0,0)-F(0,2)+F(0,4)-F(1,1)+F(1,3)-F(1,5)-(F(2,0)-F(2,2)+F(2,4)-F(3,1)+F(3,3)- \\ F(3,5))+F(4,0)-F(4,2)+F(4,4)-F(5,1)+F(5,3)-F(5,5)$$

$$Y_{3,3}^3 = F(0,1)-F(0,3)+F(0,5)+F(1,0)-F(1,2)+F(1,4)-(F(2,1)-F(2,3)+F(2,5)+F(3,0)- \\ F(3,2)+F(3,4))+F(4,1)-F(4,3)+F(4,5)+F(5,0)-F(5,2)+F(5,4)$$

$$Y_{4,3}^0 = E(0,0)-E(0,2)+E(0,4)+E(3,0)-E(3,2)+E(3,4)$$

$$Y_{4,3}^1 = -E(1,1)+E(1,3)-E(1,5)-E(4,1)+E(4,3)-E(4,5)$$

$$Y_{4,3}^2 = -E(2,0)-E(2,2)-E(2,4)-E(5,0)+E(5,2)-E(5,4)$$

$$Y_{4,3}^3 = E(0,1)-E(0,3)+E(0,5)+E(3,1)-E(3,3)+E(3,5)$$

$$Y_{4,3}^4 = E(1,0)-E(1,2)+E(1,4)+E(4,0)-E(4,2)+E(4,4)$$

$$Y_{4,3}^5 = -E(2,1)+E(2,3)-E(2,5)-E(5,1)+E(5,3)-E(5,5)$$

$$Y_{6,3}^0 = E(0,0)-E(1,0)+E(2,0)-E(3,0)+E(4,0)-E(5,0)-(E(0,2)-E(1,2)+E(2,2)-E(3,2)+E(4,2)-E(5,2))+$$

$$E(0,4)-E(1,4)+E(2,4)-E(3,4)+E(4,4)-E(5,4)$$

$$Y_{6,3}^3 = E(0,1)-E(1,1)+E(2,1)-E(3,1)+E(4,1)-E(5,1)-(E(0,3)-E(1,3)+E(2,3)-E(3,3)+E(4,3)-E(5,3))+$$

$$E(0,5)-E(1,5)+E(2,5)-E(3,5)+E(4,5)-E(5,5)$$

$$Y_{7,3}^0 = F(0,0)-F(0,2)+F(0,4)+F(3,1)-F(3,3)+F(3,5)$$

$$Y_{7,3}^1 = -F(1,0)+F(1,2)-F(1,4)-F(4,1)+F(4,3)-F(4,5)$$

$$Y_{7,3}^2 = F(2,0)-F(2,2)+F(2,4)+F(5,1)-F(5,3)+F(5,5)$$

$$Y_{7,3}^3 = F(0,1)-F(0,3)+F(0,5)-F(3,0)+F(3,2)-F(3,4)$$

$$Y_{7,3}^4 = -F(1,1)+F(1,3)-F(1,5)+F(4,0)-F(4,2)+F(4,4)$$

$$Y_{7,3}^5 = F(2,1)-F(2,3)+F(2,5)-F(5,0)+F(5,2)-F(5,4)$$

$$Y_{9,3}^0 = F(0,0)-F(0,2)+F(0,4)+F(1,1)-F(1,3)+F(1,5)-(F(2,0)-F(2,2)+F(2,4)+F(3,1)-F(3,3)+F(3,5))+$$

$$F(4,0)-F(4,2)+F(4,4)+F(5,1)-F(5,3)+F(5,5)$$

$$Y_{9,3}^3 = F(0,1)-F(0,3)+F(0,5)-F(1,0)+F(1,2)-F(1,4)-(F(2,1)-F(2,3)+F(2,5)-F(3,0)+F(3,2)-F(3,4))+$$

$$F(4,1)-F(4,3)+F(4,5)-F(5,0)+F(5,2)-F(5,4)$$

$$Y_{0,4}^0 = C(0,0)+C(1,0)+C(2,0)+C(3,0)+C(4,0)+C(5,0)+C(0,3)+C(1,3)+C(2,3)+C(3,3)+C(4,3)+C(5,3)$$

$$Y_{0,4}^2 = -C(0,2)-C(1,2)-C(2,2)-C(3,2)-C(4,2)-C(5,2)-C(0,5)-C(1,5)-C(2,5)-C(3,5)-C(4,5)-C(5,5)$$

$$Y_{0,4}^4 = C(0,1)+C(1,1)+C(2,1)+C(3,1)+C(4,1)+C(5,1)+C(0,4)+C(1,4)+C(2,4)+C(3,4)+C(4,4)+C(5,4)$$

$$Y_{1,4}^0 = D(0,0)+D(0,3)-D(2,1)-D(2,4)+D(4,2)+D(4,5)$$

$$Y_{1,4}^1 = D(1,0)+D(1,3)-D(3,1)-D(3,4)+D(5,2)+D(5,5)$$

$$Y_{1,4}^2 = -D(0,2)-D(0,5)+D(2,0)+D(2,3)-D(4,1)-D(4,4)$$

$$Y_{1,4}^3 = -D(1,2)-D(1,5)+D(3,0)+D(3,3)-D(5,1)-D(5,4)$$

$$Y_{1,4}^4 = D(0,1)+D(0,4)-D(2,2)-D(2,5)+D(4,0)+D(4,3)$$

$$Y_{1,4}^5 = D(1,1)+D(1,4)-D(3,2)-D(3,5)+D(5,0)+D(5,3)$$

$$Y_{2,4}^0 = C(0,0)+C(0,3)+C(2,2)+C(2,5)+C(4,1)+C(4,4)-C(1,1)-C(1,4)-C(3,0)-C(3,3)-C(5,2)-C(5,5)$$

$$Y_{2,4}^2 = -C(0,2)-C(0,5)-C(2,1)-C(2,4)-C(4,0)-C(4,3)+C(1,0)+C(1,3)+C(3,2)+C(3,5)+C(5,1)+C(5,4)$$

$$Y_{2,4}^4 = C(0,1)+C(0,4)+C(2,0)+C(2,3)+C(4,2)+C(4,5)-C(1,2)-C(1,5)-C(3,1)-C(3,4)-C(5,0)-C(5,3)$$

$$Y_{3,4}^0 = D(0,0)+D(0,3)-D(2,0)-D(2,3)+D(4,0)+D(4,3)$$

$$Y_{3,4}^1 = -D(1,1)-D(1,4)+D(3,1)+D(3,4)-D(5,1)-D(5,4)$$

$$Y_{3,4}^2 = -D(0,2)-D(0,5)+D(2,2)+D(2,5)-D(4,2)-D(4,5)$$

$$Y_{3,4}^3 = D(1,0)+D(1,3)-D(3,0)-D(3,3)+D(5,0)+D(5,3)$$

$$Y_{3,4}^4 = D(0,1)+D(0,4)-D(2,1)-D(2,4)+D(4,1)+D(4,4)$$

$$Y_{3,4}^5 = -D(1,2)-D(1,5)+D(3,2)+D(3,5)-D(5,2)-D(5,5)$$

$$Y_{4,4}^0 = C(0,0)+C(0,3)+C(2,1)+C(2,4)+C(4,2)+C(4,5)+C(1,2)+C(1,5)+C(3,0)+C(3,3)+C(5,1)+C(5,4)$$

$$Y_{4,4}^2 = -C(0,2)-C(0,5)-C(2,0)-C(2,3)-C(4,1)-C(4,4)-C(1,1)-C(1,4)-C(3,2)-C(3,5)-C(5,0)-C(5,3)$$

$$Y_{4,4}^4 = C(0,1)+C(0,4)+C(2,2)+C(2,5)+C(4,0)+C(4,3)+C(1,0)+C(1,3)+C(3,1)+C(3,4)+C(5,2)+C(5,5)$$

$$Y_{5,4}^0 = D(0,0)+D(0,3)-D(2,2)-D(2,5)+D(4,1)+D(4,4)$$

$$Y_{5,4}^1 = D(1,2)+D(1,5)-D(3,1)-D(3,4)+D(5,0)+D(5,3)$$

$$Y_{5,4}^2 = -D(0,2)-D(0,5)+D(2,1)+D(2,4)-D(4,0)-D(4,3)$$

$$Y_{5,4}^3 = -D(1,1)-D(1,4)+D(3,0)+D(3,3)-D(5,2)-D(5,5)$$

$$Y_{5,4}^4 = D(0,1)+D(0,4)-D(2,0)-D(2,3)+D(4,2)+D(4,5)$$

$$Y_{5,4}^5 = D(1,0)+D(1,3)-D(3,2)-D(3,5)+D(5,1)+D(5,4)$$

$$Y_{6,4}^0 = C(0,0)-C(1,0)+C(2,0)-C(3,0)+C(4,0)-C(5,0)+C(0,3)-C(1,3)+C(2,3)-C(3,3)+C(4,3)-C(5,3)$$

$$Y_{6,4}^2 = -C(0,2)+C(1,2)-C(2,2)+C(3,2)-C(4,2)+C(5,2)-C(0,5)+C(1,5)-C(2,5)+C(3,5)-C(4,5)+C(5,5)$$

$$Y_{6,4}^4 = C(0,1)-C(1,1)+C(2,1)-C(3,1)+C(4,1)-C(5,1)+C(0,4)-C(1,4)+C(2,4)-C(3,4)+C(4,4)-C(5,4)$$

$$Y_{8,4}^0 = C(0,0)+C(0,3)+C(2,2)+C(2,5)+C(4,1)+C(4,4)+C(1,1)+C(1,4)+C(3,0)+C(3,3)+C(5,2)+C(5,5)$$

$$Y_{8,4}^2 = -C(0,2)-C(0,5)-C(2,1)-C(2,4)-C(4,0)-C(4,3)-C(1,0)-C(1,3)-C(3,2)-C(3,5)-C(5,1)-C(5,4)$$

$$Y_{8,4}^4 = C(0,1)+C(0,4)+C(2,0)+C(2,3)+C(4,2)+C(4,5)+C(1,2)+C(1,5)+C(3,1)+C(3,4)+C(5,0)+C(5,3)$$

$$Y_{10,4}^0 = C(0,0)+C(0,3)+C(2,1)+C(2,4)+C(4,2)+C(4,5)-C(1,2)-C(1,5)-C(3,0)-C(3,3)-C(5,1)-C(5,4)$$

$$Y_{10,4}^2 = -C(0,2)-C(0,5)-C(2,0)-C(2,3)-C(4,1)-C(4,4)+C(1,1)+C(1,4)+C(3,2)+C(3,5)+C(5,0)+C(5,3)$$

$$Y_{10,4}^4 = C(0,1)+C(0,4)+C(2,2)+C(2,5)+C(4,0)+C(4,3)-C(1,0)-C(1,3)-C(3,1)-C(3,4)-C(5,2)-C(5,5)$$

$$Y_{1,6}^0 = D(0,0)-D(0,1)+D(0,2)-D(0,3)+D(0,4)-D(0,5)$$

$$Y_{1,6}^1 = D(1,0)-D(1,1)+D(1,2)-D(1,3)+D(1,4)-D(1,5)$$

$$Y_{1,6}^2 = D(2,0)-D(2,1)+D(2,2)-D(2,3)+D(2,4)-D(2,5)$$

$$Y_{1,6}^3 = D(3,0)-D(3,1)+D(3,2)-D(3,3)+D(3,4)-D(3,5)$$

$$Y_{1,6}^4 = D(4,0)-D(4,1)+D(4,2)-D(4,3)+D(4,4)-D(4,5)$$

$$Y_{1,6}^5 = D(5,0)-D(5,1)+D(5,2)-D(5,3)+D(5,4)-D(5,5)$$

$$Y_{2,6}^0 = C(0,0)-C(0,1)+C(0,2)-C(0,3)+C(0,4)-C(0,5)-(C(3,0)-C(3,1)+C(3,2)-C(3,3)+C(3,4)-C(3,5))$$

$$Y_{2,6}^2 = C(1,0)-C(1,1)+C(1,2)-C(1,3)+C(1,4)-C(1,5)-(C(4,0)-C(4,1)+C(4,2)-C(4,3)+C(4,4)-C(4,5))$$

$$Y_{2,6}^4 = C(2,0)-C(2,1)+C(2,2)-C(2,3)+C(2,4)-C(2,5)-(C(5,0)-C(5,1)+C(5,2)-C(5,3)+C(5,4)-C(5,5))$$

$$Y_{3,6}^0 = D(0,0)-D(0,1)+D(0,2)-D(0,3)+D(0,4)-D(0,5)-(D(2,0)-D(2,1)+D(2,2)-$$

$$D(2,3)+D(2,4)-D(2,5)) + D(4,0)-D(4,1)+D(4,2)-D(4,3)+D(4,4)-D(4,5)$$

$$Y_{3,6}^3 = D(1,0)-D(1,1)+D(1,2)-D(1,3)+D(1,4)-D(1,5)-(D(3,0)-D(3,1)+D(3,2)-D(3,3)+D(3,4)-$$

$$D(3,5))+ D(5,0)-D(5,1)+D(5,2)-D(5,3)+D(5,4)-D(5,5)$$

$$Y_{4,6}^0 = C(0,0)-C(0,1)+C(0,2)-C(0,3)+C(0,4)-C(0,5)+ C(3,0)-C(3,1)+C(3,2)-C(3,3)+C(3,4)-C(3,5)$$

$$Y_{4,6}^2 = -(C(2,0)-C(2,1)+C(2,2)-C(2,3)+C(2,4)-C(2,5)+ C(5,0)-C(5,1)+C(5,2)-C(5,3)+C(5,4)-C(5,5))$$

$$Y_{4,6}^4 = C(1,0)-C(1,1)+C(1,2)-C(1,3)+C(1,4)-C(1,5)+ C(4,0)-C(4,1)+C(4,2)-C(4,3)+C(4,4)-C(4,5)$$

REFERENCES

- [1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine computation of complex Fourier series," *Math. Comp.*, vol. 19, pp. 297-301, Apr 1965.
- [2] C. L. Liu, *Introduction to Combinatorial Mathematics*. New York: McGraw-Hill, 1968.
- [3] A. J. Pettofrezzo, D. R. Byrkit, *Elements of Number Theory*. New Jersey: Prentice-Hall, 1970.
- [4] N. Ahmed, T. Natarajan and K. R. Rao, "Cooley-Tukey-type algorithm for the Haar transform," *Electronics Lett.*, vol. 9, no. 12, pp. 276-278, 14th June 1973.
- [5] Y. L. Henry, C. L. Mar and Sheng, "Fast Hadamard transform using the H diagram," *IEEE Trans. on Comput.*, pp. 957-960, Oct 1973.
- [6] N. Ahmed, T. Natarajan and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. C-23, pp. 90-93, Jan. 1974.
- [7] G. Bongiovanni, P. Corsini and G. Frosini, "One-dimensional and two-dimensional generalized discrete Fourier transforms," *IEEE Trans. on Acoust., Speech, and Signal Process.*, pp. 97-99, Feb. 1976.
- [8] D. B. Harris, J. H. McClellan, D. S. K. Chan, H. W. Schuessler, "Vector radix fast Fourier transform," *IEEE Int. Conf. on Acoust., Speech and Signal Process.*, Hartford, May 9-11, 1977, pp. 548-551.
- [9] M. J. Narasimha and A. M. Peterson, "On the computation of the discrete cosine transform," *IEEE Trans. on Comm.*, vol. COM-26, no. 6, pp. 934-936, June 1978.
- [10] J. H. McClellan and C. M. Rader, *Number theory in digital signal processing*. Englewood Cliffs, NJ: Prentice - Hall, 1979.
- [11] H. J. Nussbaumer, and P. Quandalle, "Fast computation of discrete Fourier transforms using polynomial transforms," *IEEE Trans. Acoustics, Speech and Signal Process.*, vol. 27, no. 2, pp. 169-181, April 1979.
- [12] J. Makhoul, "A fast cosine transform in one and two dimensions," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, no. 1, pp. 27-34, Feb 1980.
- [13] P. R. Roeser and M. E. Jernigan, "Fast Haar transform algorithms," *IEEE Trans. on comput.*, vol. c-3 1, no. 2, pp. 175-177, Feb. 1982.
- [14] C. Caraiscos and B. Liu, "Two dimensional DFT using mixed time and frequency decimations," in *proc. IEEE Int. Conf. on Acoust., Speech and Signal Process.*, May 1982, vol. 7, pp. 24-27.
- [15] R. D. Preuss, "Very fast computation of the radix-2 discrete Fourier transform," *IEEE Trans. Acoustics, Speech and Signal Process.*, vol. ASSP-30, no. 4, Aug. 1982, pp. 595-607.
- [16] L. N. Bhuyan and D. P. Agrawal, "Performance analysis of FFT algorithms on multiprocessor systems," *IEEE Trans. on software engineering*, vol. se-9, no. 4, Jul. 1983, pp. 512-521.

-
- [17] C. D. Thompson, "Fourier transforms in VLSI," *IEEE Trans. on comput.*, vol. c-32, 1983, pp. 1047-1057.
- [18] D. V. Hall, *Microprocessors and digital systems*. 2nd edition, Singapore: McGraw Hill, 1983.
- [19] W. H. Chen and W. K. Pratt, "Scene Adaptive Coder," *IEEE Trans. on commun.*, vol. com-32, no. 3, pp. 225-233, Mar. 1984.
- [20] S. M. Said and K. R. Dimond, "Improved implementation of FFT algorithm on a high-performance processor," *Electron. Lett.*, vol. 20, no. 8, pp. 347-349, 12th Apr. 1984.
- [21] D. E. Dudgeon and R. M. Mersereau, *Multidimensional signal processing*. Englewood Cliffs, NJ: Prentice - Hall, 1984.
- [22] Z. Wang, "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. Acoustics, Speech and Signal Process.*, vol. ASSP-32, no. 4, pp. 803-816, Aug. 1984.
- [23] M. Vetterli, "Fast 2-d discrete cosine transform," *ICASSP Apr. 1985*, vol. 10, pp. 1538-1541.
- [24] M. A. Mehalic, P. L. Rustan and G. P. Route, "Effects of architecture implementation on DFT algorithm performance," *IEEE Trans. Acoustics, Speech and Signal Process.*, vol. asp-33, no. 3, Jun. 1985, pp. 684-693.
- [25] C.X.Fan and S.H.Wancr, "A fast Fourier transform algorithm using Hadamard transform," *ICASSP, Tokyo, 1986*, pp. 225-228.
- [26] A. Guessoum and R. M. Mersereau, "Fast Algorithms for the multidimensional discrete Fourier transform," *IEEE Trans. Acoustics, Speech and Signal Process.*, vol. assp-34, no. 4, Aug. 1986, pp. 937-943.
- [27] G. E. Bridges, W. Pries, R. D. Mcleod, M. Yunik, P. G. Gulak and H. C. Card, "Dual systolic architectures for VLSI digital signal processing systems," *IEEE Trans. on comput.*, vol. c-35, no. 10, pp. 916-923, Oct. 1986.
- [28] E. E. Swartzlander Jr., *VLSI signal processing systems*. Kluwer-Academic, 1986.
- [29] O. K. Ersoy, "A two-stage representation of DFT and its applications," *IEEE Trans. Acoustics, Speech and Signal Process.*, vol. 35, no. 6, pp. 825-831, Jun. 1987.
- [30] T.K. Truong, I. S. Reed, I. Hsu, H. C. Shyu, and H.M. Shao, "A pipeline design of a fast prime factor DFT on a finite field," *IEEE Trans. on comput.*, vol. 37, no. 3, pp.266-273, Mar. 1988.
- [31] J. J. Komo and C. Yuan, "Four level Hadamard transform," in *Proc. of the 20th Southeastern Symp. on Syst. Theory*, Mar. 20-22, 1988, pp. 188-191.
- [32] E. O. Brigham, *The fast Fourier transform and its applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
- [33] G. Knowles, "VLSI architecture for the discrete wavelet transform," *Electron. Lett.*, vol. 26 no. 15, pp. 1184-1185, 19th Jul. 1990
- [34] R. Polge and B. Lawrence, "Comparison of a new multiple radix fast Fourier number theoretic transform with FFT algorithms in terms of performance and hardware cost," in *Proc. Southeast conference*, Apr. 1-4 1990, pp. 744-749.

- [35] P. Duhamel and C. Guillemot, "Polynomial transform computation of the 2-D DCT," ICASSP, Apr. 3-6 1990, vol. 3, pp. 1515-1518.
- [36] D. Yang, "Fast discrete radon transform and 2-d discrete Fourier transform," Electron. Lett., vol. 26, no. 8, pp. 550-551, 12th Apr. 1990.
- [37] M. H. Lee and Y. Yasuda, "Simple systolic array algorithm for Hadamard transform," Electron. Lett., vol. 26, no. 18, pp. 26-28, 30th Aug. 1990.
- [38] D. Yang, "Fast computation of two-dimensional discrete Fourier transform using fast discrete radon transform," in Proc. 10th IEEE Region Conf. on Comput. and Commun. Syst., Hong Kong, Sep. 1990, pp. 207-210.
- [39] A. Gupta and V. Kumar, "On the scalability of FFT on parallel computers," in Proc. 3rd Symposium Frontiers of Massively Parallel Computation, Oct 8-10 1990, pp. 69-74.
- [40] K. J. R. Liu, "VLSI computing architectures for Haar transform," Electron. Lett., vol. 26, no 23, pp. 1962-1963, 8th Nov. 1990.
- [41] A. S. Lewis and G. Knowles, "VLSI architecture for 2-D Daubechies wavelet transform without multipliers," Electron. Lett., vol. 27, no. 2, pp.171-173, 17th Jan. 1991.
- [42] S. Uramoto, O. Inoue, A. Takabatake, J. Takeda, Y. Yamashita, H. Terane and M. Yoshimoto, "A 100-MHz 2-D discrete cosine transform core processor," IEEE J. of Solid-State Circuits. vol. 27, no. 3, pp. 492-499, Apr. 1991.
- [43] D. Rodriguez, "A new FFT algorithm and its implementation on the DSP96002," ICASSP, Toronto, Canada, Apr. 14-17, 1991, vol. 3, pp. 2189-2192.
- [44] A. Puri and R. Aravind, "Motion-compensated video coding with adaptive perceptual quantization," IEEE Trans. Circuits Syst. for Video Technol., vol. 1, no. 4, pp. 351-361, Dec. 1991.
- [45] G. K. Wallace, "The JPEG still picture compression standard," Commun. ACM, 34(4), pp. 30-40, 1991.
- [46] L. M. Napolitano, and G. R. Redinbo, "On the efficiency of a new efficient algorithm to compute the two-dimensional discrete Fourier transform," IEEE Trans. Signal Process., vol. 40, no. 2, pp. 469-470, Feb. 1992.
- [47] S. C. Chan and K. L. Ho, "Fast Algorithms for Computing the Discrete Cosine Transform," IEEE Trans. Circuits Syst.-11: Analog Digital Signal Process., vol. 39, no. 3, pp. 185-190, Mar. 1992.
- [48] I. S. Reed, M. T. Shih, T. K. Truong, E. Hendon and D. W. Tufts, "A VLSI architecture for simplified arithmetic Fourier transform algorithm," IEEE Trans. Signal Process., vol. 40, no. 5, pp. 1122-1133, May 1992.
- [49] D. Slawewski and W. Li, "DCT/IDCT processor design for high data rate image coding," IEEE Trans. Circuits Syst. for Video Technol., vol. 2, no. 2, pp. 135-146, June 1992.
- [50] S. I. Sayegh, "A pipeline processor for mixed-size FFT's," IEEE Trans. Signal Process., vol. 40, no. 8, pp. 1892-1900, Aug. 1992.
- [51] S. C. Chan and K. L. Ho, "Split vector-radix fast Fourier transform," IEEE Trans. Signal Process., vol. 40, no. 8, pp. 2029-2039, Aug 1992.
- [52] Y. Huang, H. M. Dreizen and N. P. Galatsanos, "Prioritized DCT for compression and progressive transmission of images," IEEE Trans. Image Process., vol. 1, no. 4, pp. 477-487, Oct. 1992.

- [53] K. K. Parhi, "Impact of architecture choices on DSP circuits," in *proc. 10th Regional IEEE Conf., Tencon 92*, Nov 11-13, 1992.
- [54] J. L. Wu, W. J. Duh, and S. H. Hsu, "Basis-vector-decomposition based two-stage computational algorithms for DFT and DHT," *IEEE Trans. Signal Process.*, vol. 41, no. 4, Apr. 1993, pp. 1562-1575.
- [55] C. Lu, J. W. Cooley and R. Tolimieri, "FFT algorithms for prime transform sizes and their implementations on VAX, IBM3090VF, AND IBM RS/6000," *IEEE Trans. Signal Process.*, vol. 41, no. 2, Feb. 1993, pp. 638-648
- [56] H. Park and V. K. Prasanna, "Modular VLSI architectures for computing the arithmetic Fourier transform," *IEEE Trans. Signal Process.*, vol. 41, no. 6, pp.2236-2246, Jun. 1993.
- [57] A. Kumar and L. N. Bhuyan, "Parallel FFT algorithms for cache based shared memory multiprocessors," in *Proc. Int. Conf. on Paral. Process.*, Aug. 1993, vol. 3, pp. 23-27.
- [58] K. J. Ray and Liu, "Novel parallel architectures for short-time Fourier transform," *IEEE Trans. Circuits Syst.-11: Analog Digital Signal Process.*, vol. 40, no. 12, pp. 786-790, Dec. 1993.
- [59] D. Coppersmith, E. Feig, and E. Linzer, "Hadamard transforms on multiply/add architectures," *IEEE Trans. Signal Process.*, vol. 42, no. 4, Apr. 1994 pp. 969-970.
- [60] T. S. Wailes, W. D. O'Connor and M. A. Mehalic, "Design of a 16-point Winograd fast Fourier transform algorithm," in *Proc. Spring Conf. Integrated Circuit Syst., VHDL Int. Users Forum*, May 1-4, 1994, pp. 120-129.
- [61] A. Saidi, "Decimation-in-time-frequency FFT algorithm," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1994, vol. 3, pp. 453-456.
- [62] N. Koblitz, *A course in number theory and cryptography*, 2nd Edition. New Delhi, India: Springer, 1994.
- [63] J. G. Proakis and D. G. Manolakis, *Digital signal processing: principles, algorithms and applications*. New Delhi, India: Prentice- Hall, 1995.
- [64] N. Shirazi, P. M. Athanas and A. L. Abbott, "Implementation of a 2-D fast Fourier transform on a FPGA-based custom computing machine," in *Proc. 5th Int. Workshop Field-Programmable Logic and applications*, Sept 1, 1995, pp. 282-292.
- [65] Y. T. Ma, "A VLSI-oriented parallel FFT algorithm," *IEEE Trans. Signal Process.*, vol. 44, no. 2, pp. 445-448, Feb. 1996.
- [66] X. Qingbin, Z. Qing and S. Shenghe, "Iterative structure of Winograd FFT algorithm," in *Proc. Conference Precision Electromagnetic Measurement Digest*, China, Jun.17-21, 1996, pp. 57-58.
- [67] T. S. Lee, "Image representation using 2D Gabor wavelets," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, no. 10, pp. 959-971, Oct. 1996.
- [68] H. Lim and E. E. Swartzlander, Jr., "Efficient systolic arrays for FFT algorithms," in *Proc. 29th ASILOMAR*, Oct. 30 - Nov. 2, 1996, pp. 141-145
- [69] R. Gopikakumari and C. S. Sridhar, "A pictorial representation of 6x6 – point DFT in terms of 2x2 – point DFT," in *Proc. NCBME*, Chennai, Mar. 28 – 29, 1997.
- [70] B. J. Falkowski and S. Rahartlja, "Properties and applications of unified complex Hadamard transforms," in *Proc. 27th Int. Symp. on Multiple Valued Logic*, May 1997, pp. 131-136.

- [71] R. Gopikakumari and C. S. Sridhar, "A new modulo arithmetic based hierarchical neural network (MAHNN) model to implement $N \times N$ point DFT for $((N))_4 = 2$," in Proc. NET-X, CSI, Cochin, May 16-17, 1997.
- [72] R. Gopikakumari and C. S. Sridhar, "Performance evaluation of MAHNN model to implement $N \times N$ point DFT for $((N))_4 = 2$," in Proc. NET-X, CSI, Cochin, May 16-17, 1997.
- [73] R. Gopikakumari and C. S. Sridhar, "Visual manipulation of data for analysis – a DFT example," in Proc. BECON, Cochin, Sept. 4-6, 1997.
- [74] J. Pihl, "Tradeoffs between parallel and serial architectures in high performance digital signal processing", Proc. of IEEE ISIC'97, Singapore, 10-12 September 1997.
- [75] R. Gopikakumari and C. S. Sridhar, "A parallel distributed implementation of 6×6 point DFT," in Proc. Int. Conf. (ISIC), Nanyang Technological Uty, Singapore, Sept. 10-12, 1997, pp. 106-108.
- [76] T. Taketa, K. Tannol and S. Horiguch, "Radix r parallel FFT algorithms with a global interconnection networks and its evaluation," in Proc. 3rd Int. Symp. Parallel Architectures, Algorithms, and Networks, Dec. 18-20, 1997, pp. 424-428.
- [77] R. Gopikakumari and C. S. Sridhar, "An application of parallel distributed computation of 6×6 point DFT in the determination of ray paths," in Proc. National Symp. SYMBOL, Cochin, Dec. 16-17, 1997.
- [78] A. Papoulis, Signal Analysis, 1st edition. New York: McGraw Hill, 1977.
- [79] G. A. Ruiz and J. A. Michell, "Memory efficient programmable processor chip for inverse Haar transform," IEEE Trans. Signal Process., vol. 46, no. 1, pp.263-268, Jan. 1998.
- [80] J. M. Rabaey, "VLSI design and implementation fuels the signal processing revolution," IEEE Signal Process. Magazine, January 1998 pp. 22-36.
- [81] Y. M. Huang, J. L. Wu and C. T. Hsu, "A refined fast 2-D discrete cosine transform algorithm with regular butterfly structure," IEEE Trans. on Consumer Electronics, vol. 44, no. 2, pp. 376-383, May 1998.
- [82] M. Frigo and S. G. Johnson, "FFTW: an adaptive software architecture for the FFT," in Proc. Int. Conf. Accoust., Speech Signal process., May 12-15, 1998, vol. 3, pp. 1381-1384.
- [83] R. Gopikakumari, P. Salil and C. S. Sridhar, "Visual manipulation of symbols to implement 2-D DFT," in Proc. National Symp. (ANCS), Cochin, Sept. 23-25, 1998.
- [84] J. Villasenor and B. Hutchings, "The flexibility of configurable computing," IEEE Trans. Signal Process., vol. 15, no. 5, pp. 67-84, Sept. 1998.
- [85] G.M.Megson, "Systolic arrays for the Haar transform," in Proc. IEE Comput. Digit. Tech., vol. 145, no. 6, pp. 403-410, Nov. 1998.
- [86] F.G. Gray, G.A. Frank, B. Clark, D. Ziegenbein, S. Vuppala and P. Balasubramanian, "4.2: Tools for rapid construction of VHDL performance models for DSP systems," in Proc. Int. Verilog HDL Conf. and VHDL Int. Users Forum, IVC-VIUF, 1998, pp.77.
- [87] Q. H. Liu, and N. Nguyen and X. Y. Tang, "Accurate algorithms for nonuniform fast forward and inverse Fourier transforms and their applications," in Proc. IEEE Int. Symp. Geoscience and Remote Sensing, IGARSS, 1998, pp. 288 – 90.

- [88] R. Gopikakumari, "Investigations on the development of an ANN model & visual manipulation approach for 2-D DFT computation in image processing," Ph.D. Dissertation, Cochin University of Science and Technology, Kochi, 1998.
- [89] Y. M. Huang and J. L. Wu, "A refined fast 2-D discrete cosine transform algorithm," *IEEE Trans. Signal Process.*, vol. 47, no. 3, pp. 904-907, Mar 1999.
- [90] T. Chen, G. Sunada and J. Jin, "COBRA: A 100-MOPS single-chip programmable and expandable FFT," *IEEE Trans. VLSI Syst.*, vol. 7, no. 2, pp. 174-182, Jun. 1999.
- [91] S. Bouguezal, D. Chikouche, and A. Khellaf, "An efficient algorithm for the computation of the multidimensional discrete Fourier transform," *J. Multidim. Syst. Signal Process.*, vol. 10, pp. 275-304, Jul. 1999.
- [92] P. Lenders and A. Sjostrom, "On the implementation of nonseparable two-dimensional Haar wavelet transforms," *IEEE Trans. Signal Process.*, vol. 47, no. 11, Nov. 1999, pp. 3137-3139.
- [93] P. Adriaans and D. Zantinge, *Data mining*, 1st edition. Singapore: Addison Wesley, 1999.
- [94] K. K. Parhi, *VLSI digital signal processing systems, design and implementation*. New York: John Wiley & Sons, 1999.
- [95] D. Takahashi, "High-performance parallel FFT algorithms for the HITACHI SR8000," in *Proc. 4th Int. Conf. High performance computing in the Asia-Pacific region (HPC-Asia 2000)*, May 14-17, 2000, vol. 1., pp. 192-199.
- [96] J. Han, G. Ren and C. Han, "A novel fixed-point FFT algorithm on embedded digital signal processing systems, in *Proc. Signal process., WCCC-ICSP Aug. 21-25, 2000*, vol. 1, pp.48-53.
- [97] R. Bernardini, "A new multidimensional FFT based on one-dimensional decompositions," *IEEE Trans. Circuits Syst.-11: Analog Digital Signal Process.*, vol. 47, no. 10, pp. 1123-1126, Oct. 2000.
- [98] A. M. Grigoryan and S. S. Agaian, "Method of fast 1-d paired transforms for computing the 2-D discrete Hadamard transform," *IEEE Trans. Circuits Syst.-11: Analog Digital Signal Process.*, vol. 47, no. 12, pp. 1399-1404, Dec. 2000.
- [99] A. Amira, A. Bouridane and P. Milligan, "A novel architecture for Walsh Hadamard transforms using distributed arithmetic," in *Proc. 7th IEEE Int. Conf. Circuits and Syst., ICECS, 2000*, vol. 1, pp. 182-185.
- [100] I. Pitas, *Digital image processing algorithms and applications*. Wiley – IEEE, 2000.
- [101] M. Usama, Fayyad, G. G. Grinstein and A. Wierse, *Information visualization in data mining and knowledge discovery*. San Francisco, California: Morgan kaufmann, Elsevier, 2000.
- [102] M. Haldar, A. Nayak, N. Shenoy, A. Choudhary and P. Banerjee, "FPGA hardware synthesis from MATLAB," in *Proc. 14th Int. Conf. VLSI Design., Bangalore, India, Jan. 3-7, 2001*, pp. 299-304.
- [103] M. Nilsson, "FFT, realization and implementation in FPGA," *Msc. Thesis, Griffith University, Brisbane, Feb. 2001*.
- [104] D. Takahashi, "An extended split-radix FFT algorithm," *Signal Process. Lett.*, vol. 8, no. 5, pp. 145-147, May 2001.

- [105] A. Amira, A. Bouridane and P. Milligan, "An FPGA based Walsh Hadamard Transforms," in Proc. Int. Symp. Circuits and Syst. (ISCAS), May 2001, vol. 2, pp. 569-572.
- [106] L. V. Agostini, I. S. Silva and S. Bampi, "Pipelined fast 2-D DCT architecture for JPEG image compression," in Proc. 14th Symp. Integrated circuits and syst. design, Sept 10-15, 2001, pp. 226.
- [107] W. Zhilu, R. Guanghui and Z. Yaqin, "A study on implementing wavelet transform and FFT with FPGA," in Proc. 4th Int. Conf. ASIC., Shanghai, China, Oct. 23-25, 2001, pp. 486-489.
- [108] M. Nibouche, a. Bouridane, F. Murtagh , and O. Nibouche, "FPGA-based discrete wavelet transforms system," in Proc. 11th Int. Conf. FPGA, 2001, vol. 2147, pp. 607-612.
- [109] A. Amira, A. Bouridane, P. Milligan and R. Roula, "Novel FPGA implantations of Walsh-Hadamard transforms for signal processing," in Proc. IEEE Vis. Image Signal Process, vol. 148, no. 6, Dec. 2001, pp. 377-388.
- [110] U. M. Baese, Digital signal processing with field programmable gate arrays. New York: Springer, 2001.
- [111] Z. X. Yang, Y-P Hu, C. Y. Pan and L . Yang, "Design of a 3780 point IFFT processor for TDS-OFDM," IEEE Trans. Broadcast, vol. 48, pp. 57-61, Mar. 2002.
- [112] Y. Jiang, T. Zhou, Y. Tang and Y. Wang, "Twiddle-factor-based FFT algorithm with reduced memory access," in Proc. Int. Symp. Paral. Distributed Process. IPDPS., April 15-19, 2002, pp. 70-77.
- [113] P. Rodriguez V., "A radix-2 FFT algorithm for modern single instruction multiple data (SIMD) architectures," in Proc. Int. Conf. Accoust., Speech and signal process., ICASSP May 13-17, 2002, vol. 3, pp. 3220-3223.
- [114] Z. Razak and M. H. Yaacob, "VHDL development of a discrete wavelet transform," Malayasian J. Comput. Sci., vol. 15, no. 1, pp. 84-92, June 2002.
- [115] E. C. Ifeachor and B. W. Jeril, Digital signal processing – a practical approach. Delhi: Pearson Edn., 2002.
- [116] J. Heikkinen, J. Sertamo, T. Rautiainen and J. Takala, "Design of transport triggered architecture processor for DCT," in Proc. 15th Annual IEEE Int. ASIC/SOC Conf., Sep. 2002, pp. 87-91.
- [117] A. Moopenn and R. Tawel, "Parallel FPGA implementation of the split and merge discrete wavelet transform," 12th International conference FPL 2002 proceedings, Montpellier, France, Sept 2-4 2002.
- [118] G. Dimitroulakos, N. D. Zervas, N. Sklavos and C.E. Goutis, "An efficient VLSI implementation for forward and inverse wavelet transform for JPEG 2000," in Proc. 14th Int. IEEE Conf. Digital Signal Process., 2002, vol. 1, pp. 233-236.
- [119] R. G. Gonzalez and R. E. Woods, Digital Image Processing, 2nd edition. Delhi: Pearson Edn., 2002.
- [120] A. Aggoun and I. Jalloh, "Two-dimensional DCT/IDCT architecture," in Proc. IEE Comput. Digit. Tech., vol. 150, no 1, Jan. 2003, pp. 2-10.
- [121] S. M. Phoong and Y. P. Lin, "Lapped Hadamard transforms and filter banks," ICASSP, Apr. 6-10, 2003, vol. 6, pp. 509-512.

- [122] Y. Tang, L. Qian, Y. Wang and Y. Savaria, "A new memory reference reduction method for FFT implementation on DSP," in Proc. Int. IEEE Symp. Circuits and Syst., May 25-28, 2003, vol. 4, pp. 496-499.
- [123] C. L. Philips, J.M.Parr and E.A. Riskin, Signals, Systems and Transforms, 3rd edition. Delhi: Pearson Edn, 2003.
- [124] H. I. Saleh, M A. Ashour and A. E. Salamal, "GDFT types mapping algorithms and structured regular FPGA implementation," in Proc. Int. Symp. Circuits and Syst., May 25-28, 2003, vol. 4, pp. 129-132.
- [125] R. Gopikakumari, C. K. Jayadas, R. C. Roy, R. S. Roshni and C. S. Sridhar, "Semantic rule based visual representation of 2-D DFT for $N = 6$," in Proc. CISST, Las Vegas, Nevada, USA, Jun. 23-26, 2003, vol. 1, pp. 220-224.
- [126] R. Gopikakumari, C. K. Jayadas, R. C. Roy, R. S. Roshni and C. S. Sridhar, "A fast approach to visual representation of selected DFT for $((N))_4 = 2$," in Proc. CISST, Las Vegas, Nevada, USA, Jun. 23-26, 2003, vol. 1, pp. 225-229.
- [127] M. A. B. Ayed, M. Loulou, N. Masmoudi and L. Kamoun, "Reversible integer to integer wavelet transforms for image compression: implementation and evaluation," Lebanese science J., vol. 4, No. 2, 2003.
- [128] S. M. Aziz and M. Michel, "VHDL based design of an FDWT processor," IEEE Tencos, 2003, pp. 1609-1613.
- [129] R Mateos, A. Gardel, A. Hernandez, I. Bravo and C. Garcia, "Lossless implementation in VHDL of an image wavelet transform," in Proc. IEEE Emerging Technologies and Factory Automation, Sept 16-19, 2003, vol. 2, pp. 195-198.
- [130] A K. Jain, Fundamentals of digital image processing. New Delhi, India: Prentice Hall, 2003.
- [131] S. Bouguezel, M. O. Ahmad, and M.N.S. Swamy, "An improved radix-16 FFT algorithm," CCECE, Niagara Falls, Maylmai, 2004, pp. 1089-1092.
- [132] G. X. Fan, and Q. H. Liu, "Fast Fourier transform for discontinuous functions," IEEE Trans. Antennas Propagation, vol. 52, no. 2, pp. 461-465, Feb. 2004.
- [133] P. Banerjee, M. Haldar, A. Nayak, V. Kim, V. Saxena, S. Parkes, D. Bagchi, S. Pal, N. Tripathi, D. Zaretsky, R. Anderson, and J. R. Uribe, "Overview of a compiler for synthesizing MATLAB programs onto FPGAs," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol. 12, no. 3, pp. 312-324, Mar. 2004.
- [134] S. C. Pei and W. Y. Chen, "Split vector-radix-2/8 2-D fast Fourier transform," IEEE Signal Process. Lett., vol. 11, no. 5, pp. 459-462, May 2004.
- [135] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "Efficient output-pruning of the 2-D FFT algorithm," ISCAS, May 23-26, 2004, vol. 3, pp. 285-288.
- [136] C. Panis, U. Hirschrott, S. Farfeleder, A. Krall, G. Laure, W. Lazian, J. Nurmi, "A scalable embedded DSP core for SoC applications," in Proc. Int. Symp. Syst on Chip, Nov. 16-18, 2004, pp. 85-88.
- [137] A. A. Muhit, Md. S. Islam and M. Othman, "VLSI implementation of discrete wavelet transform (DWT) for image compression," in Proc. 2nd Int. Conf. Autonomous Robots and Agents, Palmerston North, New Zealand, Dec. 13-15, 2004, pp. 391-395.

- [138] T. Rintakoski, M. Kuulusa and J. Nurmi, "Hardware unit for OVVSF/Walsh/Hadamard code generation," in Proc. Int. Symp. System-on-Chip, Nov. 2004, pp. 143-145.
- [139] I. Amer, W. Badawy and G. Jullien, "A VLSI prototype for Hadamard transform with application to MPEG-4 part 10," in Proc. IEEE Int. Conf. Multimedia and Expo (ICME), 2004, pp. 1523-1526.
- [140] R. C. Roy and R. Gopikakumari "A new transform for 2-D signal representation (MRT) and some of its properties," 2004 IEEE Int. Conf. on Signal Process. Commun. (SPCOM), Bangalore, India, Dec. 11-14, 2004, pp. 363-367.
(http://www.ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1458423)
- [141] G. A. Ruiz, J. A. Michell and A. M. Buron, "Parallel-pipeline 8 x 8 forward 2-D ICT processor chip for image coding," IEEE Trans. Signal Process., vol. 53, no. 2, pp. 714-723, Feb. 2005.
- [142] Z. Liu, Y. Song, T. Ikenaga and S. Goto, "A VLSI array processing oriented fast Fourier transform algorithm and hardware implementation," in Proc. 15th ACM Great Lake Symp. VLSI (GLSVLSI'05), Chicago, Illinois, USA, Apr. 17-19, 2005, pp. 291-295.
- [143] I.S. Uzun, A. Amira and A. Bouridane, "FPGA implementation of fast Fourier transforms for real-time signal and image processing," IEE proc. Vis. image signal process., vol. 152, no. 3, Jun. 2005.
- [144] G. Lakshminarayanan and B. Venkataramani, "Optimization techniques for FPGA-based wave-pipelined DSP blocks," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol. 13, no. 7, pp. 783-793, Jul. 2005.
- [145] K. J. Horadam, "A generalized Hadamard transform," in Proc. Int. Symp. Inf. Theory (ISIT), Sep. 2005, pp. 1006-1008.
- [146] J. Y. Oh and M. S. Lim, "Area and power efficient pipeline FFT algorithm," in Proc. Int. Workshop. Signal Process. Syst. Design implementation, SIPS, Nov. 2-4, 2005, pp. 520-525.
- [147] P. Coussy, G. Corre, P. Bomel, E. Senn and E. Martin, "A more efficient and flexible DSP design flow from matlab-simulink," ICASSP 2005, pp. V-61-64.
- [148] Zhong Cui-xiang, Han Guo-qiang and Huang Ming-he, "Some new parallel fast Fourier transform algorithms," in Proc. 6th Int. Conf. Paral. Distrib. Comput., Appl. Technol., Dec. 5-8, 2005, pp. 624-628.
- [149] K. P. Soman and K. I. Ramachandran, Insight into wavelets from theory to practice, 2nd edition. New Delhi: PHI, 2005.
- [150] Sanjay Sharma, "Digital Signal Processing," 2nd revised edition, S. K. Kataria and Sons, New Delhi, 2005.
- [151] R. B. Northrop, Introduction to Instrumentation and measurements, 2nd edition. New York: CRC press, 2005.
- [152] J. A. R. Macias and A. G. Exposito, "Efficient computation of the running discrete Haar transform," IEEE Trans. Power Delivery, vol. 21, no. 1, pp. 504-505, Jan. 2006.
- [153] Y. J. Moon and Y. K. Daejeon, "A mixed-radix 4-2 butterfly with simple bit reversing for ordering the output sequences," in Proc. ICAOT, Korea, Feb. 20-22, 2006, pp. 1771-1774.
- [154] P. Salama, M. E. Rizkalla, M. Eckauer, "VHDL implementation of the fast wavelet transform," J. of VLSI Signal Process., vol. 42, pp. 223-239, 13 Feb. 2006.

- [155] R. C. Roy, M. S. Anish Kumar and R. Gopikakumari, "MRT: An alternate frequency domain representation," - IETE zonal conf., Kochi, Kerala, India, Mar. 25, 2006.
- [156] J. C. Goswami and A. K. Chan, *Fundamentals of Wavelets*, reprint. New Delhi, India: Wiley, 2006.
- [157] O. Atak, A. Atalar, E. Arikan, H. Ishebabi, D. Kammler, G. Ascheid, H. Meyr M. Nicola and G. Masera, "Design of application specific processors for the cached FFT algorithm," ICASSP, 2006, pp. III 1028-1031.
- [158] M. S. Anish Kumar, R. C. Roy, R. Gopikakumari, "A new image compression and decompression technique based on 8 x 8 MRT," ICGST Int. J. Graphics, Vision and Image Process., vol. 6, no. 1, pp. 51-53, Jul. 2006.
- [159] T. Y. Sung, "Memory-efficient and high-speed split-radix FFT/IFFT processor based on pipelined CORDIC rotations," in Proc. IEE Proc. Vis. Image Signal Process., vol. 153, no. 4, pp. 405-410, Aug. 2006.
- [160] P. M. Puig, "A family of fast Walsh Hadamard algorithms with identical sparse matrix factorization," IEEE Signal Process. Lett., vol. 13, no. 11, pp. 672-675, Nov. 2006.
- [161] S. Lee and S. C. Park, "Modified SDF architecture for mixed DIF/DIT FFT," in Proc. Int. Conf. Commun. Technol., Nov. 27-30, 2006, pp. 1-5.
- [162] T. Lenart and V. Owall, "Architectures for dynamic data scaling in 2/4/8k pipeline FFT cores," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol. 14, no. 11, pp. 1286-1290, Nov. 2006.
- [163] X. Li, S. Areibi and R. Dony, "Parallel processing on FPGAs: the effect of profiling on performance," in Proc. 6th Int. Workshop System on Chip for Real Time Appl., Dec. 2006, pp. 179-184.
- [164] S. Bouguezel, M. O. Ahmad, and M.N.S. Swamy, "An alternate approach for developing higher radix FFT algorithms," in Proc. IEEE Asia Pacific conf. on circuits and Syst. (APCCAS), Dec. 4-7, 2006, pp. 227-230.
- [165] A. Antoniou, *Digital Signal processing, signals, systems and filters*. New york: McGraw Hill, 2006.
- [166] C. P. Fan and G. A. Su, "A grouped fast Fourier Transform algorithm design for selective transformed outputs," in Proc. IEEE APCCAS, 2006, pp. 1939-1942.
- [167] R. C. Roy, M. S. Anish Kumar, R. Gopikakumari, "An invertible transform for image representation and its application to image compression," in Proc. 9th ISSPA, Sharjah, UAE, Feb. 12-15, 2007.
(http://www.ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4555504)
- [168] C. Cheng, and K. K. Parhi, "Low-cost fast VLSI algorithm for discrete Fourier transform," IEEE Trans. Circuits Syst. - I: Regular Papers, vol. 54, no. 4, Apr. 2007, pp. 791-806.
- [169] Y. Wang, Y. (Felix) Tang, Y. Jiang, J. G. Chung, S. S. Song and M. S. Lim, "Novel memory reference reduction methods for FFT implementations on DSP processors," IEEE Trans. Signal Process., vol. 55, no. 5, pp. 2338-2349, May 2007.
- [170] S. Lee and S. Chong, "Modified SDF architecture for mixed DIF/DIT FFT," in Proc. ISCAS IEEE Int. Symp. on circuits and syst., May 27-30, 2007, pp. 2590-2593.

- [171] S. C. Hsia and S. H. Wang, "Shift-register based data transposition for cost effective discrete cosine transform," *IEEE Trans. VLSI syst.*, vol. 15, no. 6, pp. 725-728, Jun. 2007.
- [172] O. Martin and J.M. Solana, "Programmable processor for on-line computing of inverse Haar transform," *Electron. Lett.*, vol. 37, no. 16, pp. 1050-1052, 2nd August 2007.
- [173] S. Balakrishnan and C. Eddington, "Efficient DSP algorithm development for FPGA and ASIC technologies," in *Proc. Int. Conf. Very Large Scale Integration, VLSI-SoC 2007*, pp. 168-171.
- [174] S. Bouguezzel, M. O. Ahmad, and M. N. S. Swamy, "A general class of split-radix FFT algorithms for the computation of the DFT of length- 2^m ," *IEEE Trans. Signal Process.*, vol. 55, no. 8, Aug. 2007, pp. 4127- 4138.
- [175] B. J. Mohd, A. Aziz and E. E. Swartzlander, Jr, "The Hazard free superscalar pipeline fast Fourier transform algorithm and architecture," in *Proc. IFIP Int. Conf. VLSI/SoC*, Oct. 15-17, 2007, pp. 194-199.
- [176] M. Szmajda, K. Gorecki, J. Mroczka, "DFT algorithm analysis in low-cost power quality measurement systems based on a DSP processor," in *Proc. 9th Int. Conf. Electrical Power Quality and utilization*, Barcelona, Oct. 9-11, 2007, pp. 1-6.
- [177] S. K. Palaniappan and T. Z. A. Zulkifli, "Design of 16-point radix-4 fast Fourier transform in 0.18 μ m CMOS technology," *American J. of Appl. Sci.* 4 (8): 570-575, 2007
- [178] N. Mahdavi, R. Teymourzadeh and M. B. Othman, "VLSI implementation of high speed and high resolution FFT algorithm based on radix 2 for DSP application," in *Proc. 5th Student Conf. on Research and Development*, Malaysia, Dec. 11-12, 2007, pp. 1-4.
- [179] M. J. Roberts, *Fundamentals of signals and systems*, Special Indian Edition. New Delhi: Tata McGraw Hill, 2007.
- [180] J. H. Bahn, J. Yang and N. Bagherzadeh, "Parallel FFT algorithms on network-on-chips," in *Proc. Int. Conf. Inf. technol.: New generations ITNG*, Apr. 7-9, 2008, pp. 1087-1093.
- [181] Z. Szadkowski, "An optimization of 16-point discrete cosine transform implemented into a FPGA as a design for a spectral 1st level surface detector trigger in the Pierre Auger observatory," in *Proc. Nuclear Sci. Symp. Conf.*, IEEE, 2008, pp. 2596-2601.
- [182] G. Bi, A. Aung, and B. Poh Ng, "Pipelined hardware structure for sequency-ordered complex Hadamard transform," *IEEE Signal Process. Lett.*, vol. 15, pp. 401-404, 2008.
- [183] S. Chaudhuri, S. Guilley, F. Flamen, "An 8×8 run-time reconfigurable FPGA embedded in a Soc," in *Proc. 45th ACM/IEEE Design Automation Conf. DAC*, Jun. 8-13, 2008, pp. 120-125.
- [184] P. K. Meher and J. C. Patra, "Fully-pipelined efficient architectures for FPGA realization of discrete hadamard transform," in *Proc. Int. Conf. Appl. Specific Syst., Architectures and Processors (ASAP)*, Jul. 2008, pp. 43-48.
- [185] M. S. Anish Kumar, R. C. Roy and R. Gopikakumari, "A new transform coder for gray scale images using 4x4 MRT," *AEU, Int. J. Electronics and Commun.*, vol. 62, no. 8, pp. 627-630, September 2008.
- [186] S. An, C. Wang, "Recursive algorithm, architectures and FPGA implementation of the two-dimensional discrete cosine transform," *IET Image Process.*, 2008, vol. 2, no. 6, pp. 286-294.

- [187] K. Wahid, S. Shimm, M. Islam, D. Teng, S.B. Ko and M. H. Lee, "Efficient hardware implementation of hybrid cosine-Fourier-wavelet transforms on a single FPGA," Taipei, Taiwan, May 24-27, 2009.
- [188] W. Ouyang and W.K. Cham, "Fast algorithm for Walsh Hadamard transform on sliding windows," IEEE Trans. Pattern Anal. Machine Intell., pp. 1-8, May 2009.
- [189] R. C. Roy, "Development of a new transform: MRT," Ph.D. Dissertation, Cochin University of Science and Technology, Kochi, 2009.

LIST OF PUBLICATIONS

Journal Publications

- [1] Bhadran V., Rajesh Cherian Roy, and R. Gopikakumari, "Algorithm to identify basic coefficients of 2-D DFT for any even N," vol. III, no. 1, pp 73-78, International Journal of Computational Intelligence, Research and applications, Jan-June 2009.
- [2] Bhadran V., Rajesh Cherian Roy, and R. Gopikakumari, "Computation of 2-D DFT: A Visual approach", (Communicated to IETE journal of research during April 2009).

Conference Publications

- [1] Bhadran V., Somesh Kashyap, Shubham Jain, Rupesh kumar, R. Gopikakumari, "A Visual representation of MRT in terms of 2×2 DATA", in proc. IETE zonal conference, Kochi, Kerala, India, Mar. 25, 2006.
- [2] Bhadran V., Rajesh Cherian Roy, and R. Gopikakumari, "A Visual representation of 2D-DFT in terms of 2×2 Data: A pattern analysis," in proc. International Conference on Computing Communication and Networking (ICCCN'08), Chettinad college of Engineering and Technology, Karur, India, Dec. 18-20, 2008.
(<http://www.ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4787762&isnumber=4787659>)
- [3] Bhadran V., Rajesh Cherian Roy, and R. Gopikakumari, "Algorithm to identify basic coefficients of 2-D DFT for any even N," in proc. ICVCOM, Saintgits college of Engineering, Kottayam, Apr. 16-18, 2009.