EXPLOITING PARALLELISM WITHIN MULTIDIMENSIONAL MULTIRATE DIGITAL SIGNAL PROCESSING SYSTEMS

A Dissertation

by

DONGMING PENG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2003

Major Subject: Computer Engineering

EXPLOITING PARALLELISM WITHIN MULTIDIMENSIONAL MULTIRATE DIGITAL SIGNAL PROCESSING SYSTEMS

A Dissertation

by

DONGMING PENG

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Mi Lu (Chair of Committee)	
Edward Dougherty (Member)	Jennifer Welch (Member)
Xi Zhang (Member)	Chanan Singh (Head of Department)

May 2003

Major Subject: Computer Engineering

ABSTRACT

Exploiting Parallelism within Multidimensional Multirate
Digital Signal Processing Systems. (May 2003)
Dongming Peng, B.S. and M.S., Beijing University of Aeronautics and Astronautics
Chair of Advisory Committee: Dr. Mi Lu

The intense requirements for high processing rates of multidimensional Digital Signal Processing systems in practical applications justify the Application Specific Integrated Circuits designs and parallel processing implementations. In this dissertation, we propose novel theories, methodologies and architectures in designing high-performance VLSI implementations for general multidimensional multirate Digital Signal Processing systems by exploiting the parallelism within those applications. To systematically exploit the parallelism within the multidimensional multirate DSP algorithms, we develop novel transformations including (1) nonlinear I/O data space transforms, (2) intercalation transforms, and (3) multidimensional multirate unfolding transforms. These transformations are applied to the algorithms leading to systematic methodologies in high-performance architectural designs. With the novel design methodologies, we develop several architectures with parallel and distributed processing features for implementing multidimensional multirate applications. Experimental results have shown that those architectures are much more efficient in terms of execution time and/or hardware cost compared with existing hardware implementations.

To my parents and my beloved Min

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to the chair of my committee, Dr. Mi Lu, for instilling in me the desire to pursue frontier research topics in parallel computing and VLSI signal processing, and for her help and patience throughout my PhD study. Without her review, advice, push, support and encouragement, I would — not be able to come up with the research ideas and results presented in this dissertation; not have the courage to confront the challenges of the high standards of publication in IEEE Transactions; not be willing to face up to my academic weakness and take efforts to strengthen myself; and not have the ambition to pursue an academic career in the United States.

Additional thanks are extended to other members of my committee, Dr. Jennifer Welch, Dr. Edward Dougherty, Dr. Norman Griswold and Dr. Xi Zhang, for their helpful suggestions and careful review of this work. Dr. Welch led me to the Advanced Algorithm Analysis through her wonderful teaching which positively changed the way of academic thinking for me, a student with an EE background. I am also thankful to Mr. Wayne Hamilton for serving as the Graduate Council representative on my committee.

I sincerely thank my wife, Min, for her never-ending love, care, patience, gentleness, understanding and support throughout my study. It is she who has made every day, month and year so hopeful and beautiful for me by being with me during these years in College Station although I have had to work hard all the time.

I am also thankful to my son, Charlie (Zheng Zheng), for his lovely and joyful smiles. I thank my parents, parents-in-law, sisters and sister-in-law for their love and support.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

A. General Introduction 1
1. Literature Review: Architectural Designs for Multidimensional Multirate DSP Systems 3
2. Outline of Proposed Methodologies 4
a. Non-RAM-based Architectural Designs of Wavelet-Based Systems 5
b. Achieving Intra-iteration Parallelism in Multidimensional Multirate Systems 6
c. Exploiting Inter-iteration Parallelism in Multidimensional Multirate Systems 7
B. Organization of the Dissertation 8

II NON-RAM-BASED AND CONTROL-DISTRIBUTED DESIGNS OF WAVELET-BASED DSP SYSTEMS 10

A. Introduction 10
B. Novel Nonlinear I/O Data Space Transforms 13
1. I/O Data Space Modeling 13
2. Novel Nonlinear I/O Data Space Transforms 17
C. Design Example: Non-RAM-Based Architectures for Zerotree Construction Systems 27
1. Zerotree Coding Algorithm 27
2. Applying I/O Data Space Transform to Zerotree Construction 29
3. The Architecture for Rearranging 2-level 2-D DWT . . 32
4. An Arbitrary Number of Levels in Zerotree Construction Systems 37

III EXPLOITING INTRA-ITERATION PARALLELISM IN MULTIDIMENSIONAL MULTIRATE SYSTEMS 40

A. Background Introduction 40
B. Modeling MD Multirate Algorithms in MR-MDFG's 43

CHAPTER Page

C. The MD Intercalation 53
D. Retiming Formulations for MR-MDFG's 57
1. Retiming Equations for Intercalated Iteration Space . 58
2. Serial Processing Order on MD Data Set 60
3. The Complete Retiming Formulation 62

IV EXPLOITING INTER-ITERATION PARALLELISM IN MULTIDIMENSIONAL MULTIRATE ALGORITHMS 65

A. Basic Properties 65
B. Multidimensional Multirate Unfolding 67
1. Construction of Unfolded MDFG 67
2. Properties of Multidimensional Unfolding 71
3. Translating MR-MDFG into Single-rate MDFG 72
C. Exploring the Inter-Iteration Parallelism 77

V EXPERIMENTS AND DESIGN EXAMPLES 86

A. Non-RAM-Based and Control-Distributed Deigns of Zerotree Construction Systems 86
B. Fully Retiming the MR-MDFG That Represents the 3-D DWT 91
C. Exploiting Maximum Inter-Iteration Parallelism in Multidimensional Multirate Systems 96

VI CONCLUSION 102

REFERENCES 106

VITA 114

LIST OF TABLES

TABLE		Page
I	The implementation of non-RAM-based zerotree construction	87
II	The gate-level implementation of RAM-based zerotree construction .	87
III	The gate-level implementation of DWT	88
IV	The performance analysis for the zerotree construction architecture .	90

LIST OF FIGURES

FIGURE		Page
1	An example of dependence graph for the 2-D MWT modeled in I/O data space	16
2	An example for the applications of the nonlinear I/O data space transformation on a 2-D wavelet packet transform	24
3	The zerotree construction in the EZW algorithm	28
4	The regularization of dependence graphs for zerotree construction	30
5	The systolic and parallel wavelet filters integrating low-pass and high-pass filtering	33
6	The architecture of Transpose Unit (TU)	34
7	The architecture for zerotree construction	35
8	An example of local-controlled 4-1 multiplexer	37
9	A simple example of an MDFG and the corresponding iteration space	41
10	The retiming operations on MDFG and iteration space	42
11	An example of MR-MDFG	45
12	An example showing the proof of lemma 3.1	47
13	The cycle-type rate-conflict MR-MDFG	48
14	A simple example of multidimensional unfolding transform on a 2-D MDFG	68
15	Another example of multidimensional unfolding on a 2-D MDFG	69
16	A multidimensional unfolding on a cyclic 2-D MDFG	71
17	An acyclic 1-D DFG executed with arbitrary concurrency	78

FIGURE Page

18 An acyclic 2-D MDFG executed with arbitrary concurrency 79

19 A trial of parallel scheduling of a cyclic 2-D MDFG 80

20 The MR-MDFG for 3-DWT and the retiming solutions 92

21 The MR-MDFG for 3-DWT after retiming operations 94

22 The architectural design for 3-DWT 95

23 The MR-MDFG representation of an SFQ algorithm and the partition of the MR-MDFG 98

24 The final translated single-rate MDFG 99

25 The subgraph of the single-rate MDFG containing cycles only 100

26 An illustration of the implementation of the SFQ algorithm in 2-D iteration space 101

CHAPTER I

INTRODUCTION

A. General Introduction

The intense requirements for high processing rates of multidimensional Digital Signal Processing (DSP) systems in practical applications justify the Application Specific Integrated Circuits (ASIC) designs and parallel processing implementations. The designs of ASIC and/or multiprocessor systems are usually required in order to improve the performance of applications such as multimedia processing, computer vision, high-definition television, medical imaging, remote sensing, and fluid dynamics. Due to the features of hierarchical signal analysis and multi-resolution analysis, many of these applications are *multirate* in nature, meaning that the sample rate is not uniform throughout the algorithm description. There are many famous multirate multidimensional DSP applications including Discrete Wavelet Transform (DWT), Full Wavelet Transform (FWT), Multi-Wavelet Transform (MWT)[1], M-ary Wavelet Transform, Wavelet Packet Transform (WPT)[2], Embedded Zerotree Coding, Set Partitioning In Hierarchical Trees (SPIHT), Spatial-Frequency Quantization (SFQ), and etc. Though the theory of multirate Digital Signal Processing (DSP) systems has matured over the past decade, there has been not much research on the theory of designing efficient ASIC architectures for multidimensional multirate systems yet. As a result, there has been a lack of computer-aided design (CAD) tools that can translate multidimensional multirate algorithms at a behavior level into efficient VLSI architectures by exploiting the parallelism within the algorithms [3][4][5][6][7][8][9].

In this dissertation, we propose several theories and methodologies in designing

---

The journal model is *IEEE Transactions on Automatic Control.*

high-performance VLSI architectures for general multidimensional multirate digital signal processing systems by exploiting the parallelism within those algorithms. There are two types of parallelism available in n-D signal processing. The first type of parallelism is inter-iteration parallelism (i.e., concurrent execution of iterations in an algorithm), which can be achieved by increasing the amount of hardware so multiple iterations can be executed concurrently. The second type of parallelism is intra-iteration (or inter-operation) parallelism (i.e., simultaneous execution of tasks *within* an iteration). In general, the retiming technique is involved to exploit the intra-iteration parallelism so operations within an iteration can be executed in parallel, resulting in a shorter clock period because more operations can be performed in parallel in circuits during each clock cycle.

The dissertation is concerned with both types of the parallelism. To systematically explore and exploit the parallelism within the multidimensional multirate DSP algorithms, we develop novel transformations including (1) nonlinear I/O data space transforms, (2) intercalation transforms, and (3) multidimensional multirate unfolding transforms. These transforms are applied to the DSP algorithms leading to the systematic methodology in high-performance VLSI architectural designs. With the novel design methodology, we propose several ASIC and multiprocessor VLSI architectures with parallel and distributed processing features for implementing general multidimensional and multirate DSP applications. It has been demonstrated that those architectures are much more efficient than existing hardware implementations in terms of execution time and/or hardware cost.

1. Literature Review: Architectural Designs for Multidimensional Multirate DSP Systems

Much research on architectural designs for 1-D DSP algorithms has been done based on modeling the loops or iterations of DSP algorithms as the Data-Flow Graphs (DFG). The loops or iterations of multidimensional DSP algorithms are represented by Multidimensional DFG (MDFG). The multirate multidimensional algorithms can be described by Multirate MDFG's (MR-MDFG), in which the difference from the normal MDFG's is that the edges in MR-MDFG's are doubly-weighted by *delays* and *multirates*.

As an important methodology in architectural designs of single-rate multidimensional DSP algorithms corresponding to the MDFGs, the *multidimensional* retiming, which has been recently proposed to exploit the intra-iteration parallelism, improves the circuitry performance by inserting a number of registers into circuit paths and reconstructing memory elements in a legal way. This technique guarantees that all functional elements in the MDFGs can be executed simultaneously on circuits designed to solve problems involving *more than one* dimension.

Most researches on retiming operations are focused on single-rate DSP algorithms only. However, there are still many problems open regarding the application of retiming operations onto multirate DSP systems. For example, is the technique of retiming operation applicable to an ARBITRARY multirate DSP dataflow graph? What are the necessary and sufficient conditions for the applicability of the retiming on the multirate dataflow graph? Is there a unified methodology for retiming operations on both single-rate and multirate systems? These problems are to be addressed in this dissertation based on a comprehensive modeling and analysis of multirate multidimensional dataflow graphs.

No published research has been devoted to exploring the inter-iteration parallelism within multirate multidimensional DSP algorithms. Although the notion of the parallelism in multidimensional applications has existed for a long time, it was so far unknown what the bound (if any) of inter-iteration parallelism in multirate multidimensional DSP algorithms is, and whether the maximum inter-iteration parallelism can be achieved for arbitrary multirate data-flow algorithms. In the context of 1-D single-rate algorithms, signal processing programs with recursion (or feedback) have a fundamental bound on the available parallelism, referred to as *iteration bound*. In calculating the iterations of a 1-D algorithm with feedback, we can never achieve an iteration period shorter than iteration bound, even when infinite processors are available. In the case of n-D multirate algorithms, the problem of exploring maximum available inter-iteration parallelism is far more complicated, especially because of the dependencies complicatedly existing in the n-D *iteration space*. With the contributions of *multidimensional intercalation and multidimensional multirate unfolding*, this dissertation explores the inter-iteration parallelism within multirate multidimensional DSP algorithms based on the method of *general selective shrinking*, and proves that this parallelism can always be achieved in hardware system given the availability of a large number of processors and the interconnections between them.

2. Outline of Proposed Methodologies

In literature, an approach increasingly referred to as *algorithm engineering* has received research interests. Algorithm engineering is the implementation of algorithms as abstract objects in 3-D physical space and time. The main research goal is to develop a theory that allows an expression of the correspondence between *abstract* and *physical* representation of algorithms and thus allows the transformation of an algorithm from one form to another. However, at the present time, the theory considers

only a very restricted class of algorithms with *uniform* data-dependence computation structures (which exclude all multirate algorithms), and produces only a restricted class of hardware implementations such as systolic arrays. Therefore, the extension of the capabilities of the existing theory is of great interest.

The three methodologies proposed in this dissertation play a significant role in the extension of the existing theory in algorithm engineering to the architectural designs of a general class of DSP algorithms, i.e., multidimensional multirate DSP algorithms, whose applications are used extensively in many image/video processing and pattern recognition systems.

a. Non-RAM-based Architectural Designs of Wavelet-Based Systems

Wavelet-based DSP algorithms belong to a particular class of multidimensional multirate DSP algorithms, and have been found very powerful in many DSP and pattern recognition applications. There have been a great deal of useful and complex wavelet-based multidimensional algorithms studied in literature including MWT [1][10][11], WPT [2][12], EZW [13], SPIHT [14], SFQ [15][16][17], and etc. In calculating these algorithms, off-chip Random Access Memory (RAM) based systems have been necessary, where either memory address pointers or data rearrangements in off-chip memories are employed because of the large size of 2-D input data. To the best of our knowledge, all recently proposed special-purpose architectures (e.g., [18][19][20][21][22][23][24]) for these complicate algorithms have to be involved with large off-chip RAMs when calculating and rearranging multidimensional data.

In this research, *we contribute to embedding the main bodies of these algorithms into non-RAM-based architectures leading to the elimination of off-chip communications and thus the significant increase of the processing rates which are especially desirable in image and video coding.*

Generally, the broadcast or communication of control information is another bottleneck that blocks the increase of processing rates and hardware efficiency in parallel processing systems. This dissertation proposes architectures featuring localized control where wavelet algorithms are computed by some processing units and all devices operate independently with local controls except using a single global clock signal.

Prior to presenting architectural designs, we establish and follow two novel concepts in data dependence analysis for generalized and arbitrarily multidimensional wavelet-based algorithms, i.e., *wavelet-adjacent field, and super wavelet-dependence vector*. Based on the concepts, novel nonlinear *I/O data space* transformations for variables localization and dependence graph regularization for wavelet algorithms are proposed which lead to designs of non-RAM-based architectures. The major contributions are exploring the computation locality and dependency within general wavelet-based algorithms by representing them with *wavelet-adjacent fields and super dependence vectors* based upon a newly defined model of "I/O data space", then regularizing and merging the dependence graphs via novel nonlinear I/O data space transformations, and finally, proposing non-RAM-based architectures with appropriate space-time mapping techniques based on the transformed dependence graphs.

b. Achieving Intra-iteration Parallelism in Multidimensional Multirate Systems

We have constructed in this part a complete theoretical analysis of modeling multidimensional multirate DSP algorithms in dataflow graphs. Based on the analysis, we propose a novel methodology of multidimensional intercalation in an iteration space that is expanded by replicating the data-flow graphs. In the theoretical analysis, a general class of DSP systems, rate-balanced multirate multidimensional DSP systems, is identified, with its precise definition proposed later in this dissertation. Many practically used multirate multidimensional DSP algorithms belong to this class, including

all wavelet-based algorithms and most Partial Differential Equation algorithms. We have demonstrated that the multidimensional retiming technique which was recently reported is inapplicable to arbitrary multirate dataflow graphs. The technique of retiming operation can be applied to a multirate DSP system if and only if the multirate system is rate-balanced. In this dissertation, by multidimensional intercalation we fully retime the rate-balanced MDFG so that there is no zero-valued delay weight between any two nodes in the retimed MDFG, which means that the intra-iteration parallelism can be fully exploited. Storage minimization and arbitrarily linear input format of multidimensional data are also addressed in this part.

c. Exploiting Inter-iteration Parallelism in Multidimensional Multirate Systems

In this part we propose the methodology of (1) multidimensional multirate unfolding, (2) translating a multirate multidimensional data-flow graph into a single-rate multidimensional data-flow graph, and (3) cyclic MDFG shrinking. Based on these approaches, we have shown how the inter-iteration parallelism is optimally exploited against precedence constraints within multirate multidimensional DSP algorithms in parallel processing implementation. As a measurement to achieve the full inter-iteration parallelism, an upper bound on the number of processors is given, which is derived from the topology and weights of the MR-MDFG and the shape of the iteration space. Any other implementations with a number of processors beyond this bound do not lead to further improvement.

While this part is mainly directed to a theoretical understanding of parallel processing implementations of multirate multidimensional DSP algorithms and an exploration of the inter-iteration parallelism, the design of methodologies in this part also helps discussing other topics regarding high-level synthesis of multirate multidimensional DSP algorithms. The proposal of multidimensional intercalation and

unfolding leads to a combination between (1) retiming operations for multirate multidimensional DSP algorithms in exploiting intra-iteration parallelism and (2) the multiprocessor implementations in exploiting inter-iteration parallelism. A direct mapping of multirate DSP algorithms to hardware would require data to move at different rates on the chip, which involves complicated routing and synchronization of multiple clock signals. The methodology of multidimensional unfolding and translating an MR-MDFG into a single-rate MDFG can lead to mapping a multirate DSP algorithm into a single-rate VLSI architecture, where the entire system operates with the same clock signal. No sub-clocks are necessary and the hardware efficiency is improved significantly.

B. Organization of the Dissertation

The dissertation is organized as follows. In Chapter II, we propose and investigate some novel nonlinear I/O data space transformations for generalized wavelet-based DSP algorithms which have a wide range of applications in many digital image/video processing systems. Based on the transformations, we present Non-RAM-based and control-distributed architectural designs for general wavelet-based digital systems. In Chapter III, we discuss multidimensional retiming operations in achieving intra-iteration operation parallelism for multidimensional multirate systems. To adapt the multidimensional retiming operation, which has been lately reported to be applied to single-rate systems only, to multirate systems, we introduce the methodology of intercalation based on a complete theoretical analysis of modeling multirate systems in dataflow graphs and in iteration space. In Chapter IV, we are dedicated to exploring inter-iteration parallelism within multirate multidimensional DSP algorithms based on the methodologies of multidimensional intercalation, unfolding and cyclic MDFG

shrinking. Chapter V shows our experimental results and design examples in three parts. The first part is for the simulation of non-RAM-based and control-distributed architectures for wavelet-based zerotree coding systems. The second part is for the hardware simulation in achieving intra-iteration parallelism for a 3-D wavelet-based system via multidimensional intercalation and retiming operations. The third part is a design example for implementation of a SFQ algorithm by exploring inter-iteration parallelism. In Chapter VI, we summarize the dissertation and make the conclusions.

CHAPTER II

NON-RAM-BASED AND CONTROL-DISTRIBUTED DESIGNS OF WAVELET-BASED DSP SYSTEMS

A. Introduction

There have been many useful wavelet-based multidimensional algorithms studied in literature including Multi-Wavelet Transform (MWT) [1][10][11][25], Wavelet Packet Transform (WPT) [2][15], Embedded Zerotree Wavelet transform (EZW) [13], Set Partitioning In Hierarchical Trees (SPIHT) [14], Space Frequency Quantization (SFQ), and etc. In these algorithms multidimensional data are decomposed into different spectral subbands, and correlations across the subbands are further analyzed and exploited in coding systems. The calculations of these complex algorithms are based on intense and complicated manipulation of multidimensional data. For instance, the algorithms of EZW, SPIHT and SFQ have three common procedures: 1) hierarchical wavelet decompositions, 2) construction of zerotree data structures, and 3) symbol generation from the wavelet coefficients on zerotrees, quantization of the magnitudes of significant coefficients and entropy coding. The second procedure, i.e., the zerotree construction, is the most important one that efficiently encodes the coefficients with a number of symbols by exploiting the inter-subband correlations of DWT via the zerotree data structure. Because the zerotrees are created from the 2-D data generated by DWT, in applications it is difficult locating the corresponding parent coefficient for a given child coefficient among the 2-D data [20]. For another instance, the application of 2-D MWT on images involves pre-processing images into 2-D vector-valued data streams, convoluting groups of data from adjacent rows with the matrix-valued wavelet filter taps, and then convoluting groups of data from adjacent columns in

the result of row-wise convolution. In calculating these algorithms, off-chip Random Access Memory (RAM) based systems have been necessary, where either memory address pointers or data rearrangements in off-chip memories are employed because of the large size of 2-D input data. As a matter of fact, to the best of our knowledge, all recently proposed special-purpose architectures (e.g., [18][19][20]) for these complex algorithms have to be involved with large off-chip RAMs when calculating and rearranging multidimensional data.

In this chapter, *we contribute to embedding the main bodies of these algorithms into non-RAM-based architectures leading to the elimination of off-chip communications and thus the increase of the processing rates which are especially desirable in image and video coding.* For example, one of our main ideas in building zerotrees for the algorithms of EZW, SPIHT or SFQ is to rearrange the calculations of wavelet transform and take advantage of parallel as well as pipelined processing, so that **ANY** parent coefficient and its children coefficients in zerotrees are guaranteed to be calculated and output simultaneously during the computation of wavelet transform. In this way, neither locating the parent for a given child coefficient nor building zerotree data structures is necessary any more after the computation of wavelet transforms. In other words, we combine the procedures of wavelet transform and zerotree construction into a single routine without using RAMs. The same philosophy can be applied to other wavelet-based algorithms described above. Principles of Non-RAM-based architectural designs for the wavelet based digital systems are proposed in the chapter.

Generally, the broadcast or communication of control information is one of the bottlenecks that block the increase of processing rates and hardware efficiency in parallel processing systems. This chapter proposes architectures featuring localized control where algorithms are computed by some processing units and all devices

operate independently with local controls except using a single global clock signal.

Prior to presenting architectural designs, this chapter establishes and follows two novel concepts in data dependence analysis for generalized and arbitrarily multidimensional wavelet-based algorithms, i.e., *wavelet-adjacent field, and super wavelet-dependence vector*. Based on them, novel nonlinear *I/O data space* transformations for variable localization and dependence graph regularization for wavelet algorithms are proposed which lead to designs of non-RAM-based architectures.

Unlike regular iterative algorithms which can be linearly mapped onto efficient regular architectures by conventional space-time mapping techniques, wavelet-based algorithms are characterized by a rather irregular data dependence structure largely due to sequence decimations, and the efficient space-time maps are bound to employ a certain form of nonlinearity. A thorough design space exploration can be accomplished for the linear synthesis in choosing linear space-time mapping functions, but selecting appropriate nonlinear mapping functions in architectural synthesis is still an open problem. A nonlinear index space transformation applied to synthesizing parallel structures for 1-D DWT has been reported in [23]. However, the approach in [23] is heuristic but not systematic, and it is ONLY applicable to 1-D case and cannot be extended to architectural designs for generalized and arbitrarily multidimensional wavelet algorithms. The major contributions in this chapter are exploring the computation locality and dependency within general wavelet-based algorithms by representing them with *wavelet-adjacent fields and super dependence vectors* based upon a newly defined model of "I/O data space", then regularizing and merging the dependence graphs via novel nonlinear I/O data space transformations, and finally, proposing non-RAM-based architectures with appropriate space-time mapping techniques based on the transformed dependence graphs.

Although the data dependence analysis and dependence graph regularization are

proposed in this chapter as a theoretical basis of architectural designs for generalized wavelet-based algorithms, with the space limit of this chapter, it is impossible here to derive architectural designs for all complex wavelet-based algorithms. In this chapter, we use the zerotree construction algorithm as the representative of wavelet based algorithms for deriving corresponding architectures. Zerotree construction algorithm is the main body of such algorithms as EZW, SPIHT and SFQ ([13][14][15][16]). The schemes in architectural designs in this chapter form the basis for synthesizing other more generalized wavelet-based digital systems. As such, we mention Full Wavelet Transform[17], M-ary Wavelet Transform[17] and Embedded zerotree coded multi-wavelet[10].

B. Novel Nonlinear I/O Data Space Transforms

1. I/O Data Space Modeling

The basic equation for any discrete wavelet algorithms is generally represented by

$$X_{j+1}[t] = \sum_{k \in L} C[k] X_j[Mt - k] \tag{2.1}$$

where $C[k]$ are taps of a wavelet filter, X_j and X_{j+1} are the *sequence* of input data and output data respectively at the $(j+1)^{th}$ level transform, L is a set that corresponds to the size of the wavelet filter, and M is a constant scalar in the algorithm.

Generally, the algorithm is termed as *M-ary wavelet transform* when $M > 2$. There are M wavelet filters for M-ary wavelet transform. j is always used in this chapter to refer to the wavelet transform level. If X_j and X_{j+1} are scalars and C is scalar-valued taps of the wavelet filter, the algorithm is a classical scalar wavelet transform; if X_j and X_{j+1} are vector-valued data and C is matrix-valued taps of

the multiwavelet filter, it is an MWT. If t and k are scalars, the algorithm is a 1-D transform; if t and k are n-component vectors, it is an n-D transform. Wavelet-based algorithms are multiresolution algorithms, i.e., the output data at a level of transform can be further transformed at the next level. In the classical DWT, there are two wavelet filters (low-pass filter and high-pass filter) at each level of transform, and only the output of low-pass filter is further transformed at the next level. In the arbitrary wavelet tree expansion of the WPT, the output of either filter may be further transformed at the next level. Note that the domain over which the sum is calculated by Eq. (2.1) is centered at $X_j[Mt]$.

Parameter index axis: The *parameter index axis* of a signal processing algorithm is the index axis for those data to be broadcasted in the algorithm, i.e., the data used in most computations but not generated by computations. As parameters of the computations, the number of them is fixed. *Data index:* The *data index* is the index for the intermediate data, input data, or output data that are generated and/or used by computations. In signal processing algorithms, the input size is variable.

We propose the following definitions for our modeling of wavelet algorithms in the n-D I/O data space.

I/O data space: In an *I/O data space* the indexed data can only be input data or output data, and the parameters of the algorithm are ignored. The intermediate data are viewed as partial inputs or partial outputs for the intermediate computations.

Wavelet-adjacent field: In an I/O data space, a *wavelet-adjacent field* is a small domain made up of a group of source data items used by a calculation in Eq. (2.1). Its size is dependent on the wavelet filter.

Super wavelet-dependence vector: A *super wavelet-dependence vector* $\overrightarrow{d_{Wb}}$ starts from a wavelet-adjacent field W and ends at the resulting data b. Since the source of the "dependence vector" itself is a domain instead of a single datum, we term

such a dependence vector (corresponding to the calculation in Eq. (2.1)) a *super wavelet-dependence vector*. In later analysis the super wavelet-dependence vectors are generally called dependence vectors and treated similarly as traditional dependence vectors. The length of a super wavelet-dependence vector $|\overrightarrow{d_{Wb}}|$ is defined as the Euclidean distance between a and b_c, where a and b_c are arithmetic centers of W and b respectively.

We also refine the following concepts which are used throughout this chapter.

Dependence graph: Although there are many versions of the definitions of dependence graph, in this chapter we make an emphasis on the *dependence graph based on an I/O data space*, where each node in the dependence graph corresponds to a data item, and each edge corresponds to a calculation or a dependence relation between the data used in the calculation and that generated in the calculation.

Regular dependence graphs: In such dependence graphs the length of each dependence vector $\mathbf{d}$ is a constant value independent of either the input size or the data positions.

Pseudo regular dependence graphs: In such dependence graphs the dependence vectors can be partitioned into a certain number of groups and in each group the length of dependence vectors is a constant value independent of either the input size or the data positions.

As examples, a wavelet-adjacent field, a super wavelet-dependence vector and the dependence graph including some instances of dependence vectors for the algorithm of *separable* 2-D MWT[11][26][27] are shown in Fig. 1 based on these concepts. The term "separable" means that the 2-D wavelet transform can be done row-wise and column-wise consecutively. In Fig. 1, j is the transform level; n_1 and n_2 are indices for the 2-D input data or 2-D output data. In Eq. (2.1) for the MWT, both X_j and X_{j+1} are vector-valued and C corresponds to the matrix-valued multiwavelet filter.

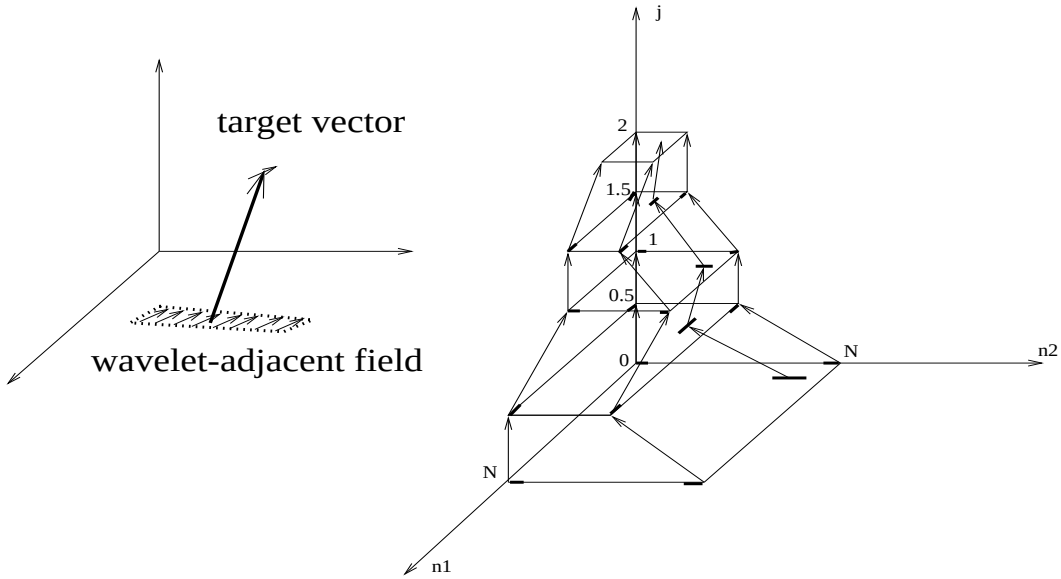


Fig. 1. An example of dependence graph for the 2-D MWT modeled in I/O data space

Thus the wavelet-adjacent field is a bunch of vectors and the end of the super wavelet-dependence vector itself is a vector (X_{j+1}) in Fig. 1. The separable 2-D MWT is performed row-wise and column-wise separately at each level of transform, and the dependence graph shown in Fig. 1 presents the dependence relationships in the I/O data space. Index j corresponds to the multi-wavelet transform level. In Fig. 1, Plane $j = 0$ in the 3-D space is the input plane which means the input data is always located at this plane, $j = 2$ is the output plane which means the final output data of the algorithm is always located at this plane, and $j = 1$ is an intermediate data plane. Plane $j = 1/2$ or $j = 3/2$ is the output plane of a row-wise transform and meanwhile the input plane of a column-wise transform. The dependence graph in Fig. 1 is apparently not a regular dependence graph as the length of dependence vectors is not a constant value yet depends on the positions of the target of the dependence vector.

2. Novel Nonlinear I/O Data Space Transforms

As illustrated in Fig. 1, the dependence graph for a wavelet-based algorithm in the I/O data space is irregular in that the lengths of the dependence vectors are dependent on the data positions as well as the input size. Whether it be a classical DWT, a vector-valued transform MWT, an arbitrary wavelet expansion WPT, or other wavelet-based algorithm, this irregularity always exists due to the sequence decimation in the nature of each level wavelet transform. Meanwhile, the output size is always M^n times less than the input size of each level transform (where n is the number of dimensions of the wavelet transforms). However, as proposed later, these irregular dependence graphs in the I/O data space can be generally regularized through a group of novel *nonlinear I/O data space transformations* so that the output data is uniformly distributed and the dependence vectors are regularized. The general dependence regularizations are formulated in the following proofs of *Theorem 2.1* and *Theorem 2.2*.

Theorem 2.1: The dependence graphs of wavelet algorithms modeled in I/O data space can always be regularized through appropriate nonlinear I/O data space transformations.

Proof:

The proof can be presented in two cases:

(1) For algorithms of 1-D wavelet transforms and non-separable n-D wavelet transforms: According to Eq. (2.1), the wavelet adjacent fields in the I/O data space correspond to groups of data $X_j[Mt - k]$ where $k \in L$, and L is a set that depends on the size of the wavelet filter and the number of the dimensions of the wavelet transform. The super dependence vectors represent the dependence relationships between the wavelet adjacent fields and the target data $X_{j+1}[t]$, and the lengths of

dependence vectors are the distances between centers of the wavelet adjacent field and the target data. When performing the nonlinear I/O data space transformation Γ_1: $j \longmapsto j$, $t \longmapsto M^j t$, we can always make the lengths of dependence vectors a constant value and thus regularize the dependence graph as analyzed in the following. After rearranging data, the dependence vector in the I/O data space corresponding to the dependence relation between $X_{j+1}[t]$ and the wavelet-adjacent filed $X_j[Mt-k]$ (where $k \in L$) changes to the dependence vector for the dependence relation between $X_{j+1}[M^{j+1}t]$ and the wavelet-adjacent field $X_j[M^{j+1}t - M^j k]$ (where $k \in L$). Meanwhile, the length of this dependence changes to the distance between $X_{j+1}[M^{j+1}t]$ and $X_j[M^{j+1}t]$, or $|(j+1)-j| = 1$. In other words, the dependence graph is regularized by performing the nonlinear I/O data space transformation Γ_1.

(2) For separable n-D wavelet transforms: The multi-dimensional filter $C[K]$ is separable for the separable n-D wavelet transforms, and Eq. (2.1) is separated to be calculated in each dimension. Generally, Eq. (2.1) is calculated equivalently by the following Eq. (2.2) in separable n-D wavelet transforms:

$$\begin{aligned} & X_{j+1}[t_1, t_2, \cdots, t_n] \\ = & \sum_{k_1 \in L_1} C_1[k_1] \sum_{k_2 \in L_2} C_2[k_2] \cdots \sum_{k_n \in L_n} C_n[k_n] X_j[Mt_1 - k_1, Mt_2 - k_2, \cdots, Mt_n - k_n] \end{aligned} \quad (2.2)$$

where $(t_1, t_2, \cdots, t_n)$ are components of n-component vector t, $(k_1, k_2, \cdots, k_n)$ are the components of k, $(Mt_1 - k_1, Mt_2 - k_2, \cdots, Mt_n - k_n)$ are the components of $Mt - k$ in Eq. (2.1), $C_1, C_2, \cdots, C_n$ are n 1-D filters separated from the n-D filter C, and $L_1, L_2, \cdots, L_n$ correspond to n 1-D wavelet-adjacent fields separated from the original n-D wavelet-adjacent field. In order to give the dependence graph for Eq. (2.2) in the I/O data space, we draw the index j (which represents the level of multiresolution

transforms and can only be integers in case (1)) in fractional numbers to represent the intermediate calculations in every level of transform. For example, the plane $j = 0.5$ in Fig. 1 corresponds to the output of the first level row-wise wavelet transform and the input of the first level column-wise wavelet transform. In the $(s+1)^{th}$ level (where s is a non-negative integer) of an n-D wavelet transform, we thus have intermediate planes $j = s + 1/n$, $j = s + 2/n$, $\cdots$, $j = s + (n-1)/n$ between the plane $j = s$ and $j = s + 1$. Accordingly, Eq. (2.2) can be calculated in the order of

$$
\begin{aligned}
&\textstyle\sum_{k_1 \in L_1} C_1[k_1] \sum_{k_2 \in L_2} C_2[k_2] \cdots \sum_{k_n \in L_n} C_n[k_n] X_s[Mt_1 - k_1, Mt_2 - k_2, \cdots, Mt_n - k_n] \\
&= \textstyle\sum_{k_2 \in L_2} C_2[k_2] \cdots \sum_{k_n \in L_n} C_n[k_n] X_{s+1/n}[t_1, Mt_2 - k_2, \cdots, Mt_n - k_n] \\
&= \textstyle\sum_{k_3 \in L_3} C_3[k_3] \cdots \sum_{k_n \in L_n} C_n[k_n] X_{s+2/n}[t_1, t_2, Mt_3 - k_3, \cdots, Mt_n - k_n] \\
&= \cdots \\
&= \textstyle\sum_{k_n} \in L_n C_n[k_n] X_{s+(n-1)/n}[t_1, t_2, \cdots, t_{n-1}, Mt_n - k_n] \\
&= X_{s+1}[t_1, t_2, \cdots, t_n] \qquad (2.3)
\end{aligned}
$$

When performing the nonlinear I/O data space transformation Γ_2: $j \longmapsto j$, $t_i \longmapsto M^{\lceil j - \frac{i}{n} + \frac{1}{2n} \rceil} \times t_i$ for $i = 1, 2, \cdots, n$, we can make the lengths of dependence vectors a constant value and thus regularize the dependence graph corresponding to Eq. (2.3) as analyzed in the following. Here $\frac{1}{2n}$ is used in the expression for adjusting the value of ceiling function. The length of the dependence vector is $|(s + i/n) - (s + (i-1)/n)| = 1/n$. In other words, the dependence graph for the calculation of the separable wavelet transform is regularized by performing the nonlinear I/O data space transformation Γ_2.

Note that we have not assumed that the data calculated in the wavelet transforms are scalar-valued, so the proof is also applicable to MWT.

In the algorithm of WPT, the computation has a structure of an arbitrary wavelet tree expansion, and the algorithm is calculated by iterating the wavelet branches of the filter bank to give a finer resolution to the wavelet decomposition, where not only the coarse component but also the detailed component is possibly further decomposed. The classical DWT can be taken as a special instance of WPT, where only the coarse component is recursively decomposed at each level of transform. The calculation for any wavelet filter in the computation of WPT follows Eq. (2.1), with C replaced by different wavelet filters. We can construct the dependence graphs corresponding to the calculations of WPT, and regularize them similarly by nonlinear I/O data space transformations Γ_1 and Γ_2 as introduced above. Nevertheless, each dependence graph represents only one path in the arbitrary expansion of wavelet tree, and the whole dependence graph of WPT should be the combination of dependence graphs corresponding to all paths in the expanded wavelet tree of WPT. Such process of combining the dependence graphs via nonlinear I/O data space transformations is formulated in the following proof of *Theorem 2.2*.

Theorem 2.2: The dependence graphs of wavelet-packet based algorithms modeled in I/O data space can always be merged and regularized to a pseudo regular dependence graphs via appropriate nonlinear I/O data space transformations.

Proof:

The symbols $j, t, X, L, t_1, t_2, \cdots, t_n, L_1, L_2, \cdots, L_n$ have the same meanings as in the previous paragraphs. The proof can be presented in three cases:

(1) For algorithms of 1-D transforms: There are M wavelet filters $(f_1, f_2, \cdots, f_M)$ at each level of 1-D M-ary wavelet transform, and each level of transform can decompose a certain subband into M components in wavelet-packet based algorithms. One of the filters (f_1) is for generating coarse component, others for detailed components. Assume M functions $F_i(x) = Mx + i - 1$ for $i = 1, 2, \cdots, M$.

Suppose that a subband Π is calculated in l levels of wavelet-packet based transform consecutively with wavelet filters $p_1, p_2, \cdots, p_l$, where $p_u = f_i$ for $u = 1, 2, \cdots, l$ and i is any integer $\in [1, M]$. Since the calculation of each level of transform follows the same format in Eq. (2.1), the corresponding dependence graph of Π can be constructed and regularized via the I/O data space transformation similarly as in the proof of *Theorem 2.1.* However, considering that there are many subbands generated together by l levels of wavelet-packet based transform, and their corresponding dependence graphs should be merged as well as regularized to get a whole dependence graph for the algorithm, the nonlinear I/O data space transformation Γ_3 is presented as follows. Without losing generality, for the dependence graph corresponding to subband Π, Γ_3 is: $j \longmapsto j; t \longmapsto t$ if $j = 0$; $t \longmapsto P_1(P_2(\cdots(P_j(t))\cdots))$ otherwise, where $P_u = F_i$ if $p_u = f_i$ for $u = 1, 2, \cdots, j$, and $j \leq l$, $i \in [1, M]$. Note that here j corresponds to the level of transform and can be only integers.

Consider another subband Π_1 different from Π generated in the algorithm. Suppose that Π_1 is calculated in l levels consecutively with wavelet filters $p'_1, p'_2, \cdots, p'_l$, where $p'_u = f_i$ for $u = 1, 2, \cdots, l$ and i is any integer $\in [1, M]$. Since there exits at least one $p'_u \neq p_u$ where $u \in [1, l]$, Γ_3 maps the data for Π and Π_1 to different positions in the I/O data space. In other words, Γ_3 can combine all dependence graphs of the subbands into a single I/O data space without conflicts.

(2) For nonseparable n-D transforms: There are $Q = M^n$ different wavelet filters $(f_1, f_2, \cdots, f_Q)$ at each level of n-D M-ary wavelet transform. Assume Q functions $F_i(x) = Mx + q$, where x and q are n-component vectors. The components of q are $q_1, q_2, \cdots, q_n$, and q_v is an integer $\in [0, M-1]$ for $v = 1, 2, \cdots, n$, and $i = \sum_{u=1}^{n} M^u q_u$. So $i \in [1, Q]$.

Suppose that a subband Π is calculated in l levels of n-D wavelet-packet based transform consecutively with wavelet filters $p_1, p_2, \cdots, p_l$, where $p_u = f_i$ for $u =$

$1, 2, \cdots, l$ and $i \in [1, Q]$. Without losing generality, for the dependence graph corresponding to subband Π, a nonlinear I/O data space transformation Γ_4 is presented as: $j \longmapsto j; t \longmapsto t$ if $j = 0$; $t \longmapsto P_1(P_2(\cdots(P_j(t))\cdots))$ otherwise, where $P_u = F_i$ if $p_u = f_i$ for $u = 1, 2, \cdots, j$, and $j \leq l$, $i \in [1, Q]$. Note that here j corresponds to the level of transform and can be only integers, and t represents n-component vectors.

For other subbands different from Π generated in the algorithm, since there exists at least one filter used in the calculation of l levels of transform different from that of Π, Γ_4 maps the data of them to different positions. In other words, Γ_4 can combine all dependence graphs of the subbands into a single I/O data space without conflicts.

Similar to case (1), a calculation corresponding to the dependence vector changes to $X_{u+1}[P_1(P_2(\cdots(P_u(P_{u+1}(t)))\cdots))] = \sum_{k\in L} p_{u+1}[k] X_u[P_1(P_2(\cdots(P_u(Mt-k))\cdots))]$, where $P_v = F_i$ if $p_v = f_i$ for $v = 1, 2, \cdots, u+1$, and $i \in [1, Q]$. The difference between the coordinates of the source and the target of the dependence vector along index j is $|u + 1 - u| = 1$. The difference along t is $P_1(P_2(\cdots(P_u(P_{u+1}(t)))\cdots)) - P_1(P_2(\cdots(P_u(Mt))\cdots)) = P_1(P_2(\cdots(P_u(Mt + w))\cdots)) - P_1(P_2(\cdots(P_u(Mt))\cdots)) = M^u w$, where *w is an n-component vector* whose components are integers $\in [1, M-1]$. Thus the length of the dependence vector is independent of t's value. We have the similar conclusion that the lengths of the dependence vectors in the I/O data space after the mapping of Γ_4 are bounded and independent of the data positions and input size, and the dependence vectors can be partitioned into a finite number of groups (according to the possible values of w and u), and the lengths of the dependence vectors in each group are the same.

(3) For separable n-D transforms: As in case (2) of the proof for *Theorem 2.1*, the n-D separable transforms are calculated separately and consecutively in every dimension. The index j is drawn in fractional numbers to represent the intermediate calculations in each level of transform. In the $(s+1)^{th}$ level (where s is a non-

negative integer) of a separable n-D wavelet transforms, we have (n-1) intermediate I/O data planes $j = s + 1/n$, $j = s + 2/n$, $\cdots$, $j = s + (n-1)/n$ between the planes $j = s$ and $j = s + 1$. In the calculations for every dimension, there are M wavelet filters $f_1, f_2, \cdots, f_M$, and a subband may be decomposed into M components on each dimension. So after each level of transform, a subband can be decomposed into M^n components. In addition, we assume M functions $F_v(x) = Mx + v - 1$ for $v = 1, 2, \cdots, M$.

Suppose that a certain subband Π is calculated in l levels of n-D separable wavelet-packet based transform consecutively with wavelet filters $p_{1,1}, p_{1,2}, p_{1,n}, p_{2,1}$, $\cdots, p_{2,n}, \cdots, p_{l,n}$, where $p_{u,i} = f_v$ for $u = 1, 2, \cdots, l$ and $i = 1, 2, \cdots, n$, and $v \in [1, M]$. $p_{u,i}$ represents the wavelet filter used for the calculation of the u^{th} level transform on the i^{th} dimension in generating Π. In order to regularize the dependence graphs, we present the nonlinear I/O data space transformation Γ_5 as follows. Without losing generality, for the dependence graph corresponding to subband Π, Γ_5 is: $j \longmapsto j$, $t_i \longmapsto t_i$ $(i = 1, 2, \cdots, n)$ if $j = 0$; $t_i \longmapsto P_{1,i}(P_{2,i}(\cdots(P_{s+1,i}(t_i))\cdots))$ otherwise, with $j \in [s + i/n, s + 1 + i/n)$, s being an integer $\in [0, l-1]$, $P_{u,i} = F_v$ for $p_{u,i} = f_v$ $(u = 1, 2, \cdots, s+1;\ i = 1, 2, \cdots, n;$ and $v \in [1, M])$.

For other subbands different from Π generated in the algorithm, since there exits at least one filter used in the calculation of l levels of transform different from that of Π, Γ_5 maps the data of them to different positions.

To sum up, the dependence graphs for wavelet-packet based algorithms are combined and regularized to be a pseudo regular dependence paragraph via the nonlinear I/O data space transformation Γ_3, Γ_4 or Γ_5.

An example of I/O data space transformation Γ_5 is illustrated in Fig. 2.

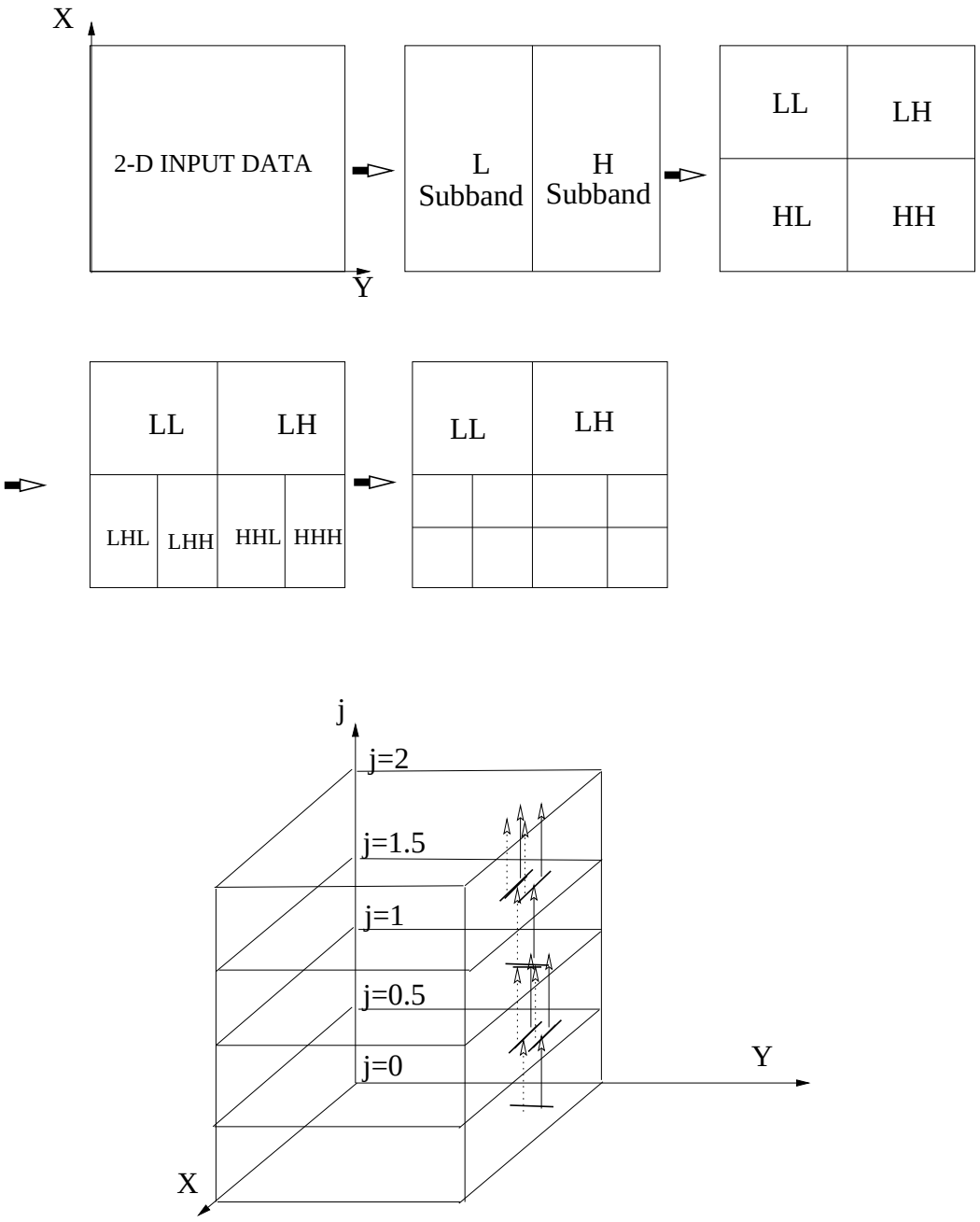


Fig. 2. An example for the applications of the nonlinear I/O data space transformation on a 2-D wavelet packet transform

Although nonlinear input formats of multidimensional data such as random access or pseudo-fractal scan[26] also exist, the chapter only considers linear input format of a multidimensional data set. This is to prevent systems from potentially involving data preprocessing and rearrangement in RAMs before computation. Suppose that the input data is n-dimensional. A group of n n-component unitary orthogonal vectors $\{S_1, S_2, \cdots, S_n\}$ is used to describe a linear input format. Assume that A and B are any two samples indexed by n-component Cartesian vectors X and Y respectively in the multidimensional data set. (X, Y) is the inner product of two vectors X and Y. The linear input format can be described as: 1) if $(X, S_1) \neq (Y, S_1)$, the sample corresponding to the less of the two inner products will be input to the system earlier; 2) else if $(X, S_2) \neq (Y, S_2)$, the sample corresponding to the less of the two inner products will be input to the system earlier; 3)$\cdots$; n) else if $(X, S_n) \neq (Y, S_n)$, the sample corresponding to the less of the two inner products will be input to the system earlier.

Now consider the dependence graphs regularized by the nonlinear I/O data space transformations proposed in this sub-section. The n-dimensional input data is always located on the super plane $j = 0$ shown as in Fig. 2, and it is scanned in a linearly indexed order when the data is input to the system. The I/O data space is (n+1)-dimensional. On super planes $j = c + i/n$ (where c and i are integers, $c > 0$ and $n > i \geq 0$) located are intermediately calculated data which makes up wavelet-adjacent fields for the next level calculation. Dependence vectors starting from the wavelet-adjacent fields are located between every two neighboring super planes.

A scheme of *free schedule* is used to optimize the system performance, which schedules a calculation in the I/O data space to be executed as soon as its operands (i.e., the data in wavelet-adjacent field) are ready. Due to the dependence graph regularization via nonlinear I/O data space transformations, dependence vectors and

wavelet-adjacent fields are uniformly distributed on super planes which are orthogonal to axis j. When the input data items are fed to the system one by one according to a linear input format, the wavelet-adjacent fields on plane $j = 0$ are scanned one by one, and the corresponding dependence vectors which start from this plane and take these wavelet-adjacent fields as sources are ready to be processed one by one. Since dependence vectors are along the orientation of axis j due to I/O data space transformations, the calculation results on the next super plane, or the targets of these dependence vectors, can be produced one by one in a linear order which is the same as the system's linear input format, resulting in that the wavelet-adjacent fields in this super plane are also "scanned" one by one, and dependence vectors between this plane and the further next plane are ready to be processed in the same linear order, and so on. Thus, if we assign each level of algorithm computation, or each layer of dependence vectors in I/O data space, to a separate processor, have all these processors execute simultaneously, and let the processing rates match the system's data-feeding rate, we can finish the algorithm computation as soon as the feeding of input data is finished. In this scheme, we avoid using RAMs to rearrange or manipulate multidimensional input data, which was necessary in the complex computation structure of wavelet algorithms. Since any wavelet-based algorithm consists of wavelet transforms that follow Eq. (2.1), and basically uses data structures on the results of wavelet transforms that can be modeled and reformulated in I/O data space as proposed in this section, we can apply the above scheme to general wavelet-based algorithms without using RAMs. The following section gives more detailed explanations in design examples for specific wavelet-based algorithms.

Because the dependence vectors and wavelet-adjacent fields are uniformly distributed on super planes which are orthogonal to axis j due to I/O data space transformations $\Gamma_1 - \Gamma_5$, when the input data items are fed to the system in linear order at

a constant rate, the processors, each of which is to perform calculations corresponding to each layer of dependence vectors in I/O data space, will operate periodically according to the scheme described in the above paragraph. For instance, when 2-D input data which is located at plane $j = 0$ in Fig. 2 is fed to the system in row-major format, the processor that needs to perform calculations for dependence vectors between plane $j = 0$ and $j = 1/2$ will alternate performing low-pass and high-pass wavelet filtering periodically. The other processor that performs calculations for dependence vectors between planes $j = 1/2$ and $j = 1$ has the similar feature of periodic operations. This feature of periodic operations of parallel processors, which is independent of input data or intermediate calculation results, leads to control-localized parallel processing architectures.

C. Design Example: Non-RAM-Based Architectures for Zerotree Construction Systems

1. Zerotree Coding Algorithm

Zerotree coding is the common and the most important part of algorithms EZW, SPIHT and SFQ. With the limit of the length of this chapter, we briefly review EZW as the typical scheme of the class of wavelet zerotree coding algorithms, and make our designs of zerotree construction based on the scheme without losing generality. Hereby we quote several definitions like parent, child, descendent, root, zerotree, significance, dominant pass, subordinate pass etc. from the reference [13] to describe the EZW coding scheme. As illustrated in Fig. 3, the input image is transformed and subsampled using the hierarchical DWT to obtain a collection of 3S+1 subband images, where S is the number of transform levels. As wavelet coefficients in the subband images have some correlations along the same orientation (horizontal, vertical

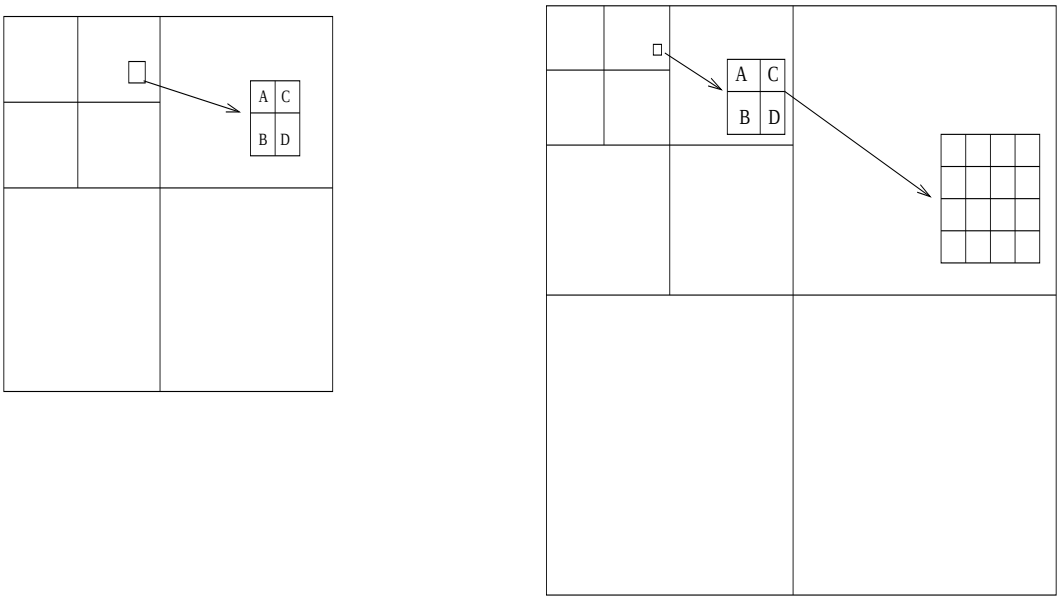


Fig. 3. The zerotree construction in the EZW algorithm

or diagonal), the dependencies can be well exploited by building a quadtree structure called "zerotrees", according to which any coefficient at a given band in Fig. 3 has four children coefficients at its lower-level subband (corresponding to the four-time larger subband image) in the same orientation. Suppose that the parent's position is (i,j) with i and j standing for the row and column number in its subband image, then its children's positions are (2i,2j), (2i,2j+1), (2i+1,2j), and (2i+1, 2j+1) in their corresponding subband image. There are two types of passes performed in EZW coding. The dominant pass finds significant coefficients (greater than a given threshold), and its following subordinate pass refines the magnitudes of all significant coefficients found in the dominant pass. A ZTR symbol (meaning "zerotree root") is used for an insignificant coefficient (less than a given threshold) that has no significant descendents. An Isolated Zero symbol (named IZ) is used when a coefficient is insignificant but has some significant descendents. Because a child coefficient is probably also insignificant when its parent coefficient is insignificant and thus many insignificant

coefficients can be represented as ZTRs, and furthermore a ZTR's descendents may be cut off from the final code stream, the use of ZTR and IZ symbols informs the locations of significant coefficients quite efficiently. Interested readers may refer to [13] for the details of zerotree coding algorithm.

2. Applying I/O Data Space Transform to Zerotree Construction

In the zerotree coding algorithm one first computes 2-D DWT and then constructs zerotree data structures on the results of wavelet transform. At each level of the 2-D separable DWT, the band is decomposed into L_r and H_r (where L_r is the result of low-pass row-wise filtering and H_r is the result of high-pass row-wise filtering); L_r is decomposed into LL and LH by low-pass and high-pass column-wise filters; H_r is decomposed into HL and HH by low-pass and high-pass column-wise filters. Subband LL is recursively decomposed in higher levels of transforms, as shown in Fig. 3. The subscript number of a subband in Fig. 3 refers to the wavelet transform level. For instance, LH_2 refers to the LH subband in the second-level wavelet transform. The dependence graphs in I/O data space and the nonlinear I/O data space transformations are illustrated in Fig. 4. According to Γ_5 in Section B of this chapter, the nonlinear I/O data space transformation for 2-D DWT is rewritten as below.

When $j = u + 1/2$ (u is supposedly a positive integer), L_r: $n_1 \longmapsto 2^{\lfloor j \rfloor} n_1, n_2 \longmapsto 2^{\lceil j \rceil} n_2, j \longmapsto j$; for H_r: $n_1 \longmapsto 2^{\lfloor j \rfloor} n_1, n_2 \longmapsto 2^{\lceil j \rceil}(n_2 + \frac{1}{2}), j \longmapsto j$; when j is an integer, for LL: $n_1 \longmapsto 2^{\lfloor j \rfloor} n_1, n_2 \longmapsto 2^{\lceil j \rceil} n_2, j \longmapsto j$; for LH: $n_1 \longmapsto 2^{\lfloor j \rfloor}(n_1 + \frac{1}{2}), n_2 \longmapsto 2^{\lceil j \rceil} n_2, j \longmapsto j$; for HL: $n_1 \longmapsto 2^{\lfloor j \rfloor} n_1, n_2 \longmapsto 2^{\lceil j \rceil}(n_2 + \frac{1}{2}), j \longmapsto j$; for HH: $n_1 \longmapsto 2^{\lfloor j \rfloor}(n_1 + \frac{1}{2}), n_2 \longmapsto 2^{\lceil j \rceil}(n_2 + \frac{1}{2}), j \longmapsto j$.

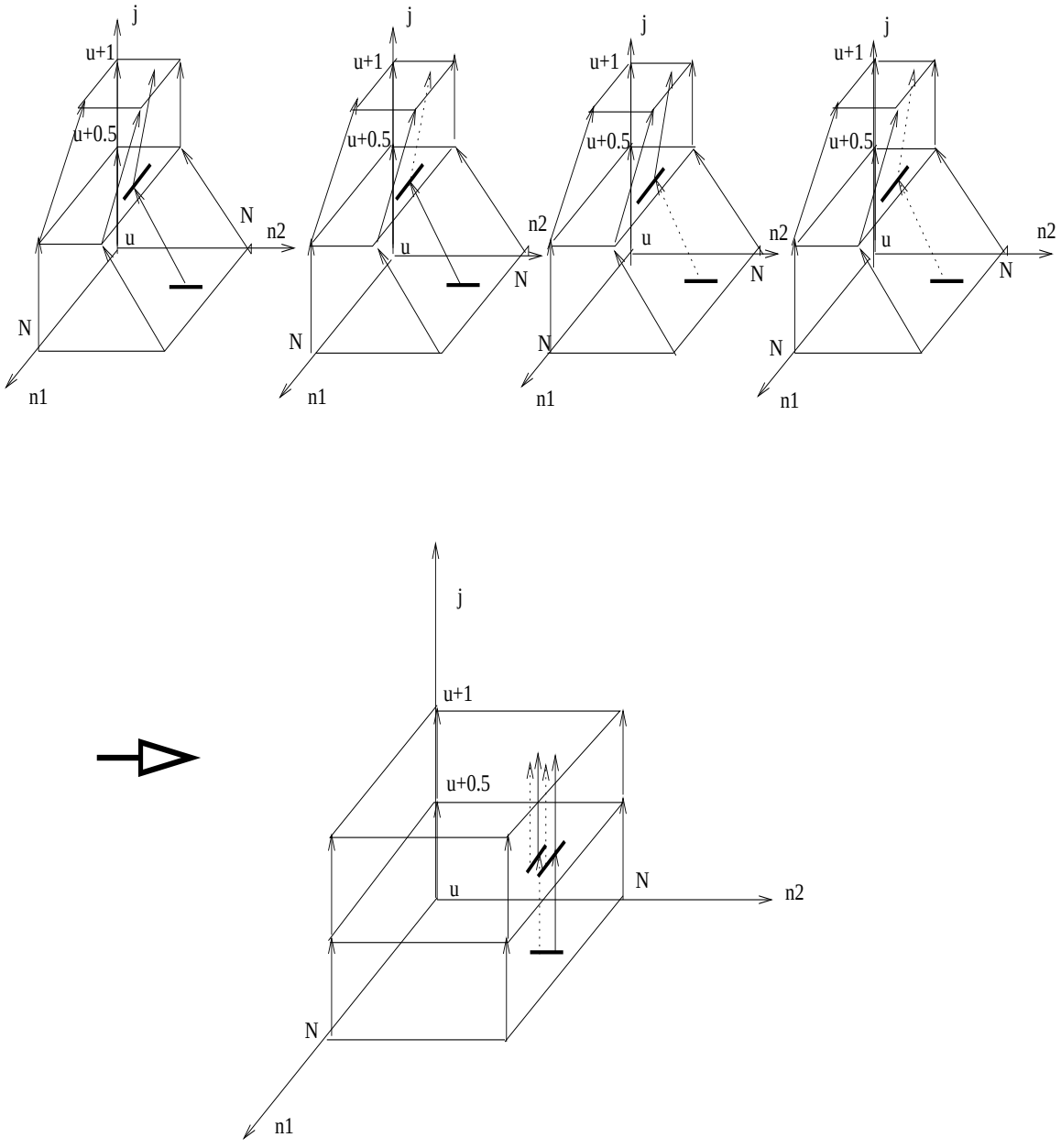


Fig. 4. The regularization of dependence graphs for zerotree construction

Considering the zerotree data structures among the results of wavelet transforms, we can find that the nonlinear I/O data space transformation Γ_5 relocate the parent-children relationships and put each parent and its children together in terms of their coordinates of n_1 and n_2 in the I/O data space. For instance, let there be a parent data item belonging to subband HL_2 with the coordinates of $(n_1 = a, n_2 = b, j = 2)$ in I/O data space before the nonlinear I/O data space transformation. According to the zerotree coding algorithm, it has four children belonging to subband HL_1 with coordinates of $(n_1 = 2a, n_2 = 2b, j = 1), (2a, 2b+1, 1), (2a+1, 2b, 1)$ and $(2a+1, 2b+1, 1)$ respectively. After the nonlinear I/O data space transformation is applied, the parent's new coordinates are $(4a, 4b + 2, 2)$, and the children's new coordinates are $(4a, 4b+1, 1), (4a+2, 4b+1, 1), (4a, 4b+3, 1)$ and $(4a+2, 4b+3, 1)$ respectively. In other words, the parent-children relationships are restricted in local domain if we get a projection of the dependence graphs in I/O data space along axis j.

To give a brief presentation, we suppose that there are two levels of wavelet transforms ($j \leq 2$ for the dependence graphs in I/O data space), and assume a row-major input format to scan a 2-D image and feed the image pixels to the system. Following the scheme of free schedule at the end of last section, we can assign the computation corresponding to each layer of dependence vectors in Fig. 4 to a separate processor, and have all processors perform calculations in parallel when the input data are fed to the system in real time. The particular challenge in architectural designs of wavelet zerotree coding algorithms stems from locating children given any parent data item. To avoid using RAMs for data rearrangement when constructing zerotrees and generating symbols as designated in the zerotree coding algorithm, we adjust the scheme of free schedule so that the calculation of any parent data and the calculation of its children are scheduled to be on the same time. In other words, we reorder the 2-D DWT computation so that the result of the DWT computation (as the outputs of

parallel processors) itself follows the zerotree structures. This has been made possible by nonlinear I/O data space transformations proposed in the chapter. When the input image pixels are fed to the system in a row-major order, the wavelet-adjacent fields on plane $j = 0$ are row-wise scanned, and the calculations corresponding to the dependence vectors between planes $j = 0$ and $j = 1/2$ are ready to be performed in the same row-wise order. Due to I/O data space transformation, the calculation results on each plane (orthogonal to axis j) above $j = 0$ can also be produced in row-wise major by parallel processors in real time. Since any parent data and its children on zerotree structure are put together via Γ_5 in terms of their coordinates of n_1 and n_2 in I/O data space, we can adjust the schedules of parallel processors only a little so that the children are calculated by a processor when their parent is calculated by another processor simultaneously. This adjustment is accomplished by using a small *Transpose Unit* in which only systolic data flow is allowed, and meanwhile a large off-chip RAM is avoided. The architectures are proposed in the following.

3. The Architecture for Rearranging 2-level 2-D DWT

We assume the width of wavelet filters is L and the size of 2-D input data is $N \times N$. For typical applications in practice, let $L = 9$ and $N = 512$ in the introduction of this subsection. We introduce a simple structure of wavelet filters whose architecture is detailed in [22]. Its structure is used in our design as a module of Processing Unit (PU) that computes wavelet filtering and decimation. As illustrated in Fig. 5 for a 9-point wavelet filter, it is made up of four registers, five multipliers and six adders. It rearranges the calculation of wavelet filtering such that the filter is cut to half taps based on the symmetry between the negative and positive wavelet filter coefficients. a, X and Y are the input sequence, low- and high-pass filtering output sequence respectively. While a datum of input sequence a is fed and shifted into

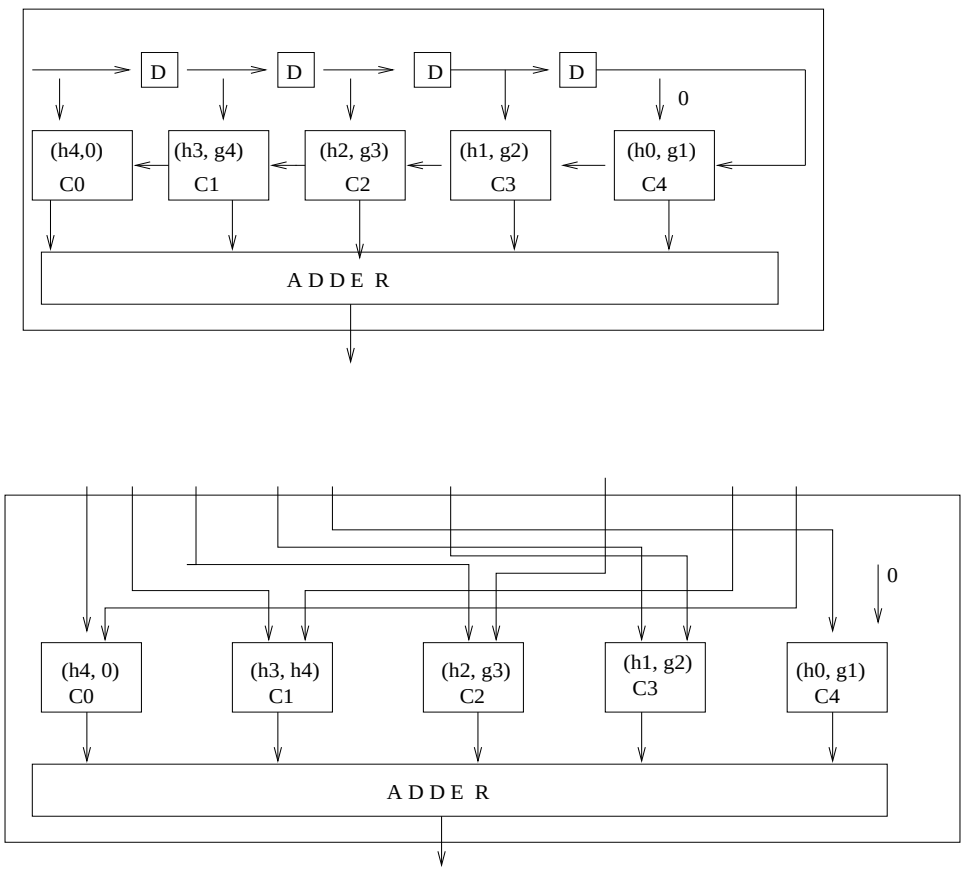


Fig. 5. The systolic and parallel wavelet filters integrating low-pass and high-pass filtering

the PU every clock cycle, a datum of X is calculated every even clock cycle and a datum of Y is calculated every odd clock cycle. Such calculations are possible because of the wavelet dyadic downsampling. The connections between computation units (multipliers or adders) are restricted local. The PU in Fig. 5 is extended to a parallel format as illustrated in Fig. 5 where if a number of data from sequence a a_{k+8}, a_{k+7}, ..., a_k are fed to the PU in parallel at a certain clock cycle, then a_{k+9}, a_{k+8}, ..., a_{k+1} are fed at the next cycle. The calculations of X and Y are the same as in Fig. 5. The PU actually takes a wavelet-adjacent field as the input.

We propose a module called Transpose Unit (TU) that can partially transpose a

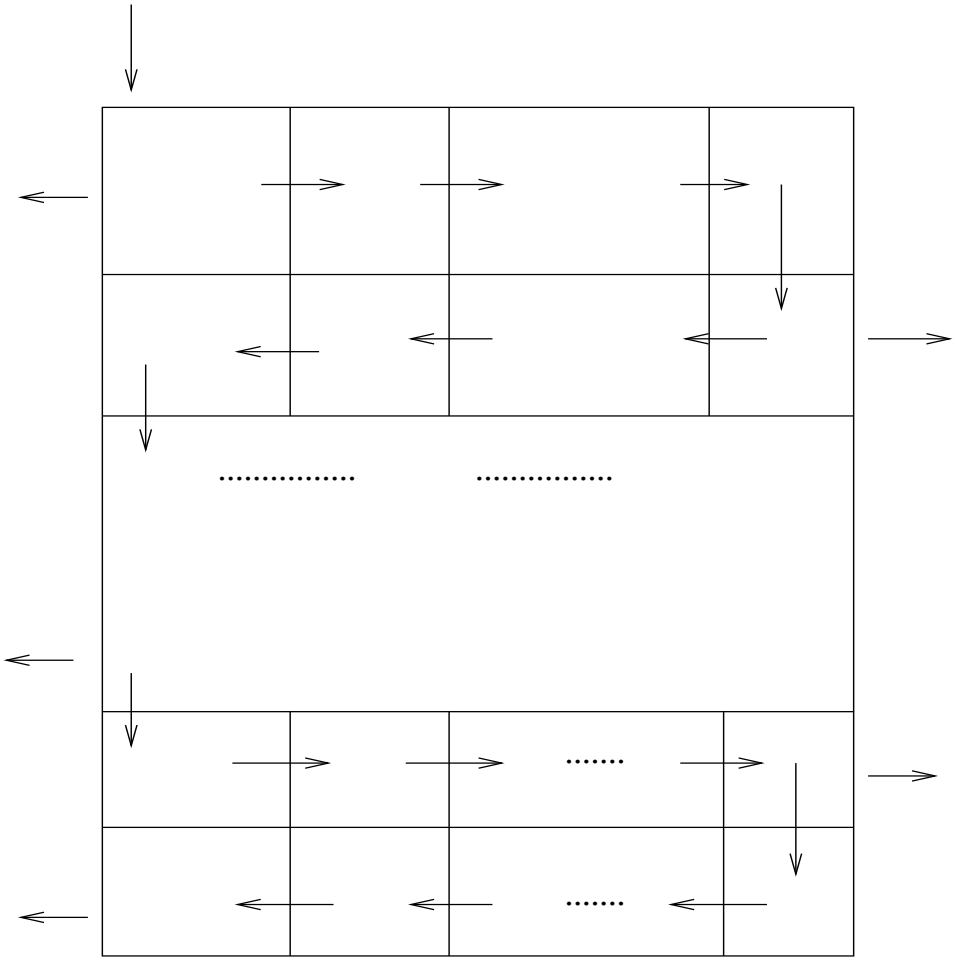


Fig. 6. The architecture of Transpose Unit (TU)

matrix on the fly. When a matrix is input to the TU in a row-wise indexing way with one element per clock cycle, the TU gives out the elements in a partially column-wise indexing way as explained in the following. The structure of TU is illustrated in Fig. 6. It is made up of $(L + 4)$ or 13 concatenated modules of First In First Out (FIFOs) [28][29], with each FIFO having N or 512 cells. The top FIFO takes as its input the row-wise indexed matrix with one element per clock cycle. Because the distance between the outputs of all FIFOs Y_i and Y_{i+1} (0<i<(L+3)) is always N, the TU's 13 outputs belong to the same column in the input matrix. The FIFOs transfer

the input data step by step in clock cycles, and meanwhile provide a new group of column-wise data (i.e., a new column-wise wavelet-adjacent field) to the next wavelet filter in each clock cycle.

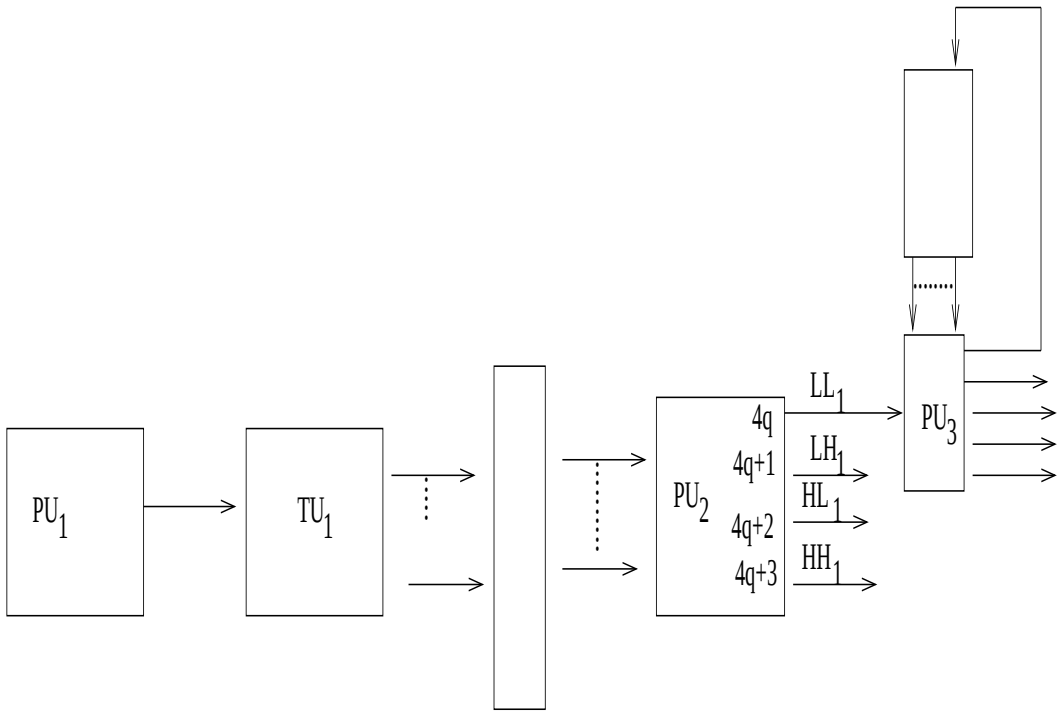


Fig. 7. The architecture for zerotree construction

Keeping in mind the dyadic downsamplings in both row-major and column-major filtering of 2-D DWT. In the following, we present our design that rearranges the computation of the DWT so that a parent and its children are calculated at the same time. The structure for the non-RAM-based zerotree construction is proposed in Fig. 7.

The coefficients generated in the first decomposition level can be separated into groups each of which contains four coefficients having the same parent (which is generated in the next decomposition level). Now we have the restriction that these four sibling coefficients be calculated together for the purpose of calculating children

and parent simultaneously. More exactly, we calculate the four siblings consecutively at a rate with one coefficient per clock cycle, and at the same time generate their parent coefficient at a four-time less rate via another output port. Due to the row-major dyadic downsampling, the row-major high/low-pass filtering is alternatively executed by PU_1 in Fig. 7 point by point in each row. Based on similar column-major dyadic downsampling, PU_2 takes turns to execute column-major high/low-pass filtering.

PU_2 selects appropriate inputs from TU_1 to generate four sibling coefficients consecutively. The order to calculate the siblings is as A, then B, C and D in the example of four siblings illustrated in Fig. 3. After PU_2's calculation of point A by taking Y_1, ..., Y_L as inputs, PU_2 has to calculate B by taking Y_3, ..., Y_{L+2} as inputs, then PU_2 comes back to take Y_1, ..., Y_L to calculate C, then PU_2 takes Y_3, ..., Y_{L+2} again to calculate D. This is for the case that A, B, C and D are in the column-major low-pass subband (HL in Fig. 3). If they are in the column-major high-pass subband (LH or HH in Fig. 3), PU_2 will alternate using Y_2, ..., Y_{L+1} and Y_4, ..., Y_{L+3} as inputs and take turns on the calculations in similar ways. Note that PUs consume one clock cycle for each calculation and the TU cells consume one clock cycle to transfer a datum so that the above calculations and the corresponding data transfers are performed step by step. In a similar way, PU_3 takes turns generating the parents as the outputs of the second level of DWT. In summary, any four siblings are always generated in turns by PU_2 and their parent is calculated by PU_3 at the same time. The control signal for the switch DM is internal and simple. There is neither external nor complex control for any device in Fig. 7. Only the clock signal is global to synchronize the system. We call this scheme of internal control as "self-controlled" device. Fig. 8 has demonstrated an example that a multiplexer selects data based on internal control signals which are generated from the input clock periodically. It is

the rearranged periodical operations for zerotree construction derived from our novel nonlinear I/O data space transformations in Section B of this chapter that make such designs of "self-controlled" devices possible.

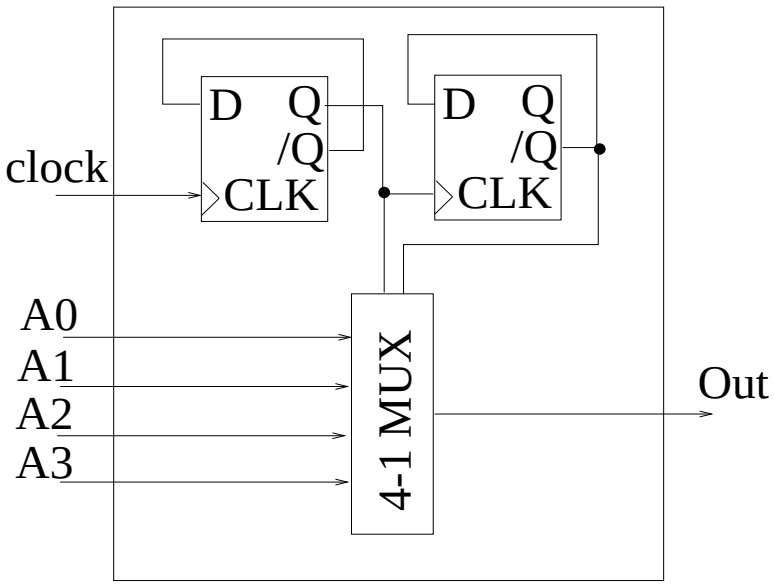


Fig. 8. An example of local-controlled 4-1 multiplexer

4. An Arbitrary Number of Levels in Zerotree Construction Systems

In this subsection we extend our design to general cases where any levels of wavelet decompositions are possible. All coefficients in the first level and in the intermediate levels which correspond to the same ancestor in the last level decomposition have to be calculated together to satisfy the restriction that any parent and its children be calculated simultaneously. As in the former subsection, by the word "together" we mean successively calculating the children at a four-time higher frequency than their parent and simultaneously outputting children and the parent via two ports respectively.

Because the input image is fed into the system in the same way as before, the

first level row-major high/low-pass filtering is still performed in PU_1 alternatively. Regarding the first level column-major high/low-pass filtering performed in PU_2, we note that there are 4^{m-1} coefficients in the first level decomposition corresponding to the same ancestor in the last level (level m) decomposition. To satisfy the restriction of generating parent and children simultaneously, it is required that these 4^{m-1} "kindred" coefficients be calculated together. Meanwhile these coefficients' parents in the intermediate levels of decomposition should be calculated together too. Note that these 4^{m-1} coefficients are located in 2^{m-1} adjacent rows and 2^{m-1} adjacent columns in their subband. PU_2 should alternatively select appropriate inputs among 2^{m-1} different groups of parallel column-major data from TU_1, and perform column-major filtering to generate the 4^{m-1} kindred coefficients in turns, where the coefficients calculated with the same group of input belong to the same row (see the next paragraph for instance). Accordingly, TU_1 is an extended version in Fig. 6 and is supposed to have output ports Y_1, ..., Y_{L+M} with M equal to 2^m, so TU_1 has (L+M)×N cells with a structure similar to Fig. 6.

For instance, there are sixteen "kindred" coefficients in HL_1 subband illustrated in Fig. 3 to be calculated successively in the first level of the three-level DWT. The order to calculate these coefficients is from A1 to A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, D1, D2, D3, and at last D4. This calculation order is according to the requirement that siblings be calculated successively, e.g., the siblings A1, A2, A3 and A4 should be calculated as a group (meanwhile their parent A is calculated in PU_3). Four parallel column-major data are selected among TU_1's output ports Y_1, ..., Y_{L+8} as: Y_1, ..., Y_L (named $\mathbf{E}_1$ for brevity); Y_3, ..., Y_{L+2} (named $\mathbf{E}_2$); Y_5, ..., Y_{L+4} (named $\mathbf{E}_3$); and Y_7, ..., Y_{L+6} (named $\mathbf{E}_4$). Based on the systolic data transfer in TU_1 and column-major dyadic subsamplings in PU_2, PU_2 first takes $\mathbf{E}_1$ as input to calculate A1, then uses $\mathbf{E}_2$ for calculating A2 in next clock cycle; after that, comes back to take

$\mathbf{E}_1$ for A3, then $\mathbf{E}_2$ for A4. PU_2's following operations are: $\mathbf{E}_3$ for B1, $\mathbf{E}_4$ for B2, $\mathbf{E}_3$ for B3, $\mathbf{E}_4$ for B4, then $\mathbf{E}_1$ for C1, $\mathbf{E}_2$ for C2, and etc.

Now we analyze the operations in PU_3. PU_3 carries out the rest computation in DWT. The second level decomposition is achieved as follows. In the first quarter of the period when 2^m rows of input image are fed to the system, PU_3 gets its inputs, i.e., the coefficients in LL_1 subband (see Fig. 3(d)) from TU_1, and alternatively performs the second level low/high-pass row-major convolution. The calculated results, or the coefficients in Lr_2 and Hr_2 are stored in two TUs in PU_3's feedback block. In the second quarter, the Lr_2 coefficients are fed back to PU_3 to be column-major filtered to get the results in LL_2 and LH_2. In the third and fourth quarter, the Hr_2 points are fed back to PU_3 to be used to calculate out HL_2 and HH_2 respectively. There are some idling intervals during PU_3's performing the second level transform due to less computation in the second level of DWT. Thus we can insert the computation of further levels of decomposition into those available intervals. For example, once an LL_2 point is generated in the second quarter, it will be fed back to PU_3 via the feedback block to be row-major filtered in available intervals. Then the generated Lr_3 and Hr_3 coefficients are also stored in TUs in PU_3's feedback block. In next available intervals Lr_2 and Hr_2 are fed to PU_3 for column-major convolution to calculate LH_3, HL_3 and HH_3 coefficients. Due to the exponentially decreasing number of parents, PU_3 can similarly proceed to more levels of decomposition in the intervals between lower-level calculations.

CHAPTER III

EXPLOITING INTRA-ITERATION PARALLELISM IN MULTIDIMENSIONAL MULTIRATE SYSTEMS

A. Background Introduction

An MD Data Flow Graph (MDFG) ([30][31][32][33][34][35][36]), G=(V, E, d, t), is a node-weighted and edge-weighted directed graph modeling an MD DSP algorithm. V is the set of computation nodes. E$\subset$ V$\times$V is the set of edges representing the data flows and dependencies between nodes. d is the set of delay-weights (n-component vectors) on E (each edge is associated with an n-component vector as its delay-weight) and represents the MD delays of data flowing between two nodes, with n being the number of dimensions of the algorithm. t is the set of computation times (in clock cycles) for the computation nodes. An *iteration* is the execution of a loop body exactly once, i.e., executing the task corresponding to each node in V exactly once. By replicating an MDFG at multi-dimensionally indexed positions, we expand an *iteration space*, where each MDFG, excluding the edges with delay vectors different from (0, 0, $\cdots$, 0), is taken as a *cell* indexed by Cartesian coordinates. Those non-zero delay weighted edges within the MDFG give specifications of the dependencies between these cells in the iteration space. Due to the causality, a legal MDFG must have no zero-delay cycle, i.e., the summation of the delay vectors along any cycle path in the MDFG can not be (0, 0, $\cdots$, 0) [37][38][39][40]. An example of an MDFG for a 2-D algorithm and the corresponding iteration space is illustrated in Fig. 9. It is an example introduced in [41].

A multi-dimensional retiming operation on a node u $\in$ V redistributes the nodes in the cells in iteration space. The retiming vector r(u) of a node u $\in$ V represents the

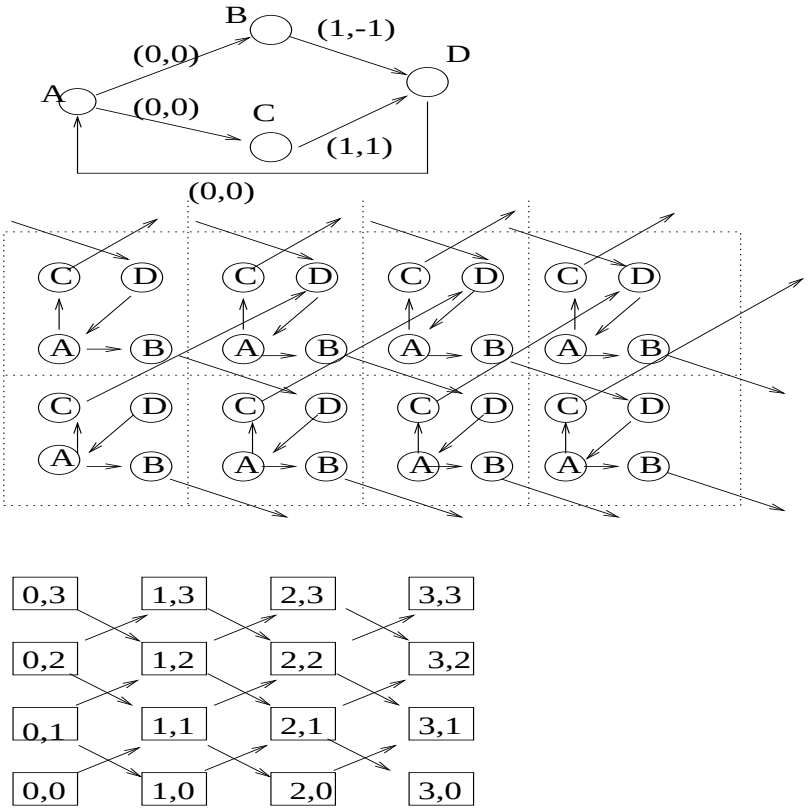


Fig. 9. A simple example of an MDFG and the corresponding iteration space

offset vector between the original cell containing u, and the one after retiming. To preserve dependencies in iteration space after retiming operations, delay-weights of edges change accordingly. Formally, for edge $e : u \rightarrow v$, we have the *retiming equation* [32]

$$d_r(e) = d(e) + r(u) - r(v) \tag{3.1}$$

where d(e) or d_r(e) is the delay-weight of edge e after or before retiming respectively. After retiming, an instance of node u in the cell indexed by **i** in iteration space is moved to cell **i-r(u)**. Obtaining full inter-operation parallelism is equivalent to obtaining non-zero delays on all edges of the MDFG by retiming techniques such

that the computation tasks corresponding to all nodes in the retimed MDFG can be executed simultaneously. An example of MD retiming is shown in Fig. 10 (a 2-D case), where r(A)=r(B)=r(C)=(0,0) and r(D)=(0,1). Fig. 10 is an example introduced in [41]. The delay-weights on the edges outgoing from a computation node in the MDFG corresponds to the necessary storage of the data outgoing from this computation node to its following computation nodes. Therefore, another key purpose of the retiming technique is to minimize the system storage requirement by modifying the delay-weights on edges in the MDFG.

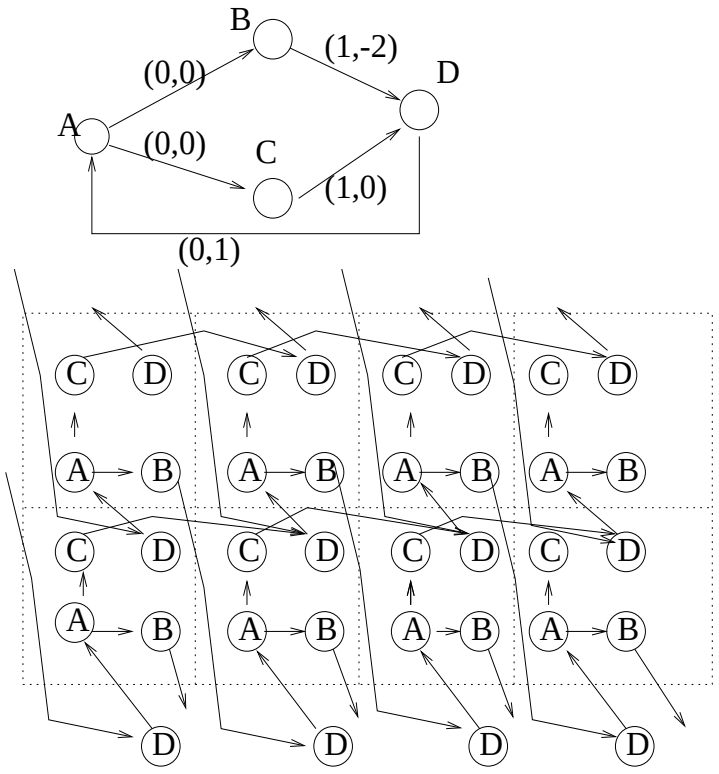


Fig. 10. The retiming operations on MDFG and iteration space

B. Modeling MD Multirate Algorithms in MR-MDFG's

Multirate DSP algorithm descriptions contain decimators and/or expanders to represent multirate data flows[33][42][43]. These decimators and expanders can be modeled by *multirate-weighted edges*[44][45][46][47][48]. We formulate the Multirate MDFG (MR-MDFG) as: G=(V, E, M, d, t), which is a node-weighted and edge-doubly-weighted directed graph that is used to model a multirate MD DSP algorithm. V is the set of computation nodes. E$\subset$ V$\times$V is the set of edges representing the data flows and dependencies between nodes. d is the set of offset-weights (n-component vectors) on E (each edge is associated with an n-component vector as its offset-weight) with n being the number of dimensions of the algorithm. M is the set of multirate-weights (n by n matrices) on E (each edge is associated with an n by n matrix as its multirate-weight). t is the set of computation times (in clock cycles) for the computation nodes.

Consider an edge e: $u \rightarrow v$ connecting two computation nodes u and v in the MR-MDFG. The dependence relationship between data streams at both sides of edge e in the MR-MDFG is represented by:

$$P(m) \leftarrow Q(M(e) \times m - d(e)) \tag{3.2}$$

where P is the data stream flowing out from node v and Q is the data stream flowing out from node u, M(e) and d(e) are the multirate-weight and the offset-weight of e respectively, and m is the index vector of data stream P. Fig. 11 shows an example of MR-MDFG (2-D case), and its corresponding multirate MD DSP algorithm is as the following.

For $m_1 = 1$ to M_1

  For $m_2 = 1$ to M_2

    $C(m_1, m_2) = A(2m_1 - 4, m_2 - 3) + B(2m_1 - 5, m_2 - 6);$

$$D(m_1, m_2) = 3 \times C(m_1 - 8, 3m_2 - 1);$$
$$F(m_1, m_2) = D(m_1 - 3, m_2 - 7) + B(2m_1 - 2, 3m_2 - 5);$$

In order to illustrate the dependence relationship at both sides of an edge in the MR-MDFG with Eq. (3.2), we take edge E3 in Fig. 11 as an example. Computation node Y, a multiplier, corresponds to the multiplication in statement Y of the DSP algorithm. Node Y takes data stream C as the input, and computes data stream D as the output. Consider an output data item $D(15, 3)$ with $m = (15, 3)'$ as the index vector. According to Eq. (3.2), the dependence relationship between computation nodes Y and X is specified by edge E3, i.e., $D(15, 3) \leftarrow C(M(E3) \times m - d(E3)) = C((1 * 15, 3 * 3) - (8, 1)) = C(7, 8)$.

The multirate-weights are n by n matrices (where n is the number of dimensions) and only diagonal elements can be non-zero values. If all diagonal elements of a multirate-weight are equal to 1, the corresponding edge is equivalent with a traditional single-rate edge. If all edges in an MR-MDFG are of single-rate, the MR-MDFG is equivalent with a traditional single-rate MDFG. Applying traditional retiming vectors r(U) and r(V) on the nodes of U and V, we have the new offset-weight of e: U $\rightarrow$ V as

$$d_r(e) = d(e) + M(e)r(U) - r(v) \tag{3.3}$$

Definition 3.1 In an MR-MDFG, the *rate of a path P*, Z(p), is defined as $Z(p) = \prod_i M(e_i)$, where $\{e_i\}$ represents all edges along path p.

The *sources* of an MR-MDFG are those nodes connecting with the data stream(s) of system input. Without loosing generality, we assume that each computation node in an MR-MDFG can have at most two input edges to receive operands (otherwise we could add some auxiliary computation nodes and edges to satisfy this assumption).

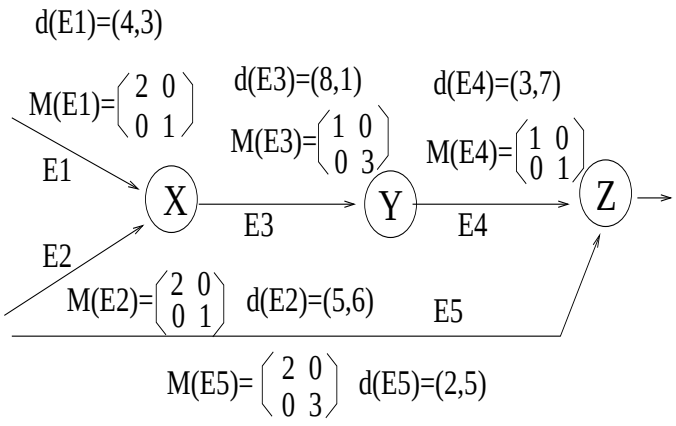


Fig. 11. An example of MR-MDFG

The *internal computation nodes* of an MR-MDFG are those nodes at which some edges end and from which some edges start. The *sinks* of an MR-MDFG are those nodes from which no edges start.

Definition 3.2 The rate of an edge $e : u \rightarrow v$, $R(e)$, is defined as $R(e) = MAX[Z(p)]$ where the maximum is taken over all paths from the sources of the MR-MDFG to v and going through edge e.

Definition 3.3 We define an MR-MDFG as a *rate-balanced* MR-MDFG if, for any internal computation node in the MR-MDFG, the rates of this node's two input edges are equal. Otherwise, the MR-MDFG is defined as *rate-conflict*. Single-rate MDFGs can be taken as a special class of rate-balanced MR-MDFGs where multirate-weights are all 1's. It can be shown that the MR-MDFGs for many generally used multirate DSP algorithms such as all wavelet-based algorithms and most Partial Differential Equation problems are rate-balanced.

Lemma 3.1 If and only if an MR-MDFG is rate-balanced, for any edge in the MR-MDFG, e.g., e: $u \rightarrow v$, we have $R(e) = Z(p)$, where p is ANY path from a source of the MR-MDFG to node v and goes through edge e, i.e., all $Z(p)$ are equal for ANY path p that starts from a source of the MR-MDFG to v and goes through e.

Proof:

1) Assume that an MR-MDFG is rate-balanced and there exists a path P_1 starting from one of the sources of the MR-MDFG with e as the last edge such that $R(e) \neq Z(P_1)$. According to *Definition 3.2*, there is at least a path P_{max} starting from one of the sources of the MR-MDFG with e as the last edge such that $Z(P_{max}) = R(e)$. So we have $Z(P_{max}) > Z(P_1)$. As shown in Fig. 12, since P_{max} and P_1 are different but have the same destination node, we can find a node called U_1 as their branch node. Let e_{A1} and e_{B1} be the two input edges of node U_1. Path P_{max} goes through edge e_{A1} and path P_1 goes through edge e_{B1}. Based on *Definition 3.1* and *Definition 3.3*, we have $R(e_{A1}) = R(e_{B1})$ implying that $Z(P_{11}) < Z(P_{max1}) = R(e_{B1}) = R(e_{A1})$, where path P_{max1} starts from one of the sources of the MR-MDFG and with e_{B1} as the last edge, and $Z(P_{max1})$ is equal to $R(e_{B1})$. Path P_{11} is the part of P_1 that ends at U_1. Since paths P_{11} and P_{max1} are different but have the same destination node (U_1), we can find a node called U_2 as their branch node. e_{A2} and e_{B2} are the two input edges of node U_2. Path P_{max1} goes through edge e_{A2} and path P_{11} goes through edge e_{B1}. Along path P_1, we can find in Fig. 12 $P_{max2}, P_{12}, U_3, e_{A3}, e_{B3}, P_{max3}, P_{13}, \cdots$ recursively, until we reach node U_t that is right after the *source* (called S_1) of the MR-MDFG on path P_1. Let the edge between U_t and S_1 be e_{Bt}. Similarly, path P_{1t} is the part of P_1 that ends at U_t (Here it is the same as e_{Bt}). Based on the recursive implications described above, we have $Z(P_{1t}) < R(e_{Bt})$. It must be false because it is contradict to *Definition 3.1*. Therefore, the assumption at the beginning of this paragraph can never be true. In other words, if an MR-MDFG is rate-balanced, the property mentioned in *Lemma 3.1* must be true.

2) Assume that an MR-MDFG is rate-conflict. According to *Definition 3.3*, in the MR-MDFG exists at least an edge e: u→v such that $R(e_A) \neq R(e_B)$, where e_A and e_B are the two input edges of node u. Suppose M(e) is the multirate-weight on

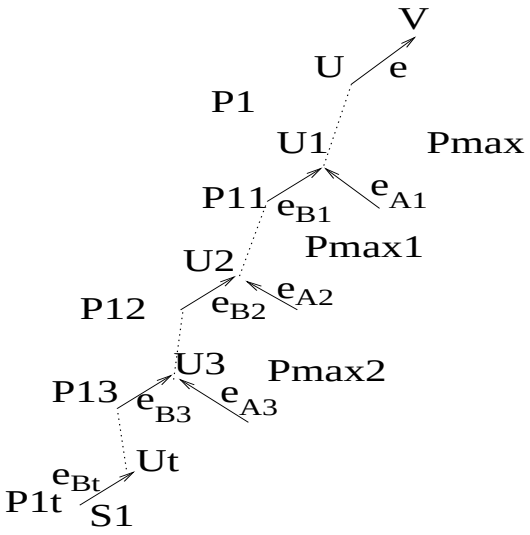


Fig. 12. An example showing the proof of lemma 3.1

edge e. Because of *Definition 3.2*, $R(e)$ is equal to either M(e)×$R(e_A)$ or M(e)×$R(e_B)$, i.e., there exists a path P from one of the sources of the MR-MDFG to node v, going through either e_A or e_B, such that $R(e) \neq Z(P)$.

Lemma 3.1 is critical to prove the following *Theorem 3.1* and *Theorem 3.2*, which are the foundation for *Procedure 3.1: MD intercalation* that is to be introduced in Section C of this chapter.

In a rate-conflict MR-MDFG, an edge e: $u \rightarrow v$ is called a *rate-conflict edge* if $R(e_A) \neq R(e_B)$, where e_A and e_B are the two input edges of node u. Suppose P_A is a path from one of the sources of the MR-MDFG to node u such that $Z(P_A) = R(e_A)$. Suppose P_B is a path from one of the sources of the MR-MDFG to node u such that $Z(P_B) = R(e_B)$. We call P_A and P_B the *primary paths* of the rate-conflict edge e.

Definition 3.4: If in a rate-conflict MR-MDFG there is no such rate-conflict edge that at least one of its primary paths goes through itself, the MR-MDFG is called a *branch-type rate-conflict MR-MDFG*; otherwise the MR-MDFG is called a *cycle-type*

rate-conflict MR-MDFG, as illustrated in Fig. 13.

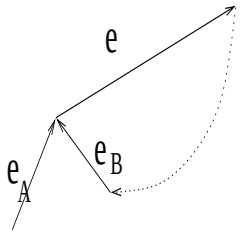


Fig. 13. The cycle-type rate-conflict MR-MDFG

Definition 3.5: *The delay of a path p*, $d(p)$, is defined as $\sum_{i=1}^{K}[(\prod_{j=0}^{i-1} M(e_j))d(e_i)]$, where $M(e_0)$ is assumed to be 1, and $e_1, e_2, \cdots$, and e_K are all K edges along p from the beginning to the end.

Definition 3.5 is inspired by concatenating the dependence relations specified by all edges on path p according to Eq. (3.2). Similarly, we define the delay of a path p *after retiming operations* $d_r(p)$ by $\sum_{i=1}^{K}[(\prod_{j=0}^{i-1} M(e_j))d_r(e_i)]$, where $d_r(e_i)$ is the retimed offset-weight on edge e_i. *The increment of the delay of a path p after retiming operations* $\Delta d(p)$ is $d_r(p) - d(p)$.

Lemma 3.2: Such relation always exists for the retiming operations along any path p in the multirate-MDFG: $\Delta d(p) = r(u) - R(p)r(v)$, where u and v are respectively the starting node and the destination node of path p.

Proof:

Based on Eq. (3.3), in the formula of $d_r(p)$ (*Definition 3.5*) we can replace $d_r(e_i)$ by $d(e_i) + r(u_{i-1}) - M(e_i)r(u_i)$ for $i = 1, 2, 3, \cdots, K$, where $u_0, u_1, u_2, \cdots, u_K$ are the nodes along path p from the beginning to the end. After simplifying the formula of $d_r(p)$, we get $d_r(p) = d(p) + r(u_0) - Z(p)r(u_K)$. Letting u_0 and u_K be u and v respectively, we have derived the above conclusion.

An MR-MDFG is called *stable for retiming* if in an MR-MDFG, given any two nodes u and v and two corresponding retiming vectors r(u) and r(v), we have $\Delta d(P) = C$ for any path P from u to v where C is a constant value independent of P.

Theorem 3.1: 1) All rate-balanced MR-MDFGs are stable for retiming; 2) If in a branch-type rate-conflict MR-MDFG there exists a rate-conflict edge whose primary paths have more than one common node, the MR-MDFG is not stable for retiming; 3) None of the cycle-type rate-conflict MR-MDFG is stable for retiming.

Proof:

1) Suppose u and v are any two nodes in a rate-balanced MR-MDFG. Assume two different paths P_A and P_B from u to v. Let P_0 be the path from one of the sources of the MR-MDFG to u. Since P_A and P_B have the same destination node (i.e., v), we can find a node U_0 such that P_A and P_B respectively go through edges e_A and e_B (the two input edges of node U_0), and path $U_0 \rightarrow v$ (called P_C) is the common part of P_A and P_B. Thus we have $P_A = P_{A1} + P_C$ and $P_B = P_{B1} + P_C$ where paths P_{A1} and P_{B1} are respectively another part of P_A and P_B. Finally, we conclude that $R(e_A) = R(e_B)$ (per *Definition 3.3*) $\Longrightarrow Z(P_0 + P_{A1}) = Z(P_0 + P_{B1})$ (per *Lemma 3.1*) $\Longrightarrow Z(P_0) \times Z(P_{A1}) = Z(P_0) \times Z(P_{B1})$ (per *Definition 3.1*) $\Longrightarrow Z(P_{A1}) = Z(P_{B1})$ $\Longrightarrow Z(P_{A1}) \times Z(P_C) = Z(P_{B1}) \times Z(P_C) \Longrightarrow Z(P_A) = Z(P_B)$. This leads to the conclusion that all rate-balanced MR-MDFGs are stable for retiming if *Lemma 3.2* is considered.

2) In a branch-type rate-conflict MR-MDFG where there exists a rate-conflict edge e: $v \rightarrow v_0$ whose primary paths P_A and P_B have more than one common nodes, we can assume a node u (which is different from v) as one of the common nodes of P_A and P_B. P_A goes through e_A and P_B goes through e_B, where e_A and e_B are the two input edges of node v. If u is a source of the MR-MDFG, based on *Lemma*

3.2, we simply draw the conclusion that the MR-MDFG is not stable for retiming because $Z(P_A) \neq Z(P_B)$. If u is not a source of the MR-MDFG, we suppose that $P_{A1} : u \rightarrow v$ and $P_{B1} : u \rightarrow v$ are respectively a part of P_A and P_B, and P_{A2} and P_{B2} are respectively another part of P_A and P_B. Since $R(e_A) = Z(P_A)$, we have $Z(P_A) \geq Z(P_{A1+B2})$ based on *Definition 3.2*, where P_{A1} is the first part and P_{B2} is the second part of path P_{A1+B2}. Thus we have $Z(P_{A1}) \times Z(P_{A2}) \geq Z(P_{A1}) \times Z(P_{B2})$ $\Longrightarrow Z(P_{A2}) \geq Z(P_{B2})$. Similarly considering edge e_B, we also have $Z(P_{B2}) \geq Z(P_{A2})$. So $Z(P_{A2}) = Z(P_{B2})$, implying $Z(P_{A1}) \neq Z(P_{B1})$. Thus the MR-MDFG is not stable for retiming based on *Lemma 3.2*.

3) Suppose there is a rate-conflict edge e: $u \rightarrow v$ whose primary paths are P_A and P_B in a cycle-type rate-conflict MR-MDFG, and P_A goes through e. Let the starting node of P_B be s, a source of the MR-MDFG. Because P_A is different from P_B, we have two paths from s to u: P_B and $(P_A + P_B)$ as shown in Fig. 13. Apparently $Z(P_B) \neq Z(P_B + P_A)$. Thus the MR-MDFG is not stable for retiming based on *Lemma 3.2*.

The property of the retiming stability described in *Theorem 3.1* is very important in retiming an MR-MDFG. We can use the constraint of retiming legality to explain the importance of the retiming stability. When performing the operations of retiming on an MR-MDFG, we are under the restriction of retiming legality, i.e., the restriction that no dependence relationships between computation nodes in an MR-MDFG should be changed after the transformation, otherwise the function of the DSP algorithm represented by the MR-MDFG is modified. The delay of a path (d(p)) reflects the dependence relationship between two computation nodes at both ends of the path. If for a couple of nodes in an MR-MDFG, retiming operations result in different increments of path delay along different paths between the two nodes, (in

other words, the MR-MDFG is not stable for retiming), the constraint of retiming legality will not be observed since the path delays after retiming are different for different paths between the two nodes and thus the dependence relationship between these two nodes cannot be reserved. However, this important property of retiming stability has never been mentioned in literature on the MD retiming to the best of our knowledge. The reason is that all single-rate MDFGs are stable for retiming (immediately from *Theorem 3.1* if the single-rate MDFGs are taken as rate-balanced MR-MDFGs where the multirate-weights are unit matrices), and the previous researches on retiming deal with single-rate MDFGs only. According to *Theorem 3.1*, not all MR-MDFGs are stable for retiming.

Definition 3.6: In an MR-MDFG, a *rate-identical cut* is a set of multirate-weighted edges whose rates are identical.

Theorem 3.2: If and only if an MR-MDFG is rate-balanced, such property exists: when we cut off all multirate-weighted edges (i.e., those edges whose multirate-weights are not unit matrices), and obtain subgraphs each of which is a single-rate MDFG, all those multirate-weighted edges in the original MR-MDFG that are pointed directly into the same subgraph belong to a rate-identical cut.

Proof:

1) Suppose that an MR-MDFG is rate-balanced, and there are two edges e_A and e_B with different rates directly pointed into the same subgraph G_1. Let u and v be the nodes in G_1 that are directly connected to e_A and e_B respectively. Because G_1 is a connected graph, we can find a node t in it such that there are two paths P_A and P_B from the sources of the original MR-MDFG going through e_A and e_B respectively and having t as the same destination. Since all edges in G_1 are of single-rate, $Z(P_A)$ and $Z(P_B)$ are equal to $R(e_A)$ and $R(e_B)$ respectively. So we have $Z(P_A) \neq Z(P_B)$, which is contradictory to *Lemma 3.1*.

2) If an MR-MDFG is rate-conflict, we can find a rate-conflict edge e: $u \rightarrow v$ and let its primary paths be P_A and P_B. Suppose u is in subgraph G_1 if all multirate-weighted edges in the MR-MDFG are cut off. Now we claim that there exist two multirate-weighted edges with different rates in the original MR-MDFG pointed directly into G_1. Otherwise, any two different paths from the sources of MR-MDFG to node u would have the same rate because all edges in G_1 are of single-rate.

For clear presentation later, we define that each node in a rate-balanced MR-MDFG has a rate equal to the rate of its input edges, noticing that this is well-defined in rate-balanced MR-MDFGs because the rates of two input edges for any computation node are the same. After all multirate-weighted edges are cut off from the original MR-MDFG, each of the obtained subgraphs is a single-rate MDFG. Thus, the rates of all nodes in each of such subgraphs are identical. We furthermore define that such subgraph has a rate equal to the rates of all nodes in it.

Corollary 3.1: If an MR-MDFG is rate-balanced, such property exists: when we cut off all multirate-weighted edges (i.e., the multirate-weights are not unit matrices) from this MR-MDFG to get subgraphs each of which is a single-rate MDFG, all the multirate-weighted edges in the original MR-MDFG that outgo from the same subgraph belong to a rate-identical cut.

Proof:

Immediately following the conclusion of *Theorem 3.2* that all the multirate-weighted edges in the original MR-MDFG which are pointed directly into the same subgraph have the same rate, and the fact that every subgraph is of single-rate.

C. The MD Intercalation

The previous section gives a general analysis on the theory of MR-MDFG-represented MD multirate DSP algorithms, where the difference between rate-balanced MR-MDFGs and rate-conflict MR-MDFGs are defined. Some desirable properties of the rate-balanced MR-MDFG are analyzed and claimed which will be exploited in the following sections. In this section we propose the concept and the technique of *MD intercalation* based on *Theorem 3.2*. The MD intercalation is used to adapt the rate-balanced MR-MDFG's to be feasible for MD retiming such that appropriate analyses on retiming legality, memory requirement and arbitrary input format will be allowed in the next section.

In an iteration space expanded by replicating an MDFG as in Fig. 9, the cell dependence vectors are designated by the non-zero offset weights of the edges within the MDFG. The lengths of these dependence vectors correspond to the storage requirement in hardware systems: the data (whose amount is evaluated by the length of corresponding cell dependence vectors) outgoing from a computation node in the MDFG needs to be stored and later (depending on the value of the offset-weight) consumed by another computation node which is located in a cell that is pointed by a dependence vector in iteration space.

Lemma 3.3: The cell dependence vectors in the iteration space that correspond to the multirate-weighted edges in an MR-MDFG have lengths of $O(N)$ in the iteration space, where N is the input size of the DSP algorithm represented by the MR-MDFG.

Proof:

Assume an edge e: $U \rightarrow V$ with the offset-weight d(e) and the multirate-weight M(e) in the MR-MDFG. Suppose that a cell dependence vector in the iteration space corresponding to e is from the cell including U (indexed by S) to the cell including V

(indexed by T). According to Eq. (3.2) and the concept of the cell dependence vector, the length of this cell dependence vector should be $|S - T| = |S - M(e)S + d(e)| = O(Max(s_1, s_2, \cdots, s_n)) = O(N)$, where $m_1, m_2, m_3, \cdots, m_n$ are diagonal elements in matrix M(e), $d_1, d_2, d_3, \cdots, d_n$ are elements in offset-vector d(e), both M(e) and d(e) are taken as constant values independent of the DSP algorithm's input size N, and $s_1, s_2, s_3, \cdots, s_n$ are Cartesian coordinates of S in the iteration space ranged by the algorithm's input size N.

Theorem 3.3: The traditional MD retiming techniques cannot asymptotically reduce the minimum storage requirement in the hardware mapping of multirate MD DSP algorithms which are represented by MR-MDFGs.

Proof:

Consider an edge e: $U \rightarrow V$ in an MR-MDFG, and two retiming vectors r(U) and r(V). From Eq. (3.3) we have e's retimed offset weight $d_r(e) = d(e) + M(e)r(U) - r(v)$. A retiming vector is applied to the same computation node in all cells (MDFGs). In other words, the same computation node in all cells will be moved by the same distance in the iteration space according to the retiming vector, thus the length of the retiming vector (or the moving distance) should be a small value independent of N (the algorithm's input size). Otherwise the epilogue[40] or prologue[40] will be so big that the retiming operations are meaningless. Because only r(U) and r(V) affect the changing of the length of the cell dependence vector between these two nodes in retiming operations, after retiming operations, the length of the cell dependence vector is still $O(N)$, the same as in *Lemma 3.3*. Since to reduce the lengths of cell dependence vectors is the only benefit that can be exploited by retiming operations to reduce the minimum storage requirement, we reach the conclusion described in this theorem.

Based on the previous analyses, the normal MD retiming techniques [49][50][51] [52][53] published in present research papers are not applicable to the MR-MDFGs. Addressing this problem, we introduce the technique of *MD intercalation* which can be done in three steps in the following descriptions.

Procedure 3.1: MD intercalation

Step 1. Partitioning the rate-balanced MR-MDFG into single-rate subgraphs: Cut off all multirate-weighted edges from the MR-MDFG (i.e., those edges whose multirate-weights are not unit matrices) to obtain subgraphs each of which is a single-rate MDFG (as in *Theorem 3.2*).

Step 2. Creating the normalization matrix: Suppose that the original MR-MDFG has been partitioned into K subgraphs: $G_1, G_2, \cdots, G_K$ in Step 1, and the rates of these subgraphs are represented by matrices $R_1, R_2, \cdots, R_K$. Assume $r_{i,1}, r_{i,2}, \cdots, r_{i,n}$ are the diagonal elements of the matrix R_i $(1 \leq i \leq K)$. Create a special diagonal matrix R_0 called *normalization matrix* whose diagonal elements $r_{0,1}, r_{0,2}, \cdots, r_{0,n}$ are evaluated in the following way: $r_{0,j}$ is equal to the least common multiple of $r_{1,j}, r_{2,j}, \cdots, r_{K,j}$ $(1 \leq j \leq n)$.

Step 3. MD expansion and intercalation in the iteration space: Consider any a cell indexed by an n-component vector t in iteration space, and a subgraph G_i $(1 \leq i \leq K)$ which is partitioned from the original MR-MDFG in the first step and located in this cell. The operation of MD intercalation on this subgraph in this cell is to move G_i (including all the nodes and edges within G_i) to a new position in iteration space indexed by vector $(R_0/R_i) \times t$. Apply such operation of MD intercalation to all partitioned subgraphs (in the first step) in all cells in the iteration space.

We call *Procedure 3.1* as "MD intercalation" because the MR-MDFG is parti-

tioned into single-rate subgraphs and these subgraphs are "intercalated" in iteration space according to their rates.

Theorem 3.4: After the MD intercalation for rate-balanced MR-MDFG's, the lengths of the dependence vectors in the iteration space are independent of the DSP algorithm's input size.

Proof:

Assume an edge e: $U \rightarrow V$ with the offset-weight d(e) and the multirate-weight M(e).

1) If its multirate-weight M(e) is a unit matrix, e is in a subgraph (assumed to be G_i) partitioned in Step 1. The dependence vector in the iteration space corresponding to e is from the cell including U (indexed by S) to the cell including V (indexed by T). The length of the dependence vector before MD intercalation is $|S - T| = |d(e)|$. Suppose that G_i's rate is R_i. The instance of G_i in the cell at S is moved to the position S'=$(R_0/R_i) \times S$. The instance of G_i at T is moved to the position T'=$(R_0/R_i) \times T$ according to Step 3 in the procedure of MD intercalation. The length of the dependence vector is $|S' - T'| = |(R_0/R_i) \times S - (R_0/R_i) \times T| = |(R_0/R_i) \times d(e)|$, which is independent of the algorithm's input size (the range of the iteration space).

2) If e's multirate-weight M(e) is not a unit matrix, e must be between two subgraphs (assumed to be G_i and G_j) partitioned in Step 1. Suppose U is in G_i and V is in G_j, G_i and G_j's rates are R_i and R_j respectively. The dependence vector in the iteration space corresponding to e is from the cell including U (indexed by S) to the cell including V (indexed by T). The length of the dependence vector before MD intercalation is $|S - T| = |S - M(e)S + d(e)|$. After MD intercalation, the instance of G_i (including U) in cell S is moved to the position S'=$(R_0/R_i) \times S$, and the instance of G_j (including V) in cell T is moved to the position T'=$(R_0/R_j) \times T$. Thus the length of the dependence vector after MD intercalation is $|S' - T'| = |(R_0/R_j) \times d(e)|$,

which is independent of the algorithm's input size, or the range of the iteration space.

By *Theorem 3.4* we clarify one of the benefits of the MD intercalation which is to make the dependence vectors in the iteration space constant values that are independent of either the DSP algorithm's input size or the positions of the dependence vectors. The lengths of dependence vectors correspond to the distances of data dependencies in the iteration space, and thus correspond to the storage requirement in hardware mapping. Other benefits of MD intercalation, such as the availability of the systematic designs for multirate MD DSP algorithms where *MD retiming*, *full parallelism* and *arbitrary input format* are allowed, will be addressed in the following section. In a schematic view of the iteration space, the effect of the MD intercalation is just to relocate the multirate subgraphs partitioned from the MR-MDFG in each cell, so that the dependence vectors are *uniformly* distributed in the iteration space. In this way, we can apply the retiming techniques to the iteration space with quite simple equations and unified formulations for both single-rate edges and multirate edges, as proposed in the following section.

D. Retiming Formulations for MR-MDFG's

The first subsection of this section derives unified retiming equations for MR-MDFG's based on an intercalated iteration space. The second subsection summarizes mathematical functions describing arbitrary linear processing order of an MD data set, which can be taken as an extension of the analysis of 2-D case in [32]. Based on these two subsections, the final retiming formulations are proposed in Subsection D.3 in this chapter.

1. Retiming Equations for Intercalated Iteration Space

In the proof of *Lemma 3.3* and *Theorem 3.4* we have calculated the lengths of the dependence vectors in the iteration space of an MR-MDFG before and after the MD intercalation. In this subsection we continue the calculations in iteration space and give unified retiming equations on the MR-MDFG. These unified retiming equations on the MR-MDFG directly lead to the derivation of the unified methodology for a complete formulation of retiming on both single-rate and multirate DSP dataflow graphs in Subsection D.3 in this chapter.

In this subsection, we also demonstrate the significance of MD intercalation in unifying the retiming for multirate edges and single-rate edges in the MR-MDFG. As introduced in Section A of this chapter, in literature the MD retiming operation has been defined as the redistribution or movement of computation nodes in the n-D iteration space [32]. Since all single-rate subgraphs of the MR-MDFG have been moved according to rates of these subgraphs as specified in the steps of *Procedure 3.1* after the MD intercalation, the retiming operation needs to be reinterpreted as the movement of computation nodes on the *grids* of the n-D iteration space, where the *grids* are the sets of uniformly-distributed points (i.e. iteration cells) in the intercalated iteration space for various subgraphs of the MR-MDFG.

Assume an edge e: $U \rightarrow V$ with offset-weight d(e) and multirate-weight M(e) in a rate-balanced MR-MDFG. Suppose in the first step of the MD intercalation the MR-MDFG is partitioned, and nodes U and V are in the subgraphs G_i and G_j respectively, with R_i as the rate of G_i, and R_j as the rate of G_j. R_0 is the normalization matrix. Assume that the retiming operations on nodes U and V are represented by n-component retiming vectors r(U) and r(V). The retiming equations give the dependence relations between instances of U and V in iteration space after retiming

operation. Now we decide the dependence vector in iteration space corresponding to edge e before and after the MD intercalation.

1) If M(e) is a unit matrix, or e is a single-rate edge, G_i is the same as G_j and $R_i = R_j$. A) *Before MD intercalation*: retiming vector r(U) (or r(V)) is defined as the difference vector between the original position and the retimed position of node U (or V) in the iteration space in terms of Cartesian coordinates. Similar to [32][39][40][41], the dependence vector before retiming is d(e), and the dependence vector after retiming is $d_r(e)$=d(e)+r(U)-r(V). B) *After MD intercalation*: nodes U and V in any cell in the iteration space are moved according to R_i, thus U and V are only possibly located in the grid whose positions are indexed by $(R_0/R_i) \times X$ ranged in the iteration space, where X is any n-component vector whose elements are integers. We redefine the retiming vector r(U) (or r(V)) after MD intercalation as *the moving distance within the grid* of node U (or V). As in the proof of *Theorem 3.4*, the dependence vector in the iteration space after MD intercalation yet before retiming is $(R_0/R_i) \times d(e)$. Furthermore, the dependence vector in the iteration space after MD intercalation and after retiming is $(R_0/R_i) \times d_r(e) = (R_0/R_j) \times (d(e) + r(U) - r(V))$.

2) If M(e) is not a unit matrix, in other words, if e is not a single-rate edge, G_i is different from G_j and R_i is not equal to R_j. A) *Before MD intercalation*: retiming vector r(U) (or r(V)) is defined as the difference vector between the original position and the retimed position of node U (or V) in the iteration space in terms of Cartesian coordinates. Suppose in a cell indexed by S in the iteration space is found a copy of the MR-MDFG in which a node U is located. The dependence vector in the iteration space starting from this node U is $d(e) + S - M(e)S$. The dependence vector after retiming (yet before MD intercalation) is $(d(e) + S - M(e)S) + M(e)r(U) - r(V)$, based on Eq. (3.2), Eq. (3.3) and the restriction that the dependence relationships should not be changed after retiming. B) *After MD intercalation*: as in the above

paragraph, U (or V) is only possibly located in the grid whose positions are indexed by $(R_0/R_i) \times X$ ranged in the iteration space, where X is any n-component vector whose elements are integers. We also redefine the retiming vector r(U) (or r(V)) after MD intercalation as *the moving distance within the grid* for node U (or V). The dependence vector in the iteration space after MD intercalation yet before retiming is $(R_0/R_j) \times d(e)$ according to *Theorem 3.4*. Considering the dependence vector in the iteration space after retiming yet without MD intercalation, and the different moving distances of r(U) and r(V) because of MD intercalation, we have that the dependence vector in the iteration space after MD intercalation and after retiming is $(R_0/R_j) \times (d(e) + r(U) - r(V))$, with the same equation as for single-rate edge.

Based on the above analysis, by MD intercalation, we have *unified* MD retiming equations for single-rate edges and for multirate edges in the MR-MDFG. Moreover, the lengths of dependence vectors after MD intercalation and retiming are independent of the data positions in iteration space. The retiming equations for MR-MDFG after MD intercalation have similar format as traditional retiming equations for single-rate MDFG. These unified retiming equations become the basis for the complete retiming formulation in Subsection D.3 in this chapter.

2. Serial Processing Order on MD Data Set

There are two basic types of parallelism available in MD multirate DSP algorithms. One type of parallelism is inter-iteration parallelism where a parallel execution order of data set is needed to speed up the executions of the MD DSP algorithms. Another type of parallelism is inter-operation (or intra-iteration) parallelism, which involves retiming the MDFG so that operations of the computation nodes can be executed in parallel, resulting in a shorter clock period. In this chapter we consider retiming and assume an MD data set being processed with a serial linear processing order (e.g.

row-wise or column-wise).

The arbitrary linear processing order of an MD data set introduced here can be taken as a straightforward extension of the analysis for the 2-D case in [32].

Suppose the system is n-dimensional. A group of n n-dimensional unitary orthogonal vectors $\{S_1, S_2, \cdots, S_n\}$ is used to describe the processing order. Let (A, B) be the inner product of any two vectors A and B. Assume any two data samples indexed by n-component vectors X and Y respectively in the MD data set. The processing order of X and Y is decided as the following. 1) If $(X, S_1) \neq (Y, S_1)$, the sample corresponding to the smaller one of the two inner products will be processed earlier. 2) Else if $(X, S_2) \neq (Y, S_2)$, the sample corresponding to the smaller one of the two inner products will be processed earlier. 3)$\cdots$. n) Else if $(X, S_n) \neq (Y, S_n)$, the sample corresponding to the smaller one of the two inner products will be processed earlier.

Let H_1 be the maximum number of samples from the MD data set on a hyperplane indicated by equation $(X, S_1) = C_1$, where X is the n-component vector to index the sample, and C_1 is any constant value. Let H_2 be the maximum number of samples on a hyperplane indicated by equations $(X, S_1) = C_1$ as well as $(X, S_2) = C_2$, where C_1 and C_2 are any constant values. Let H_3 be $\cdots$. $\cdots$. Let H_{n-1} be the maximum number of samples on a hyperplane indicated by equations $(X, S_1) = C_1$, $(X, S_2) = C_2, \cdots$, and $(X, S_{n-1}) = C_{n-1}$, where $C_1, C_2, \cdots$, and C_{n-1} are any constant values.

Definition 3.7: The Eigen-function $F(X)$ of a linear processing order on an MD data set represented by $\{S_1, S_2, \cdots, S_n\}$ is $F(X) = \sum_{i=1}^{n-1} H_i \times (X, S_i)$, where X is any n-component vector.

3. The Complete Retiming Formulation

Many researches on MD retiming for single-rate MDFG's have been conducted. Most details within this subsection come from similar derivations in earlier researches about MD retiming for single-rate MDFG's ([32][39][40][41]). Our contribution is to unify the MD retiming formulations for single-rate MDFG's and for MR-MDFG's according to the contents in previous parts of this chapter.

Let R_v be the storage cost for a computation node v in the MR-MDFG in hardware mapping. R_v is the minimum storage necessary to store the data outgoing from node v and later consumed by other computation nodes in the MR-MDFG. Define W(u,v) as min $\{F(d(p_x))\}$ where the minimum is taken over $\{p_x | p_x$ is any path from node u to node v$\}$. Let t(u) be the time for the computation node u to perform a calculation. Define t(p) as $\sum_{i=0}^{i=K} t(v_i)$ where path p consists of nodes $\{v_i | i \in [1, K]\}$. Define t(u, v) as Max$\{$t(p)$\}$ where the maximum is taken over $\{$p | p is any path from u to v such that F(d(p))=W(u,v)$\}$.

Theorem 3.5: The complete formulation of retiming for MD intercalated iteration space should be under the following constraints: 1) cost constraint: $R_u = Max(F(d(e)+r(u)-r(v))$ for any edge e: $u \rightarrow v$ outgoing from u in the MR-MDFG; 2) causality constraint: $F(r(v) - r(u)) \leq F(d(e))$ for any edge $u \rightarrow v$ in the MR-MDFG; 3) clock period constraint: $F(r(v) - r(u)) \leq W(u, v) - 1$ for all nodes u and v in the MR-MDFG such that $t(u, v) > c$, where c is the requirement of the system's minimum clock period.

Proof:

In the case of single-rate MR-MDFG, the MD intercalation will not lead to the partition of the MR-MDFG because the multirate-weight of any edge in the MR-MDFG is a unit matrix. Then the proof of this theorem is simply an extension from

the 2-D case [32] to n-D case. The proof follows the similar proof for 2-D case. Note that we use unitary vectors in n-D space to designate the processing order.

In the case that not all edges in the MR-MDFG have unit matrices as the multirate-weights, the retiming vector r(u) has been redefined as the moving distance vector within the *grid* by intercalation (instead of within iteration space), but the retiming equations for an MR-MDFG are the same as the normal retiming equations for a single-rate MR-MDFG according to the analyses in Subsection D.1 in this chapter. Based on the unified MD retiming equations for the single-rate edges and the multirate edges in the MR-MDFG, and considering that the three constraints for the case of single-rate MR-MDFG had been derived only from the same retiming equations in [32], we can conclude the same three constraints of the retiming formulations for the MR-MDFG, followed by the reasoning similar to the proof in [32].

When the requirement of the system's minimum clock period is set to 1, the clock period constraint can be represented by the constraint that any edge in the MR-MDFG should have non-zero valued offset-weight after retiming.

The complete formulation of the MD retiming for a multirate MD DSP algorithm represented by an MR-MDFG, which are preprocessed by the MD intercalation, is described as: *to find retiming vectors for the computation nodes in the MR-MDFG so as to minimize COST=*$\sum_{v \in V} R_v$ *under the cost constraint, causality constraint and clock period constraint, where V is the set of all nodes in the MR-MDFG.*

This formulation of the MD retiming for multirate MD DSP algorithms is the same as that of the traditional retiming for single-rate algorithms. There are many practical procedures that have been derived from this traditional formulation, some of which can be found in [32][34][38][39][40][41]. With the space limit, we do not present a detailed discussion of systematic procedures to locate the retiming vec-

tors for the computation nodes under constraints of minimum cost. Since we have proposed a unified formulation of retiming technology, the systematic procedures for determining retiming vectors in multirate systems according to the formulation are to be established in the same way as those reports in previous literature for single-rate systems.

CHAPTER IV

EXPLOITING INTER-ITERATION PARALLELISM IN MULTIDIMENSIONAL MULTIRATE ALGORITHMS

A. Basic Properties

In this chapter, we continue to use those symbols and terms which have been defined and used in Chapter III. Before going to further analysis, we present some basic concepts and properties to be used later.

Property 4.1: Suppose $\mathbf{M}(e_1), \mathbf{M}(e_2), \cdots, \mathbf{M}(e_s)$ are the multirate-weights for all edges on a cycle in a rate-balanced MR-MDFG. To avoid the rate of data flow in the system being infinite or zero, and so as for the system to be meaningful, we have that $\prod_{i=1}^{s} \mathbf{M}(e_i) = 1$. This is consistent with the claim that for any edge $e : u \rightarrow v$, we have $\mathbf{R}(e) = \mathbf{Z}(p)$ in a rate-balanced MR-MDFG, where p is any path from one of the sources of the MDFG to node v and going through edge e.

Property 4.2: Suppose that $(U_1, U2, \cdots, U_s)$, and $e_1, e_2, \cdots, e_s$ are successively s nodes and edges on *any* a path in a rate-balanced MR-MDFG, and y_i (or x_i) is the input (or output) of node U_i for $1 \leq i \leq s$ on the path. Thus we have that $y_2(\mathbf{M}(e_1)\mathbf{t}_0 + \mathbf{d}(e_1)) = x_1(\mathbf{t}_0)$. $y_3(\mathbf{M}(e_2)\mathbf{M}(e_1)\mathbf{t}_0 + \mathbf{M}(e_2)\mathbf{d}(e_1) + \mathbf{d}(e_2))$ is dependent on $x_1(\mathbf{t}_0)$, $y_4 \cdots$, where $\mathbf{t}_0$ is the index of x_1 and integral such that the indices of y_i are also integral. If appropriate integral values of components of $\mathbf{t}_0$ can always be chosen periodically solely determined by $\mathbf{M}(e_i), 1 \leq i \leq s$, which is the original meaning of a "multirate system", we have that $\mathbf{d}(e_l) \times \prod_{j=l+1}^{k} \mathbf{M}(e_j)$ (for $1 \leq l \leq s-1; l+1 \leq k \leq s$) should be integral.

Property 4.3: Suppose that $(U_1, U2, \cdots, U_s)$, and $e_1, e_2, \cdots, e_s$ are successively s nodes and edges on a path L in an MR-MDFG. $\mathbf{d}(L)$ should be integral.

Property 4.4: Suppose that a and b are co-prime integers. Suppose that c and d are co-prime integers. Then $GCD(a/b, c/d) = GCD(a, c)/LCM(b, d)$.

Property 4.5: For integers a, b and c, $(a\%b + c)\%b = (a + c)\%b$. When a is a fractional number, and b, c, ab, ac are integers, the second equations also holds. The property also holds for integral vectors a, b, and c.

Property 4.6: If and only if an MR-MDFG is rate-balanced, for any edge in the MR-MDFG, e.g., $e : u \rightarrow v$, we have $\mathbf{R}(e) = \mathbf{Z}(p)$, where p is any path from one of the sources in the MDFG to node v and going through edge e.

Definition 4.1: The *rate of a node* U in a rate-balanced MR-MDFG, $\mathbf{R}(U)$, is equal to the rates of the input edges of node U if U is not a source of MR-MDFG; otherwise $\mathbf{R}(U) = \mathbf{1}$. Note that this is well-defined because the rates of input edges (if any) for any node are equal in a rate-balanced MR-MDFG.

Definition 4.2: When we partition a rate-balanced MR-MDFG by cutting off all multirate edges to get subgraphs containing nodes and single-rate edges only, each of such subgraphs is called a *unit subgraph*.

Definition 4.3: The *rate of a unit subgraph* X in a rate-balanced MR-MDFG, $\mathbf{R}(X)$, is equal to the rate of any node in X. Note that this is well-defined because all nodes in X are connected by single-rate edges so that the rates of their input edges are equal and then the rates of these nodes are equal.

Property 4.7: For any edge $e : U \rightarrow V$ in a rate-balanced MR-MDFG, we always have $\mathbf{R}(V) = \mathbf{R}(U) \times \mathbf{M}(e)$.

Definition 4.4: The *dependence vectors between the cells in the iteration space* are the vectors corresponding to the dependence relations of operations among different cells (or iterations) in iteration space.

B. Multidimensional Multirate Unfolding

This section studies systematic multidimensional unfolding on the MR-MDFG and properties of unfolded MR-MDFG. References [54][55] have given presentations on systematic unfolding transforms on 1-D algorithms, where successively executed iterations are represented by an unfolded 1-D DFG. However, the extension of 1-D unfolding to n-D unfolding is not trivial especially when considering the newly generated delay-weights in the unfolded MDFG and when the original MDFG is cyclic.

1. Construction of Unfolded MDFG

The 1-D unfolding transform exploits the inter-iteration precedence constraints in addition to the intra-iteration precedence constraints, and can lead to the overlapped schedules (i.e., the tasks of an iteration are scheduled to be executed before all the tasks of previous iterations have been executed). Now we present the general procedure of unfolding transform on an MDFG which corresponds to an n-D algorithm. Suppose that $\mathbf{f}$ is the unfolding factor (an n-component vector, with positive integers $f_1, f_2, \cdots, f_n$ as components).

A simple illustration of an unfolding transform of a 2-D MDFG, G, which contains only two nodes A and B and an edge e with delay-weight $\mathbf{d}(e) = (1, 1)$ is given in Fig. 14, where $\mathbf{f} = (2, 3)$.

Procedure 4.1: Multidimensional unfolding on MDFG

Input: G, a general MDFG; *Output: G', the unfolded G by unfolding factor* $\mathbf{f}$.

Step 1: Suppose a node in G is U. Draw $|\mathbf{f}|$ nodes in G' which are corresponding to U and are taken as instances of U, and label any one of these $|\mathbf{f}|$ nodes by $U_{\mathbf{t}}$, where $\mathbf{t}$ is one of $|\mathbf{f}|$ integral n-component vectors that satisfy $\mathbf{0} \leq \mathbf{t} \leq (\mathbf{f} - \mathbf{1})$. Apply such operations to all nodes in G.

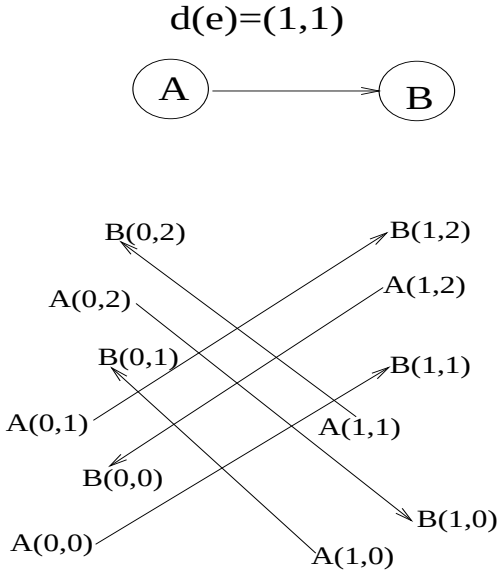


Fig. 14. A simple example of multidimensional unfolding transform on a 2-D MDFG

Step 2: Suppose an edge in G is $e : U \rightarrow V$, whose delay-weight is $\mathbf{d}(e)$. Corresponding to e, draw $|\mathbf{f}|$ edges in G' such that: there is an edge from $U_{\mathbf{t}}$ to $V_{\mathbf{s}}$ in G', where $\mathbf{t}$ is any one of $|\mathbf{f}|$ integral n-component vectors that satisfy $\mathbf{0} \leq \mathbf{t} \leq (\mathbf{f} - \mathbf{1})$; $\mathbf{s}$ is equal to $(\mathbf{t} + \mathbf{d}(e))\%\mathbf{f}$; and the delay-weight of the new edge $e_{\mathbf{t}} : U_{\mathbf{t}} \rightarrow V_{\mathbf{s}}$ in G', $\mathbf{d}(e_{\mathbf{t}})$, is equal to $\lfloor(\mathbf{t} + \mathbf{d}(e))/\mathbf{f}\rfloor$. Apply such operations to all edges in G.

In the example of Fig. 14, we copy 2×3 instances of each node from G into G', and label the instances by $\mathbf{t}$ which is any one of $|\mathbf{f}|$ integral 2-component vectors that satisfy $\mathbf{0} \leq \mathbf{t} \leq (\mathbf{f} - \mathbf{1})$, where $\mathbf{f} = (2, 3)$. The example is a specific application of *Procedure 4.1*. Another more complicate case of 2-D unfolding transform on an 2-D MDFG is shown in Fig. 15.

Theorem 4.1: The procedure of multidimensional unfolding on MDFG described in *Procedure 4.1* preserves all dependence relations existing in the n-D iteration space (Φ).

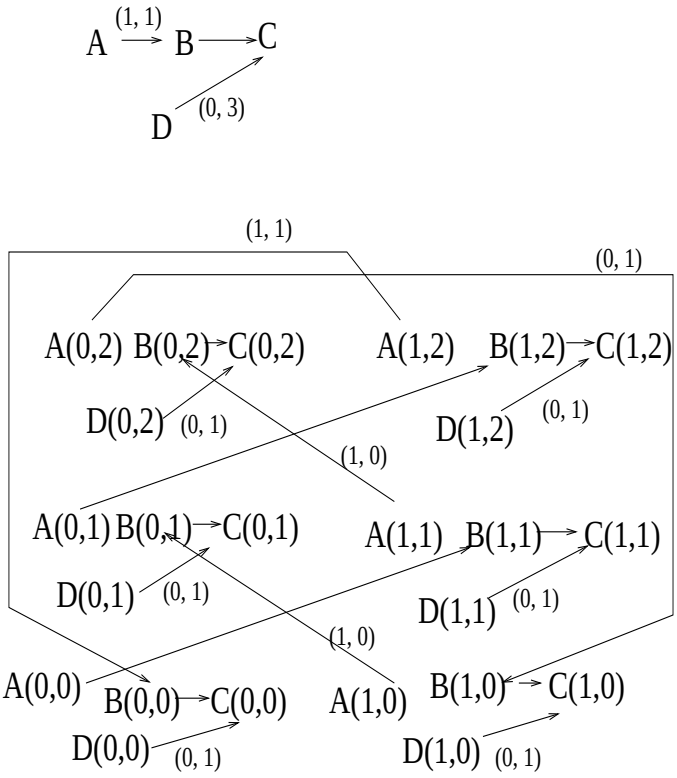


Fig. 15. Another example of multidimensional unfolding on a 2-D MDFG

Proof:

Suppose that the original MDFG is G. The task of this proof is to demonstrate that all dependence vectors between cells in Φ (in which each cell is an instance of G, excluding non-zero delay-weighted edges) are preserved, and all dependence relations within a cell in Φ (specified by edges in G whose delay-weights are $\mathbf{0}$) are also maintained after unfolding transform. After *Procedure 4.1*, we obtain G' with unfolding factor $\mathbf{f}$. A new n-D iteration space Φ' is constructed based on G'. Now we show that all dependencies (including those among cells and within each cell) existing in Φ and Φ' have one-to-one corresponding relationship.

Suppose that ξ represents any one of dependencies in Φ: $U_{\mathbf{x}} \longmapsto V_{\mathbf{y}}$, where $U_{\mathbf{x}}$ is an instance of node U (a node of G) located in the cell indexed by $\mathbf{x}$ in Φ, and $V_{\mathbf{y}}$

is an instance of node V (a node of G) located in the cell indexed by $\mathbf{y}$ in Φ. ξ is specified by edge $e : U \rightarrow V$ in G. If $\mathbf{d}(e) = \mathbf{0}$, we have $\mathbf{x} = \mathbf{y}$ and ξ represents a dependence within cell $\mathbf{x}$ in Φ, otherwise ξ represents a dependence between different cells $\mathbf{x}$ and $\mathbf{y}$ in Φ, where $\mathbf{y} = \mathbf{x} + \mathbf{d}(e)$.

Now we are to prove that we can always find a unique dependence ξ' from Φ' that corresponds to ξ. First, we present a one-to-one corresponding relationship between nodes in cells of Φ and those in cells of Φ'. Corresponding to any node W (a node of G) in the cell indexed by $\mathbf{z}$ in Φ, we find a unique node $W_{\mathbf{t}}$ (a node of G') in the cell indexed by $\mathbf{r}$ in Φ', where $\mathbf{t} = \mathbf{z}\%\mathbf{f}$ and $\mathbf{r} = \lfloor \mathbf{z}/\mathbf{f} \rfloor$. Vice versa, we can always find a unique node in Φ given any node in Φ'. Thus, we can find a unique node in Φ' corresponding to $U_{\mathbf{x}}$ in Φ: $U_{\mathbf{t}}$ (a node of G') in the cell indexed by $\mathbf{p}$ in Φ', where $\mathbf{t} = \mathbf{x}\%\mathbf{f}$ and $\mathbf{p} = \lfloor \mathbf{x}/\mathbf{f} \rfloor$. We label this node as $U_{\mathbf{t},\mathbf{p}}$. In Φ', there is a dependence: $U_{\mathbf{t},\mathbf{p}} \longmapsto V_{\mathbf{s},\mathbf{q}}$ (where $V_{\mathbf{s},\mathbf{q}}$ is an instance of node $V_{\mathbf{s}}$ from G' located in the cell indexed by $\mathbf{q}$), which is specified by an edge of G', $e_{\mathbf{t}} : U_{\mathbf{t}} \rightarrow V_{\mathbf{s}}$, with $\mathbf{s}$ equal to $(\mathbf{t}+\mathbf{d}(e))\%\mathbf{f}$, and $\mathbf{d}(e_{\mathbf{t}})$ equal to $\lfloor (\mathbf{t}+\mathbf{d}(e))/\mathbf{f} \rfloor$ (per Step 2 of *Procedure 4.1*). Moreover, $\mathbf{q} = \mathbf{p} + \mathbf{d}(e_{\mathbf{t}})$. In accordance with the one-to-one corresponding relationship between nodes in cells of Φ and those nodes in cells of Φ' which we specify at the beginning of this paragraph, $V_{\mathbf{s},\mathbf{q}}$ in Φ' corresponds to $V_{\mathbf{y}'}$ in Φ, where $\mathbf{y}' = \mathbf{q}\mathbf{f} + \mathbf{s}$. Replacing $\mathbf{q}$ and $\mathbf{s}$ by their values as derived above, we have $\mathbf{y}$=$\mathbf{y}$. In other words, $V_{\mathbf{s},\mathbf{q}}$ in Φ' uniquely corresponds to $V_{\mathbf{y}}$ in Φ, or, the dependence: $U_{\mathbf{t},\mathbf{p}} \longmapsto V_{\mathbf{s},\mathbf{q}}$ is ξ' that we are looking for in Φ' corresponding to ξ in Φ.

In a similar way, we can demonstrate that a dependence ξ can always be found from Φ given any ξ' in Φ'. Thus, we conclude that all dependence relations existing in Φ are preserved in Φ' after *Procedure 4.1*.

2. Properties of Multidimensional Unfolding

As introduced in the next section, the cycles in the original MDFG play a key role in affecting the inter-iteration parallelism within the multidimensional DSP algorithms, so we pay special attention to the cycles in the unfolded MDFG when studying the transform of multidimensional unfolding. Fig. 16 gives an example of 2-D unfolding where some distinct cycles are generated in the unfolded MDFG.

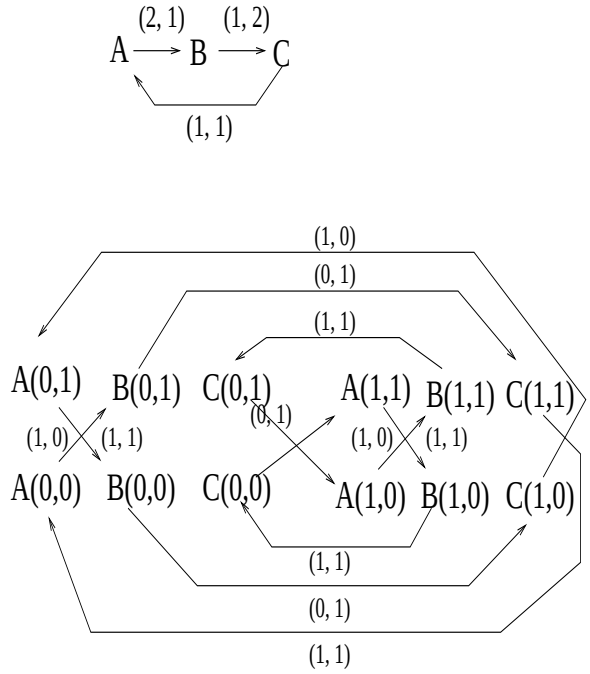


Fig. 16. A multidimensional unfolding on a cyclic 2-D MDFG

Property 4.8: Corresponding to each cycle (if any) in the original MDFG, there are $|\mathbf{f}|/Q$ distinct cycles in the unfolded MDFG after the unfolding transform as described in *Procedure 4.1*. $\mathbf{f}$ is the unfolding factor. Q is evaluated in the following way. Suppose $\mathbf{d}(L)$ is the sum of the delay-weights along all edges in cycle L in the original MDFG. $d_1, d_2, \cdots, d_n$ are components of $\mathbf{d}(L)$. Suppose $f_1x_1 = d_1y_1, f_2x_2 = d_2y_2, \cdots, f_nx_n = d_ny_n$, where x_i and y_i are co-prime integers for $1 \leq i \leq n$. Then

$Q = LCM(y_1, y_2, \cdots, y_n)$.

Property 4.9: Corresponding to each edge $e : U \rightarrow V$ with the delay-weight $\mathbf{d}(e)$ in the original MDFG, the total delay-weights of the instances of edge e in the unfolded MDFG is $|\mathbf{f}| \times \mathbf{d}(e)/\mathbf{f}$.

Property 4.10: Corresponding to each cycle L in the original MDFG, the sum of delay-weights of all edges in each distinct cycle Δ (corresponding to L) in the unfolded MDFG is equal to $Q \times \mathbf{d}(L)/\mathbf{f}$, where $\mathbf{d}(L)$ is the sum of delay-weights of all edges in L, and Q and $\mathbf{f}$ were previously defined.

3. Translating MR-MDFG into Single-rate MDFG

The procedure of multidimensional intercalation, *Procedure 3.1*, relocates the cells of unit subgraphs of an MR-MDFG in iteration space. K is the number of unit subgraphs in the MR-MDFG. Consider a unit subgraph in an MR-MDFG, G_i $(1 \leq i \leq K)$, whose rate is assumed to be $\mathbf{R}_i$. From every cell of iteration space (e.g., a cell indexed by $\mathbf{t}$), G_i is moved to another cell in iteration space (e.g., a cell indexed by $((\mathbf{R}_C)/\mathbf{R}_i) \times \mathbf{t}$) in *Procedure 3.1*. Thus after *Procedure 3.1*, there is only one cell containing G_i every other $\mathbf{R}_C/\mathbf{R}_i$ lattice points in iteration space. Let $\mathbf{R}_T$ be LCM$(1/\mathbf{R}_1, 1/\mathbf{R}_2, \cdots, 1/\mathbf{R}_K)$. Furthermore, if we combine the cells in every $|\mathbf{R}_C\mathbf{R}_T|$ adjacent lattice points in iteration space into a group, we can get a *combined data-flow graph* (including copies of G_i at these cells, $1 \leq i \leq K$) represented by such a group. So we get a *reduced iteration space* where each lattice point is such a group. In each of such group, there are $(R_C R_T)/(R_C/R_i)$ or $R_i R_T$ copies of G_i. Each group in the *reduced iteration space* contains *the same combined data-flow graph*, and the dependencies between groups are represented by constant vectors instead of variable vectors. Thus the reduced iteration space can actually be constructed based on a single-rate MDFG (i.e. all of its edges are single-rate edges with their multirate-

weights equal to 1) which is the combined data-flow graph corresponding to each group. The translation of an MR-MDFG into a single-rate MDFG is formulated in *Procedure 4.2*, which can be taken as a procedure combining *Procedure 3.1* and *Procedure 4.1*.

Procedure 4.2: Multidimensional unfolding on the multidimensionally intercalated MR-MDFG

Input: An MR-MDFG; *Output:* A single-rate MDFG.

Step 1: Cut off all multirate edges from the original MR-MDFG to partition it into K unit subgraphs: $G_1, G_2, \cdots, G_K$, and the rates of these unit subgraphs are $\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_K$. Assume $\mathbf{R}_C$ as evaluated in Step 2 of *Procedure 3.1*.

Step 2: Let $\mathbf{R}_T$ be LCM$(1/\mathbf{R}_1, 1/\mathbf{R}_2, \cdots, 1/\mathbf{R}_K)$. Take $\mathbf{R}_i\mathbf{R}_T$ as the multidimensional unfolding factor, apply *Procedure 4.1* onto G_i $(1 \leq i \leq K)$ respectively. Suppose that G_i is unfolded to H_i $(1 \leq i \leq K)$.

Step 3: Now we draw edges to connect $H_1, H_2, \cdots, H_K$. Consider each multirate edge connecting two unit subgraphs in the original MR-MDFG, e.g., $e : U \rightarrow V$, where $U \in G_i, V \in G_j, i \neq j, 1 \leq i, j \leq K$. Suppose the delay-weight and the multirate-weight of e are $\mathbf{d}(e)$ and $\mathbf{M}(e)$ respectively. Let $\mathbf{W} = MIN(\mathbf{R}_i, \mathbf{R}_j)$ and $\mathbf{f} = \mathbf{W} \times \mathbf{R}_T$. In H_i, find the following $|\mathbf{f}|$ instances of U: $U_{\mathbf{t}}$, where $\mathbf{t} = (\mathbf{R}_i/\mathbf{W}) \times \mathbf{A}$ and $\mathbf{A}$ is any one of $|\mathbf{f}|$ integral vectors that satisfy $\mathbf{0} \leq \mathbf{A} \leq \mathbf{f} - \mathbf{1}$. Starting from each $U_{\mathbf{t}}$, draw edge $e_{\mathbf{t}} : U_{\mathbf{t}} \rightarrow V_{[\mathbf{t} \times \mathbf{M}(e) + \mathbf{d}(e)]\%(\mathbf{R}_j \times \mathbf{R}_T)}$, where $V_{[\mathbf{t} \times \mathbf{M}(e) + \mathbf{d}(e)]\%(\mathbf{R}_j \times \mathbf{R}_T)}$ belongs to H_j, and the delay-weight of $e_{\mathbf{t}}$, $\mathbf{d}(e_{\mathbf{t}}) = \lfloor (\mathbf{t} \times \mathbf{M}(e) + \mathbf{d}(e))/(\mathbf{R}_j \times \mathbf{R}_T) \rfloor$.

In *Step 2*, G_i is unfolded to H_i with unfolding factor $\mathbf{R}_i\mathbf{R}_T$ and G_j is unfolded to H_j with unfolding factor $\mathbf{R}_j\mathbf{R}_T$ $(i \neq j, 1 \leq i, j \leq K)$. As assumed in the description of *Step 3*, G_i contains node U, G_j contains node V, and there is a multirate edge $e : U \rightarrow V$ in the original MR-MDFG. $\mathbf{R}_i \times \mathbf{M}(e) = \mathbf{R}_j$ per *Property 4.7* and

Definition 4.3. There are $|\mathbf{R}_i\mathbf{R}_T|$ instances of U in H_i and $|\mathbf{R}_j\mathbf{R}_T|$ instances of V in H_j because of unfolding transforms. Without losing generality, we may suppose $\mathbf{M}(e) < \mathbf{1}$, and thus there are more instances of U in H_i than those of V in H_j. We can draw edges (corresponding to e) only from some (NOT all) instances of U to (all) instances of V.

To explain the last sentence of *Step 3*, we employ one auxiliary node λ and two auxiliary edges $\delta : U \rightarrow \lambda$ and $\theta : \lambda \rightarrow V$ to replace e in the original MR-MDFG. $\mathbf{d}(\delta) = \mathbf{0}, \mathbf{M}(\delta) = \mathbf{M}(e), \mathbf{d}(\theta) = \mathbf{d}(e)$, and $\mathbf{M}(\theta) = \mathbf{1}$. After *Step 1*, δ is cut off, and λ, θ and V are located in G_j. After *Step 2* of *Procedure 4.2* (i.e. *Procedure 4.1*) is applied onto G_j with $\mathbf{R}_j\mathbf{R}_T$ as the unfolding factor, we have the unfolded MDFG, H_j, in which there are $|\mathbf{R}_j\mathbf{R}_T|$ instances of λ, θ and V. Following *Step 2* of *Procedure 4.1*, for each $\lambda_{\mathbf{s}}$ in H_j where $\mathbf{s}$ is any integral vector that satisfy $\mathbf{0} \leq \mathbf{s} \leq \mathbf{R}_j\mathbf{R}_T - \mathbf{1}$, we draw an edge $\theta\mathbf{s} : \lambda_{\mathbf{s}} \rightarrow V_{[\mathbf{s}+\mathbf{d}(e)]\%(\mathbf{R}_j\mathbf{R}_T)}$, with $\mathbf{d}(\theta_{\mathbf{s}}) = \lfloor(\mathbf{s}+\mathbf{d}(e))/(\mathbf{R}_j\mathbf{R}_T)\rfloor$. Now we connect H_i and H_j by drawing edges corresponding to δ starting from each $U_{\mathbf{t}}$ in H_i (the evaluation of $\mathbf{t}$ is the same as in *Step 3* of *Procedure 4.2*). Recalling that δ is an edge with delay-weight as zero and multirate-weight as $\mathbf{M}(e)$, and reviewing the input/output relation corresponding to a multirate-weighted edge, we draw $\delta_{\mathbf{t}} : U_{\mathbf{t}} \rightarrow \lambda_{\mathbf{t}\times\mathbf{M}(e)}$. Combining $\delta_{\mathbf{t}}$ and $\theta\mathbf{s}$, and replacing $\mathbf{s}$ by $\mathbf{t} \times \mathbf{M}(e)$, we have the expressions in *Step 3* of *Procedure 4.2*.

Including $H_1, H_2, \cdots, H_K$ and the edges connecting them generated in *Step 3* as above, we obtain a larger MDFG, which performs the same function as the original MR-MDFG as claimed in the following *Theorem 4.2*.

Theorem 4.2 The multidimensional unfolding on the multidimensionally intercalated MR-MDFG as formulated in *Procedure 4.2* preserves the dependence relationships inherent in the algorithm represented by the original MR-MDFG.

Proof:

In this proof we use the same symbols as before. The Multidimensional unfolding on the subgraphs G_i $(1 \leq i \leq K)$ does not change the dependence relationships represented by the edges within the subgraphs per *Theorem 4.1*. The multidimensional intercalation on the MR-MDFG does not change the dependence relations between the cells (in iteration space) corresponding to the subgraphs of the MR-MDFG as designated in Step 3 in *Procedure 3.1*. Now we show that the edges connecting the unfolded subgraphs $(H_i, 1 \leq i \leq K)$ drawn in Step 3 of *Procedure 4.2* are strictly corresponding to the dependence relations inherent in the algorithm represented by the original MR-MDFG. Consider any multirate edge in the MR-MDFG $e : U \rightarrow V$, where $U \in G_i, V \in G_j, i \neq j, 1 \leq i, j \leq K$. In the unfolded MDFG that combines all cells at $|\mathbf{R}_C\mathbf{R}_T|$ adjacent lattice points in iteration space (after multidimensional intercalation), there are $|\mathbf{R}_i\mathbf{R}_T|$ copies of subgraph G_i and $|\mathbf{R}_j\mathbf{R}_T|$ copies of subgraph G_j respectively. In the unfolded H_i (or H_j), there are $|\mathbf{R}_i\mathbf{R}_T|$ (or $|\mathbf{R}_j\mathbf{R}_T|$) instances of U (or V). It is obvious that not all instances of U are connected to all instances of V because $\mathbf{R}_j = \mathbf{R}_i \times \mathbf{M}(e) \neq \mathbf{R}_i$ per *Property 4.7*. For every instance of U, there are $|\mathbf{M}(e)|$ instances of V. Noticing this, and transferring $V_{\mathbf{t}}$ in terms of $V_{\mathbf{t}\times\mathbf{M}(e)}$, we could drag e (with V) into G_i when unfolding it to H_i. Note that there are $|\mathbf{W}\mathbf{R}_T|$ instances of e when unfolding G_i and G_j to H_i and H_j. According to Step 3 of *Procedure 4.1*, we could get the target instance of V when connecting U_t via e, and the delay-weight of e after unfolding. The only difference from *Procedure 4.1* is that we need to replace $\mathbf{t}$ by $\mathbf{t} \times \mathbf{M}(e)$ when V in H_j is involved and recall the unfolding factor for V as R_iR_T. Thus, *Step 3* of *Procedure 4.2* is just an application of *Step 3* of *Procedure 4.1* onto the result of multidimensional intercalation for those multirate edges in the MR-MDFG. So the dependence relations inherent in the algorithm represented by the original MR-MDFG are preserved in *Procedure 4.2*.

Property 4.11: Suppose L is a cycle in the original MR-MDFG containing s edges: $e_1, e_2, \cdots, e_s$ and s nodes: $U_1, U_2, \cdots, U_s$ *consecutively*. $\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_s$ are the rates of these nodes respectively. e_1 is the edge from U_1 to U_2. The *delay of* L, $\mathbf{d}(L)$, is defined as $\sum_{i=1}^{s}[(\prod_{j=i+1}^{s}\mathbf{M}(e_j))\mathbf{d}(e_i)]$ based on *Definition 3.5*. The unfolding factor for U_1, $\mathbf{f}$, is defined as $\mathbf{R}_1\mathbf{R}_T$, with the same $\mathbf{R}_T$ as in *Procedure 4.2*. $d_1, d_2, \cdots, d_n$ are n components of $\mathbf{d}(L)$. $f_1, f_2, \cdots, f_n$ are n components of $\mathbf{f}$. Suppose $f_1x_1 = d_1y_1, f_2x_2 = d_2y_2, \cdots, f_nx_n = d_ny_n$ where x_i and y_i are co-prime integers for $1 \leq i \leq n$, and $Q = LCM(y_1, y_2, \cdots, y_n)$. The following always holds: each distinct cycle corresponding to L in the translated single-rate MDFG (which is generated in *Procedure 4.2*) contains Q instances of nodes $U_1, U_2, \cdots, U_s$.

Property 4.12: When another node on *cycle* L is arbitrarily selected and re-labeled as U_1 in *Property 4.11* and the rest nodes on *cycle* L are re-labeled as $U_2, U_3, \cdots, U_s$ *successively* (as before, s is the number of nodes on cycle L), we accordingly re-label the edges on L as $e_1, e_2, \cdots, e_s$ *successively*, with e_1 being the edge from U_1 to U_2. The values of $\mathbf{d}_L$ and $\mathbf{f}$ which are defined in the description of *Property 4.11* change in accordance, but Q's value remains the same.

Property 4.13: Corresponding to cycle L in the original MR-MDFG, there are $|GCD(\mathbf{R}_1\mathbf{R}_T, \mathbf{R}_2\mathbf{R}_T, \cdots, \mathbf{R}_s\mathbf{R}_T)|/Q$ distinct cycles in the translated single-rate MDFG which is generated by *Procedure 4.2*. The symbols are used in the same way as before.

Property 4.14: Corresponding to each edge $e : U \rightarrow V$ in the original MR-MDFG, G, the summed delay-weights of all instances of edge e in the translated single-rate MDFG, G', is $\mathbf{D}$, where $\mathbf{D} = |\mathbf{F}|\mathbf{W}/\mathbf{F}$, $\mathbf{F} = MIN(\mathbf{R}_U\mathbf{R}_T, \mathbf{R}_V\mathbf{R}_T)$, $\mathbf{R}_U$ and $\mathbf{R}_V$ are the rates of U and V respectively, $\mathbf{R}_T$ is the same as before, $\mathbf{W} = MIN(\mathbf{d}(e), \lfloor \mathbf{d}(e)/\mathbf{M}(e) \rfloor)$, and $\mathbf{d}(e)$ and $\mathbf{M}(e)$ are e's delay-weight and multirate-weight respectively.

Property 4.15: Corresponding to cycle $L : U_1 \rightarrow U_2 \rightarrow \cdots \rightarrow U_s \rightarrow U_1$ in the

original MR-MDFG, a cycle $\Delta : U_{1,\mathbf{t_1}} \rightarrow \cdots \rightarrow U_{1,\mathbf{t_1}}$ is in the translated single-rate MDFG, where $\mathbf{t_1}$ is any one of $|\mathbf{R}_1\mathbf{R}_T|$ integral vectors that satisfy $\mathbf{0} \leq \mathbf{t_1} \leq \mathbf{R}_1\mathbf{R}_T - \mathbf{1}$. $\mathbf{R}_1, \mathbf{R}_2, \cdots, \mathbf{R}_s$ are the rates of nodes on L: $U_1, U_2, \cdots, U_s$ respectively. $\mathbf{d}(\Delta)$ is the summed delay-weights of all edges along Δ. The following always holds: $\mathbf{d}(\Delta) = Q\mathbf{d}(L)/\mathbf{f}$, where the *delay of* L, $\mathbf{d}(L)$, is defined based on *Definition 3.5*, and $\mathbf{f} = \mathbf{R}_1\mathbf{R}(T)$. Other symbols are the same as before.

Property 4.16: When another node on cycle L is arbitrarily selected and re-labeled as U_1 in *Property 4.15*, and its following nodes are re-labeled as $U_2, U_3, \cdots, U_s$ consecutively, the values of $\mathbf{d_L}$ and $\mathbf{f}$ change in accordance, but $\mathbf{d}(\Delta)$'s value remains the same.

Property 4.17: After *Procedure 4.2*, each cell in the iteration space for the data-flow graph represents $|\mathbf{R}_C\mathbf{R}_T|$ adjacent cells which are located in the iteration space before *Procedure 4.2*.

C. Exploring the Inter-Iteration Parallelism

The iteration period bound in any 1-D data-flow program with feedback loops is given in [54] by

$$T_0 = MAX[T_l/D_l]. \tag{4.1}$$

The maximum is taken over all feedback loops l in the DFG, T_l is the sum of the execution times associated with all the nodes in feedback loop l, and $\mathbf{D}_l$ is the sum of the delay-weights of all nodes in feedback loop l.

Periodic schedules are said to be *rate-optimal* if the iteration period is the same as the iteration bound. One can never achieve an iteration period less than this bound even when infinite processors are available. Parhi [55] has demonstrated that rate-optimal schedules can always be constructed for algorithms represented by a 1-

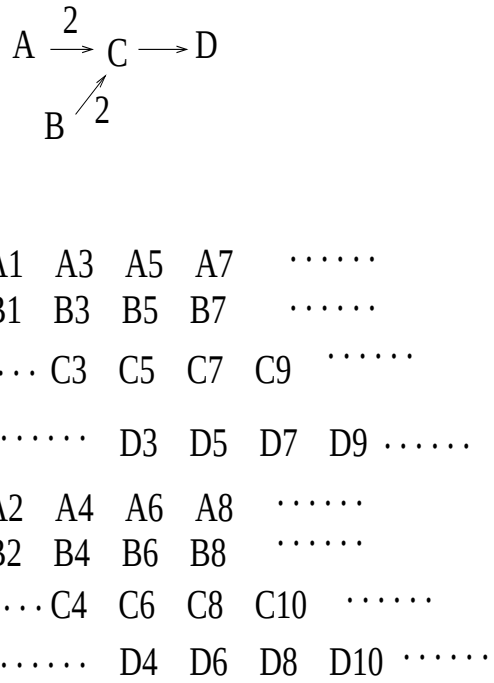
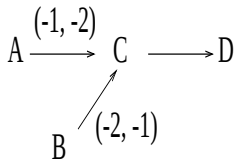


Fig. 17. An acyclic 1-D DFG executed with arbitrary concurrency

D DFG based on the construction of a so-called *perfect-rate DFG* shown as in Fig. 17. However, when the analysis is extended to multidimensional cases, the formula for the iteration bound is no longer applicable since the delay-weights in the data-flow graph are vectors instead of scalars. Moreover, the *perfect-rate DFG* can be constructed in[55] if and only if the delay-weight of an edge would not be changed by the operation of 1-D unfolding, and thus such *perfect-rate DFG* can NEVER be constructed in a multidimensional case based on *Property 4.9* and *Property 4.14* in this chapter. To the best of our knowledge, there is no research reported on exploring inter-iteration parallelism within the MDFG and especially within the MR-MDFG in literature to this date, one of the difficulties for which lies in the complex dependence relationships between cells in n-D iteration space.

Consider a multidimensional algorithm described by an *acyclic* MDFG in Fig. 18. Since all paths starting from the input nodes (sources) in the acyclic MDFG are



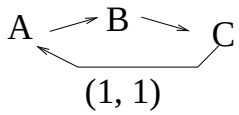
P11: A21 B12 C33 D33 A54 B45 C66 D66 ·····
P21: A31 B22 C43 D43 A64 B55 C76 D76 ·····
P31: A41 B32 C53 D53 A74 B65 C86 D86 ·····
P12: A22 B13 C34 D34 A55 B46 C67 D67 ·····
P22: A32 B23 C44 D44 A65 B56 C77 D77 ·····
P32: A42 B33 C54 D54 A75 B66 C87 D87 ·····
P13: A23 B14 C35 D35 A56 B47 C68 D68 ·····
P23: A33 B24 C45 D45 A66 B57 C78 D78 ·····
P33: A43 B34 C55 D55 A76 B67 C88 D88 ·····

Fig. 18. An acyclic 2-D MDFG executed with arbitrary concurrency

non-circular, we could assign a processor to each distinct path starting from sources of the MDFG in each cell in the iteration space, and have all these processors execute the tasks corresponding to the nodes along the paths at different cells concurrently. Though there are dependencies between processors because of the delay-weights in the MDFG, the task corresponding to the same node in the MDFG yet in different cells can be executed simultaneously without schedule conflict if the communication constraints among processors are neglected. Thus, considering that signal processing data-flow programs are non-terminating in nature, or that the order of the input size is more than that of the size of the data-flow graph, we can achieve arbitrarily shorter iteration period on average for algorithms represented by the acyclic MDFG given the availability of a large number of processors.

On the other hand, for an algorithm described by a *cyclic* MDFG in Fig. 19, though we could assign a few processors to each cell in iteration space, the same node

in the MDFG yet in some different cells CANNOT be executed simultaneously since there are dependence relationships regarding this node between cells. A conclusion can be drawn from Figs. 17, 18 and 19: in either a 1-D DSP algorithm or an n-D DSP algorithm, whether an arbitrarily shorter iteration period can be achieved is determined on whether there exists a node in the data-flow graph in a cell that depends on itself but in different cells in iteration space. More exactly, there is still a lower bound on the iteration period for algorithms represented by the cyclic MDFG, although the formula for the definition of iteration bound is not applicable to multidimensional cases because $\mathbf{D}_l$ is a vector instead of a scalar. In the remainder of this chapter, we do not consider the nodes of the MDFG which do not belong to any cycle, and those nodes can be scheduled using postprocessors with arbitrarily shorter iteration bound.



P11:	A11	B11	C11	$\cdots\cdots$
P12:	A12	B12	C12	$\cdots\cdots$
P13:	A13	B13	C13	$\cdots\cdots$
P21:	A21	B21	C21	$\cdots\cdots$
P22:	A22	B22	C21	$\cdots\cdots$
P23:	A23	B23	C23	$\cdots\cdots$
P31:	A31	B31	C31	$\cdots\cdots$
P32:	A32	B32	C32	$\cdots\cdots$
P33:	A33	B33	C33	$\cdots\cdots$

Fig. 19. A trial of parallel scheduling of a cyclic 2-D MDFG

To explore the iteration period bound in algorithms represented by the MDFG is to explore the inter-iteration parallelism within them. Only when as many as possible nodes located in all cells in iteration space are executed concurrently can the least

iteration period be achieved.

Prior to the discussion of the inter-iteration parallelism within cyclic multidimensional data-flow graphs, we introduce the concepts of *free schedule* and *time-optimal linear schedule*. In simple terms, an algorithm is represented as an ordered subset (index set) of a multidimensional iteration space. A *free schedule* assigns an operation indexed by x in iteration space to execute as soon as its operands are ready (i.e., all those operations finish execution on which the operation x depends). Free schedule is also usually refered to as soon as possible scheduling in synthesis community. Free schedules fully explore the parallelism of algorithms and the total execution achieved by a free schedule is exactly the lower bound of the execution time[56]. A *linear schedule* is a mapping from the multidimensional algorithm index set into a one-dimensional time space; this mapping is expressed as a linear transformation that involves the multiplication of a vector (*linear schedule vector*) by each lattice point of the index set. The purpose of time-optimal linear scheduling is to show how a linear schedule vector can be determined so that the algorithm can be executed in a minimal amount of time[56] [57]. The difference between the total execution times achieved by the free schedule and the optimal linear schedule is bounded by a constant, and often this constant is zero; Linear schedules are easier to use to measure the parallelism since the total execution time by a linear schedule is a closed function of linear schedule vector[57]. The problem of *time-optimal linear schedule* for uniform dependence algorithms has been well-analyzed based on similar approaches *simple cycle shrinking*, *generalized selective shrinking* and *generalized truth dependence shrinking*([56][57]). Extending those approaches, we present *cyclic MR-MDFG shrinking* and its optimal solution, based on which the algorithm for optimal scheduling on cyclic MR-MDFG shrinking is proposed, and an upper bound of the number of processors for exploiting parallelism within MDFG or MR-MDFG is also given.

The cyclic MR-MDFG shrinking is an adaptation from the procedure of generalized selective shrinking. The only difference in between is the problem modeling. Our contribution is to reformulate the problem modeling of cyclic MDFG scheduling such that the methodology and results in [56][57] can be employed in this chapter.

A pair (J,D) can be used to characterize the *uniform dependence algorithm*, where J is the index set or iteration space. Each element(cell) in J is an n-tuple column vector corresponding to one iteration. Matrix D is the dependence matrix, with m columns, each of which is a dependence vector $D_i, i = 1, \cdots, m$. J is considered as convex polyhedron index sets, which can be described by $R = \{x : Ax \leq b, A \in Z^{a \times n}, b \in Z^a, a \in N^+\}$, and $J \subset \{j : j \in R \text{ and } j \in Z^n\}$.

A generalized selective shrinking[56] is a mapping $\Gamma_\pi : J \rightarrow N$ such that $\Gamma_\pi(j) = \lfloor(\pi j + c)/disp_\pi\rfloor, j \in J$, where the row vector π is called the schedule vector specifying $\Gamma_\pi, disp_\pi = min\{\pi D_i : D_i \in D\} > 0, GCD(\pi_1, \cdots, \pi_n) = 1$ and $c = min\{\pi j : j \in J\}$. For algorithm (J,D), the total execution time (with time-optimal linear schedule) by the generalized selective shrinking is $t = \lceil(max\{\pi(j_1 - j_2) : j_1, j_2 \in J\} + 1)/(disp_\pi)\rceil$. The derivation and an explanation for this equation can be found in [56]. Thus the problem of finding the optimal generalized shrinking for minimum execution time can be formulated as: find π to minimize $f = (max\{\pi(j_1 - j_2) : j_1, j_2 \in J\})/(min\{\pi D_i : D_i \in D\})$ subject to (1) $\pi D > 0$ (2) $GCD(\pi_1, \cdots, \pi_n) = 1$.

The cyclic MR-MDFG shrinking is an extension on the generalized selective shrinking. Consider an MR-MDFG that is translated into a single-rate MDFG by *Procedure 4.2*. Since non-circular paths can be executed with arbitrary inter-iteration parallelism, only cycles in the MR-MDFG are considered. Suppose there are m cycles $C_1, \cdots, C_m$ in the MR-MDFG. For $1 \leq i \leq m$, let D_i be the summed delay-weights of the edges on C_i, and T_i be the summed execution time of the nodes on C_i. The

number of cycles in the translated MDFG is derived per Property 4.8 and Property 4.13. The total delay-weights of each distinct cycle in the translated MDFG can be derived per Property 4.10, Property 4.15 and Property 4.16. J is the iteration space for the algorithm represented by the original MR-MDFG, and the iteration space for the translated MDFG would be $J' = J/(R_C R_T)$ based on *Property 4.17*. In linear scheduling, and with the availability of parallel processors, the execution time for C_i is $t_i = T_i \times \lceil (max\{\pi(j_1 - j_2) : j_1, j_2 \in J'\} + 1)/(\pi D_i) \rceil$, where π is the schedule vector. In executing C_i the same as in [56], the computation indexed by j_A in iteration space J' is executed only after the execution of computation indexed by $j_B - D_i$, upon which j_A depends. This follows the constraint $D_i\pi > 0$. Given the availability of parallel processors, the parallel execution time for all cycles is $t = MAX\{T_i \times \lceil (max\{\pi(j_1 - j_2) : j_1, j_2 \in J'\} + 1)/(\pi D_i) \rceil\}$, where the maximum is taken over all cycles in the MDFG translated from the original MR-MDFG. This formula is called *cyclic MR-MDFG shrinking*.

The optimal solution to cyclic MR-MDFG shrinking is to find π so as to minimize t or to minimize $f = (max\{\pi(j_1 - j_2) : j_1, j_2 \in J'\})/(min\{\pi D_i/T_i : D_i \in D\})$ subject to (1) $\pi D > 0$ (for respecting the dependence constraints and for the cycles to be computable) (2) $GCD(\pi_1, \cdots, \pi_n) = 1$, where delay-matrix D is with m columns, each of which is a vector D_i $(i = 1, \cdots, m)$. Construct matrix D' with m columns each of which is $(D_i/T_i) \times c, i = 1, \cdots, m$, where c is a minimum positive integer such that elements in D' are integers. In the formula of generalized selective shrinking, replacing J and D by J' and D' respectively, we unite the ways in finding solutions for optimal generalized selective shrinking and for optimal cyclic MR-MDFG shrinking. Optimal solution of the linear schedule vector π, denoted as π_0, can be found based on the procedures in [56][57].

We define the *Criticality* of a cycle C_i in the MDFG as $T_i/(\pi_0 D_i)$. A schedule of

a list of s nodes $N_1 \rightarrow N_2 \rightarrow \cdots \rightarrow N_s$ is said to be *contiguous* if the nodes are scheduled without any intermediate gap or idle time. In this chapter, the communication latency in multiprocessor implementation is neglected for focusing on the key topics. Given vector π_0, we define an ortho-plane as a hyperplane that is orthogonal to π_0 in iteration space. Moreover, we define F as the maximum number of lattice points on any ortho-planes in the iteration space. Now we construct the scheduling algorithm for executing the cycles in the MDFG in parallel according to the solution π_0 for the optimal cyclic MR-MDFG shrinking.

Algorithm 4.1: In n-D iteration space J' along the orientation of π_0, for $1 \leq i \leq m$, consider Set S_i including $\pi_0 D_i$ adjacent ortho-planes, with at most $A_i = F \times \pi_0 D_i$ lattice points and with one iteration of cycle C_i in each lattice point. All $\sum_{i=1}^{m} A_i$ iterations of cycles in sets $S_1, \cdots, S_m$ are then ordered, and scheduled according to the decreasing order of the criticalities of cycles. For those iterations of cycles with equal criticality, they are ordered among themselves at random. The nodes in each iteration of cycles are also ordered to form a list so that the precedence constraints are satisfied. A separate processor is assigned for scheduling of each iteration. The nodes of the iteration with the greatest criticality are scheduled contiguously in processor P_1. Then, the nodes of the next iteration in the criticality list are scheduled in processor P_2 such that P_1 and P_2 compute in parallel and the schedules completed so far are preserved. In other words, if some of the nodes of this iteration also belong to the previously scheduled iteration (this might be true if the two iterations correspond two intersecting cycles), then the schedule of these nodes should remain unaltered. This process is repeated similarly in scheduling all other ordered iterations. In scheduling, there may be necessary communications between processors or between a processor and storage to exchange data or load/store data. Once scheduling of all iterations of

C_i in Set S_i $(1 \leq i \leq m)$ is completed, substitute next $\pi_0 D_i$ adjacent ortho-planes in the orientation of π_0 for S_i right away, and continuously repeat the scheduling of iterations of C_i as above by using the same processors among $P_1, P_2, \cdots$ again. This process is repeated until all iterations of all cycles in iteration space J' are scheduled.

Property 4.18: All nodes in any cycle C_i $(1 \leq i \leq m)$ scheduled by *Algorithm 4.1* are computed contiguously, unless C_i has common nodes with another cycle C_j $(1 \leq j \leq m, j \neq i)$, and the criticality of C_i is less than that of C_j.

Property 4.19: *Algorithm 4.1* respects all dependence constraints in the MDFG.

Property 4.20: Given π_0, the constructed *Algorithm 4.1* for scheduling the MDFG achieves the least execution time designated in cyclic MDFG shrinking.

Property 4.21: The complexity in *Algorithm 4.1* is $mlog_2(m)$ (m is the number of cycles in the MDFG), for sorting the criticalities of cycles in the MDFG.

Property 4.22: When we explore the inter-iteration parallelism within the rate-balanced multirate multidimensional digital signal processing algorithms represented by the MR-MDFG: G=(V,E,T,D,M), via intercalation, unfolding and cyclic MR-MDFG shrinking, the complexity is made up of three parts: part I is O(VE), to find cycles in the MR-MDFG (using Bellman-Ford algorithm), and to calculate rate R_i for each node U_i (as *Step 1* and *Step 2* of *Procedure 3.1*), Q, R_C, R_T, m and delay-weights of edges after unfolding (as in *Procedure 4.1* , *Procedure 4.2* and their properties); part II is to find optimal solution π_0 to cyclic MDFG shrinking, and the complexity analysis is the same as for generalized selective shrinking, which is found in [56][57], mostly dependent on n (where n is the number of dimensions and small in practical algorithms); part III is $mlog_2(m)$, to construct *Algorithm 4.1*, where m is the number of cycles in the MR-MDFG.

CHAPTER V

EXPERIMENTS AND DESIGN EXAMPLES

This Chapter presents our experimental results and design examples for architectural designs in exploiting parallelism in multidimensional multirate DSP applications based on the theories and design methodologies presented in Chapter II, III and IV. The theories and methodologies presented in previous chapters are applicable to the general class of multidimensional multirate DSP algorithms which include many practical applications in acoustic, image and video processing systems. Due to the space limit in this dissertation, we provide three major representative results as the applications of the theoretical work reported in the previous chapters.

A. Non-RAM-Based and Control-Distributed Deigns of Zerotree Construction Systems

As part of our study, various experimental results have been obtained regarding the implementation of architectures proposed in Chapter II. We have achieved the gate-level synthesis of the architecture with Cadence Verilog HDL simulation package. The simulation results have shown advantages in our designs in contrast with those RAM-based architectures for implementing the algorithm of zerotree construction. We also compare the gate-level synthesis of our architecture for zerotree construction with the results of the architecture for basic wavelet transforms (without zerotree construction). The experimental results have shown that the execution time, hardware cost and delay are comparable although the algorithm of zerotree construction is much more complicated than the algorithm of basic wavelet transform.

In simulation of the module of Processing Unit for PU_1, PU_2 and PU_3, we chose to implement the computation of Harr-basis[17] integral wavelet filtering using shifters

Table I. The implementation of non-RAM-based zerotree construction

The width of image	The number of gates	Clock cycles
32	8388	1264
64	16808	4576
128	31428	17344
256	62148	67456
512	122958	265984

Table II. The gate-level implementation of RAM-based zerotree construction

The width of image	The number of gates	Clock cycles
32	1024B RAM cells + 2388	2592
64	4096B RAM cells + 4436	10304
128	16384B RAM cells + 8532	41088
256	65536B RAM cells + 16724	164096
512	262144B RAM cells + 33108	655872

and adders based on calculations of integers. The computation of other wavelet-basis filterings can be implemented similarly when using general structures of floating-point adders. All devices are locally controlled and the hardware connections are localized and optimized. The size of input image is $N \times N$, with each pixel represented in 8 bits. The input image is fed to the system with one pixel per clock cycle.

To evaluate the performance, we also present a gate-level synthesis of a typical RAM-based design for zerotree construction[18] as a contrast with the architectures proposed in this dissertation. In this design, the image is stored in an off-chip RAM

Table III. The gate-level implementation of DWT

The width of image	The number of gates	Clock cycles
32	3612	1120
64	6684	4288
128	12828	16768
256	25116	66304
512	49692	263680

and accessed by an ASIC processing structure for the wavelet transforms and zerotree construction. The processing structure performs the calculation of hierarchical wavelet transforms on the image, and constructs the zerotree data structure based on the results of wavelet transforms by locating the parent-children relationships with calculation of address pointers.

In literature, researchers have proposed many RAM-based architectural designs for zerotree coding ([18][19][20]). All of these designs use off-chip RAMs since the sizes of input images are much bigger than the size of what on-chip caches can hold. There exist bottlenecks in nature limiting the processing rate in these designs — the frequent data transfers between the processor and the RAM, the limitation on the off-chip RAM bus bandwidth and on the size of the data bus. The difference between access rates from off-chip RAMs and access rates from on-chip storage can be referenced from [28][29]. Generally, access from off-chip RAMs is at least tens of times slower than access from on-chip FIFOs. The comparison in Table I and II between our design of zerotree construction and a typical RAM-based implementation shows overwhelming advantages of the non-RAM-based design of zerotree construction against the RAM-based implementation of zerotree construction. The number of gates used in the

implementation and the execution time in clock cycles are shown in Table I, II and III.

In the following we present the complexity analysis of our architectural designs for arbitrary level of wavelet transforms, generic wavelet filters and data precisions. Since L (the width of wavelet filters) is far less than N (the width or length of input image) and the size of boundary effect of wavelet transforms is only dependent on L, in the analysis we ignore the boundary effect to simplify the expressions. The area of the architecture in Fig. 7 in Chapter II is dominated by PUs and TUs since all connections are restricted in locality. A PU contains pL MACs (Multiplier and Accumulator Cell), where p is the number of precision bits of data, thus three PUs contain 3pL MACs, and the area for these MACs is O(pL). Because a TU is necessary for the column-major filtering in every level transform, and the number of cells in TU at the i^{th} level transform is N(L+2^i), the area for TUs is N(L+2^{m+1}-1), where m is the number of levels in DWT. Assuming 2^{m+1} is a constant C, the whole area of the architecture for m level DWT is $\mathbf{A}$=O(pNL). Noting that a new datum is calculated and outputted as a pixel arrives every cycle, and the output size is not more than input size if the boundary effect is disregarded, we have the system's latency (execution time) $\mathbf{T}$ as N^2 clock cycles. Thus the product of $\mathbf{A}$ and $\mathbf{T}$ for the system is O(pN^3L), where pN^2 is the input size of the algorithm. The hardware utilization of PU_1 and PU_2 is 100%. The utilization of PU_3 for m levels of DWT can be figured out by the comparison between the computation tasks of PU_3 and PU_1 or PU_2. PU_1 and PU_2 respectively processes a half amount of the computation for the first level of 2-D DWT; on the other hand, PU_3 with the same hardware structure takes all computation tasks for the other levels of 2-D DWT. Considering that the computation amount for every level of 2-D DWT is four times less than that of the previous level, we have the utilization of PU_3 equal to $(\sum_{i=1}^{m-1}(T/4^i))/(T/2)$, where T

Table IV. The performance analysis for the zerotree construction architecture

Device	Area	Execution Time	Hardware Utilization
PU_1	$O(pL)$	N^2	100%
PU_2	$O(pL)$	N^2	100%
PU_3	$O(pL)$	N^2	$2\sum_{i=1}^{m-1}(1/4^i)$
ALL PUs	$O(pL)$	N^2	$2/3 + (2/3)\sum_{i=1}^{m-1}(1/4^i)$
TU	$O(pNL)$	N^2	100%

is the computation amount for the first level of 2-D DWT. Thus the total hardware utilization of three processors is $(2 + 2\sum_{i=1}^{m-1}(1/4^i))/3$, which is around 90% for the 2-D DWT of more than 4 levels of transform. The utilization of TUs is 100%. The result of performance analysis has been summarized in Table IV.

The proposed architectures can be extended to the computation of more complex algorithms. For instance, with the coefficients grouped in zerotrees, it is easy to generate the zerotree-coding symbols (as the third step of EZW algorithm) based on the architecture in Fig. 7 in Chapter II. Put detectors for insignificant coefficients and registers at output ports and connect the registers at those output ports corresponding to the relation of parent and children. A symbol of IZ is generated and kept in the pertinent register if one coefficient is detected to be insignificant. According to the scheme of EZW, this register is then reset to the symbol of ZTR either if all of its corresponding children are ZTRs or if the children are leaves of the zerotree and IZs. Then the registers of the ZTR's children are set as *bubbles* so that nothing is output to the generated symbol stream. Also the symbols POS and NEG [13] can be generated by trivially detecting the signs of the significant coefficients. Thus, the zerotree-coding symbols can be streamed out via system output ports in real time

when the input image is fed to the system.

B. Fully Retiming the MR-MDFG That Represents the 3-D DWT

In this section we propose a simple design example: retiming the MR-MDFG which represents the algorithm of 3-D discrete wavelet transform based on the design methodology in Chapter III. The use of the algorithm of 3-D Discrete Wavelet Transform (3-DWT) on the compression and the processing of medical imaging, image database and video signals becomes a hot topic lately ([44][45][37]). A challenge to the design of 3-DWT that is applied onto a 3-D data set is the intensity of the computation. As many other MD applications, the design of application specific integrated circuits is usually required in order to improve the performance. Fully retiming the MR-MDFG that represents the algorithm of 3-DWT is necessary to exploit the inter-operation parallelism within the 3-DWT when mapping the 3-DWT to hardware which uses serial input format to process a 3-D input data set.

The complete retiming formulations proposed in this dissertation allow the designs for more complicated cases of serial processing order and computation structure of wavelet filtering, but due to the purpose of a concise illustration of our ideas in Chapter III, we assume a simple case of 3-DWT in which the serial processing order is simply frame-wise, column-wise and then row-wise and the input data stream is fed to system with one item at each clock cycle.

The MR-MDFG corresponding to 3-DWT is illustrated in Fig.20, where the row-wise and column-wise filters are 4-tap filters, and the frame-wise filter is a 5-tap filter, considering that a wavelet basis used on the frame orientation is different from other orientations in some practical applications [44]. The computation nodes No. 2–5, No. 9–12 and No. 16–20 are multipliers where the data is multiplied by a filter coefficient.

Other nodes are adders. Node No. 1 takes input data and node No. 24 sends out the output. The offset-weights and multirate-weights of edges are illustrated in Fig.20.

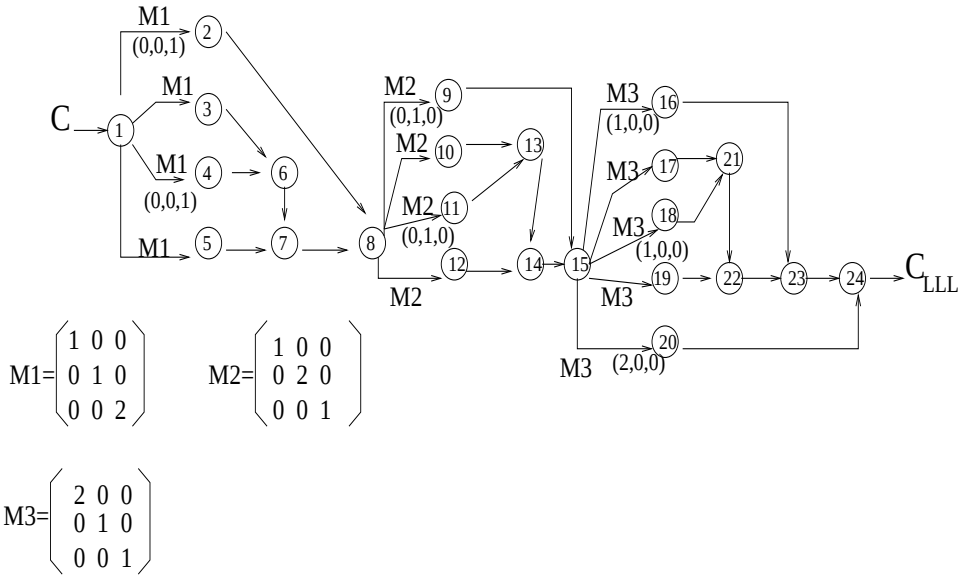


Fig. 20. The MR-MDFG for 3-DWT and the retiming solutions

By MD intercalation, the MR-MDFG in Fig.20 is partitioned into four single-rate subgraphs G_0, G_1, G_2 and G_3. G_0 contains node No. 1 alone, G_1 contains nodes No. 2–8, G_2 contains nodes No. 9–15, and G_3 contains nodes No. 16–24. The rate of G_0 is a unit matrix whose diagonal elements are 1's and other elements are 0's. $M_1, M_1 \times M_2$ and $M_1 \times M_2 \times M_3$ as shown in Fig.20 are the rates of G_1, G_2 and G_3 respectively. Based on their rates, the subgraphs are moved by various offsets within the iteration space according to the procedure of MD intercalation. Three grids based on the rates of M_1, M_1M_2 and $M_1M_2M_3$ are created in iteration space for G_1, G_2 and G_3 respectively, and the instances of subgraphs are always located in their respective grid only. Following the complete formulation of retiming operations on MD intercalated iteration space, we have that the retiming operations on the nodes in these subgraphs are re-defined in their respective grids, i.e., the retiming vectors

correspond to the moving distances within the grids.

The size of 3-D input data set is assumed to be $N \times N \times N$. The group of vectors representing the processing order is $\{(S_1, S_2, S_3\}$, where $S_1 = (0,0,1)^T$, $S_2 = (0,1,0)^T$, and $S_3 = (1,0,0)^T$. The eigen-function F(X)=N^2(X,S_1)+N(X,S_2)+(X,S_3). Assume that the system clock period is required to be 1, i.e., it is restricted that any edge in the MR-MDFG should have non-zero valued offset-weight after retiming. Thus the final retiming formulation is summarized to: minimize COST=$\sum_{v \in V} R_v$ under the restrictions A: $R_u \geq F(d(e) + r(u) - r(v)$ and B: $F(d(e) + r(u) - r(v)) \geq 1$ for any edge $u \rightarrow v$ in the MR-MDFG.

To find an approximately optimal solution of retiming vectors for this problem in polynomial time, we use a *rate-oriented* retiming, with the techniques similar to the *orthogonal 2-D retiming* introduced in [32]. By rate-oriented retiming, or considering the solutions of retiming vectors under the restriction A and B along the orientations of M_1, M_2 and M_3 separately, we derive the retiming vectors for the computation nodes as illustrated in Fig.20. Based on these solutions, the MR-MDFG for 3-DWT is fully retimed such that any edge has a non-zero valued offset weight, so the computation nodes in the MR-MDFG can be executed in parallel and thus the system clock period is minimized to 1. The MR-MDFG is illustrated in Fig.21 after all retiming operations are applied.

Following the fully retimed MR-MDFG, we propose the architectural design of 3-DWT as shown in Fig.22. All calculation units work simultaneously and the clock cycle period is minimized. In Fig.22, a structure called Shift Unit (SU) is employed to store the intermediate data corresponding to offset-weights of edges in Fig.21. The structure of SU that consists of a register array with $A \times B$ cells (or registers) is illustrated in Fig.22. The systolic data flow among the cells is specified by the arrows. A cell transfers its content to the next adjacent cell once it receives a datum

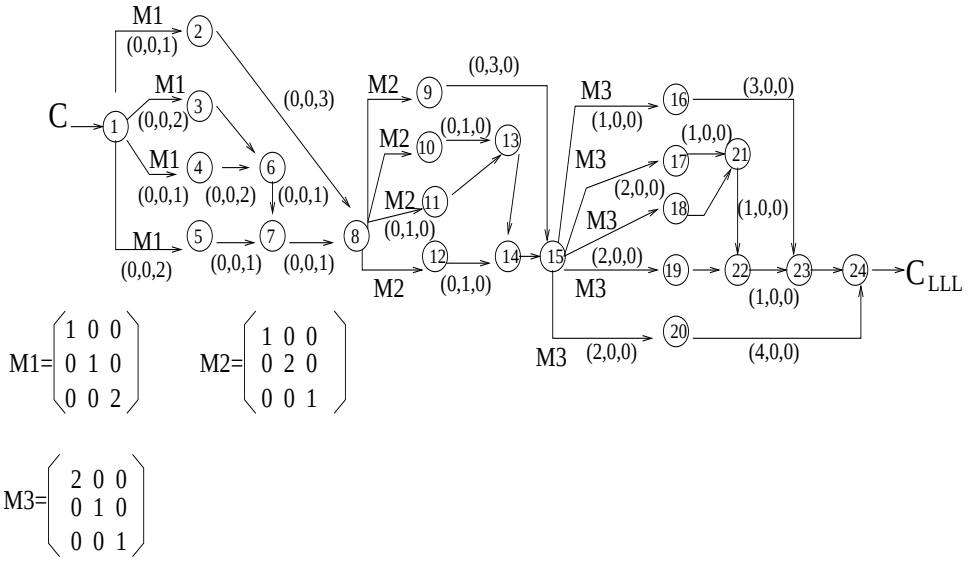


Fig. 21. The MR-MDFG for 3-DWT after retiming operations

from its preceding cell in each clock cycle. From the scheme of systolic data flow, it is apparent that Fig.22 is a snake-like representation of a linear shift register array.

As part of our study, we have simulated a Register Transition Level (RTL) implementation of the architecture for 3-DWT in Fig.22 in Verilog Hardware Description Language (Verilog HDL). To evaluate the performance, we also present a Verilog HDL implementation of a typical architecture of 3-DWT following the design in [45] based on addressable memories. The simulation results have demonstrated considerable performance advantages in our design over the typical architecture for implementing the 3-DWT algorithm. In Verilog implementation of calculation units for wavelet filtering, we chose to implement the computation of Harr-basis[43] wavelet filtering based on integer operations. The computation of other wavelet-basis filtering can be implemented in similar architectures using floating-point units. The 3-D input data set has $N \times N \times N$ elements, with each one represented by a byte. Note that the clock cycle period in our design is tens of times shorter than that in [45] because all function units in our design work in parallel based on the fully retimed MR-MDFG.

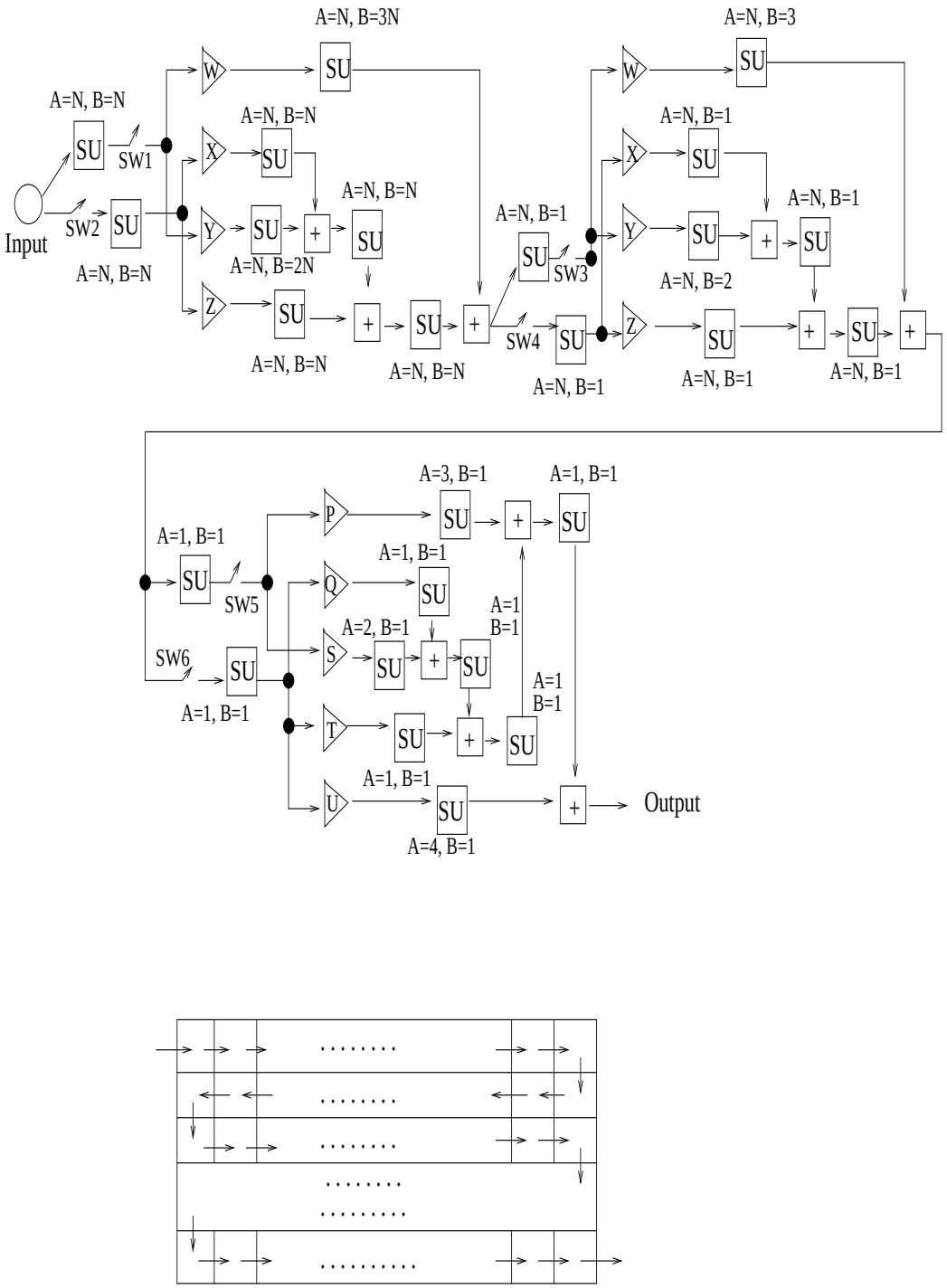


Fig. 22. The architectural design for 3-DWT

C. Exploiting Maximum Inter-Iteration Parallelism in Multidimensional Multirate Systems

This section we provide an application of the design methodology proposed in Chapter IV for exploring the bound of the inter-iteration parallelism in multidimensional multirate digital systems. There are many practical DSP algorithms with feedback recursions such as SFQ[15][16], adaptive-computation based algorithms and some other algorithms with data-dependent computation structures. At low level of signal processing, the computation structures of algorithms are basically *static* or data-independent. The algorithms include fundamental wavelet transforms and main parts of JPEG and MPEG algorithms. Many of those computation structures can be represented in acyclic data flow graphs without using feedback recursions. On the other hand, for high level signal processing algorithms such as the SFQ, pattern recognition in acoustic signals, face detection and fingerprint detection, systems are mostly adaptive and/or feedback-based in nature, and the computation structures are *dynamic*, i.e., data-dependent. Those algorithms can usually be represented by cyclic data flow graphs where data might flow within the system along circular paths because of the feedbacks. In literature there have been many software implementations for high-level signal processing algorithms and data-dependent computations, but many are of low performance in terms of execution time and can be used seldom in real-time applications. With more and more advanced VLSI techniques nowadays we can integrate larger circuits into chips and have a broader range of options in designing complex circuit systems. Within such circumstances, this dissertation makes contributions for a theoretical understanding of the parallelism exploitation which is essential in designs of hardware implementations for multidimensional DSP algorithms.

In this section we briefly introduce the implementation of a typical multirate

multidimensional DSP algorithm with feedback recursions. We use the SFQ algorithm [15] as the prototype for our example to implement. The SFQ scheme in [15] is an adaptive transform coding framework that extends basic wavelet transforms to the signal-dependent wavelet packet transforms. It is one of the best image coding algorithms in terms of coding efficiency that have been reported. The optimal scheme of SFQ is a high-complexity algorithm whose computation structure includes feedback recursions where the input image is transformed with all wavelet packet trees (i.e., arbitrary frequency resolutions), and a specific wavelet packet decomposition and a scalar quantizer are selected based on rate-distortion measurements. In a near-optimal and practical solution [15] whose computation complexity is relatively low, the schemes of transform and quantizer designs are decoupled, and the data are transformed and quantized based on rate-distortion values partially. In literature there are only software implementations of SFQ algorithms which are executed in sequential ways. No parallelism within the SFQ algorithms has ever been explored or exploited.

Generally, the SFQ computation structures can be represented in cyclic data flow graphs with delay-weights on the edges connecting various calculation units and feedback cycles for data-adaptive computations. Fig.23 shows an illustration of the MR-MDFG representation for the calculation of a subband of data as a part of the algorithm of near-optimal wavelet packet SFQ. For clarity and brevity, we specify the nodes in the MR-MDFG as computation units with buffers inside and they can perform wavelet filtering or other complicated computations. The edges with delay and multirate weights correspond to the dependence relationships between calculation units. The values of the weights are dependent on particular wavelet filters. In Fig.23, i.e., the original MR-MDFG for the SFQ algorithm, node A and B are regarding the computations of wavelet filtering; the other computation nodes are responsible for

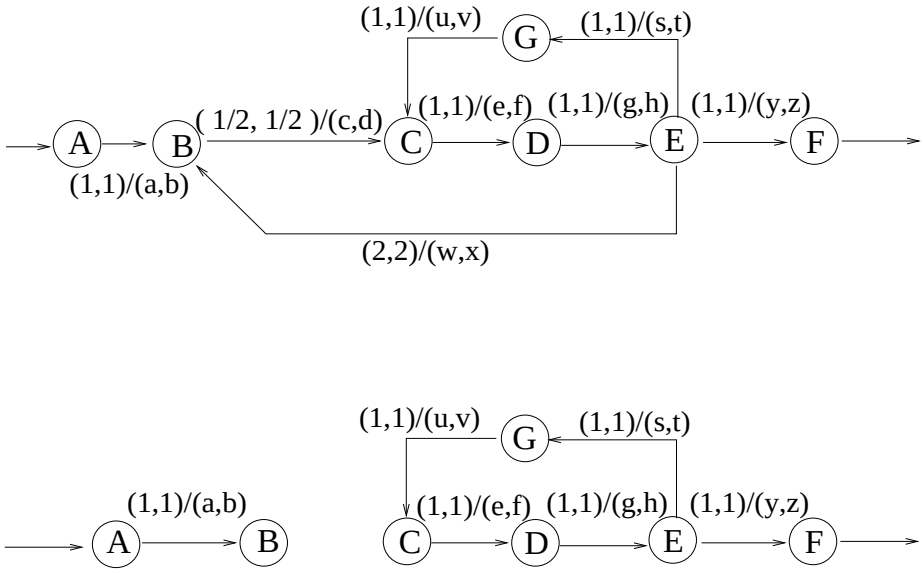


Fig. 23. The MR-MDFG representation of an SFQ algorithm and the partition of the MR-MDFG

the computation tasks of quantizations, error estimations and etc. Each edge in the graph is labeled with two vectors, e.g., edge $B \rightarrow C$ is labeled with $(1/2, 1/2)/(a, b)$ where the first vector $(1/2, 1/2)$ is the multirate weight and the second vector (a, b) is the delay weight. The symbols $a, b, c, \cdots, x, y, z$ represent integers corresponding to the delays determined by the details of computation equations which include the selection of wavelet filters and etc. For brevity and without loosing generality, let $c = 3, d = 2, e = -2, f = 2, g = -1, h = -3, s = 2, t = 3, u = 3, v = 2, w = 6$ and $x = 8$.

Based on the procedures and properties in Chapter IV, we can translate Fig.23 into a single-rate MDFG. Following *Procedure 3.1*, the original MR-MDFG is partitioned into two disjoint unit subgraphs, shown as in Fig.23, by cutting off all multirate edges in Fig.23. Suppose that subgraph Q_1 contains A and B; subgraph Q_2 contains C, D, E, F and G. The rates of the tow subgraphs are $\mathbf{R_1} = (1, 1)$ and $\mathbf{R_2} = (1/2, 1/2)$ respectively per *Definition 4.3*. According to *Step 2* of *Procedure*

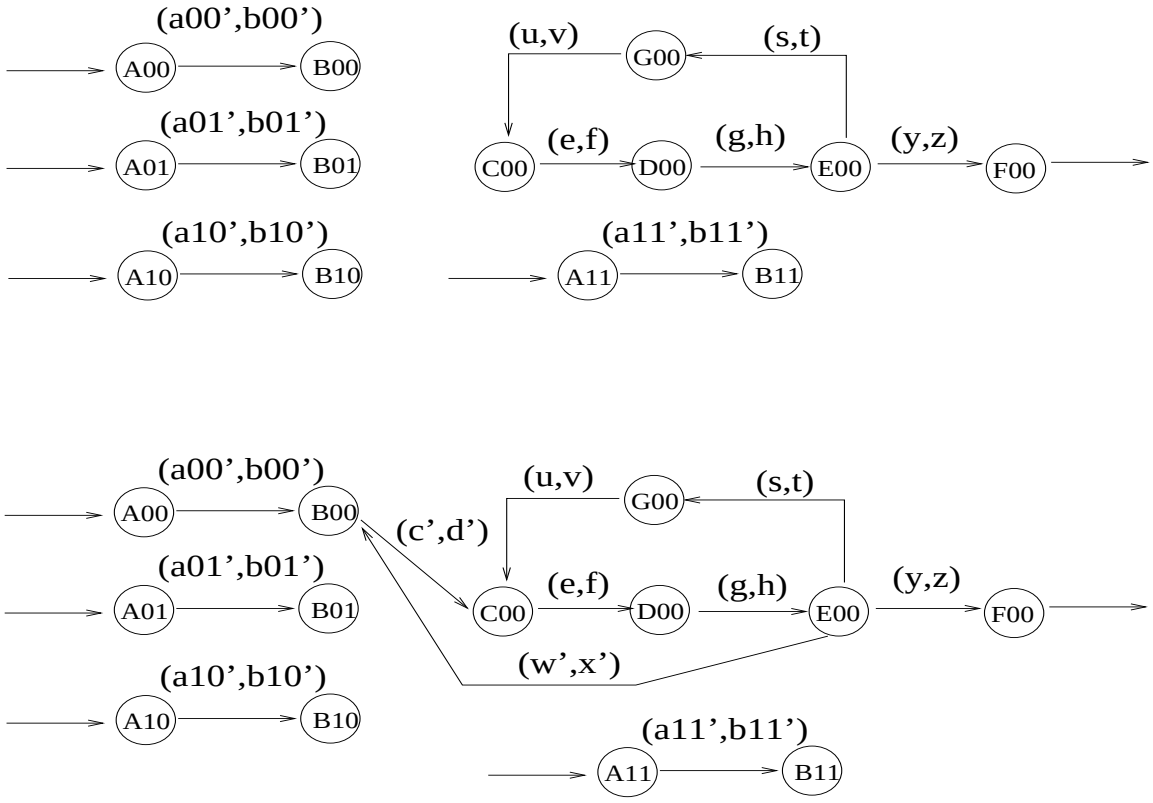


Fig. 24. The final translated single-rate MDFG

4.2, we obtain $\mathbf{R_T} = LCM(\mathbf{1/R_1}, \mathbf{1/R_2}) = (2, 2)$, and take $\mathbf{R_1 R_T}$ and $\mathbf{R_2 R_T}$ as the unfolding factors to unfold Q_1 and Q_2, i.e., apply *Procedure 4.1* on the unit subgraphs. The number of cycles and the sum of the delays on the cycles after unfolding are calculated per *Property 4.15* and *Property 4.16*. It is straightforward in this case since only Q_2 has a cycle and the unfolding factor for this subgraph turns out to be $(1, 1)$. All unfolded subgraphs after *Procedure 4.1* are shown in Fig.24. The new delay vectors on edges in the unfolded graphs are determined in *Procedure 4.1*. As the last step to transform the orignal MR-MDFG into a single-rate MDFG, it is necessary to connect the disjoint subgraphs by employing *Step 3* of *Procedure 4.2*. The delay vectors on edges connecting node instances from separate subgraphs are calculated per equations in *Step 3* of *Procedure 4.2*.

The number of all cycles and the delay weights on the cycles after *Procedure 4.2* are determined per *Property 4.11* through *Property 4.15*. The result of the translation of the original MR-MDFG into a single-rate MDFG is shown in Fig.24. Let the sum of delay vectors on Cycle 1, $B01 \rightarrow C00 \rightarrow D00 \rightarrow E00 \rightarrow B01$, be $\mathbf{d_1}$. $\mathbf{d_1} = (\mathbf{c}', \mathbf{d}') + (\mathbf{e}, \mathbf{f}) + (\mathbf{g}, \mathbf{h}) + (\mathbf{w}', \mathbf{x}') = (\mathbf{3}, \mathbf{5})$. Let the sum of delay vectors on Cycle 2, $C00 \rightarrow D00 \rightarrow E00 \rightarrow G00 \rightarrow C00$, be $\mathbf{d_2}$. $\mathbf{d_2} = (\mathbf{u}, \mathbf{v}) + (\mathbf{e}, \mathbf{f}) + (\mathbf{g}, \mathbf{h}) + (\mathbf{s}, \mathbf{t}) = (\mathbf{2}, \mathbf{4})$. To consider the exploitation of the inter-iteration parallelism, we only investigate the subgraph containing cycles after *Procedure 4.2* as shown in Fig.25. According to the procedure in [56], we obtain the optimal solution of linear scheduling vector π_0 equal to $(5, 4)$ based on the values of $\mathbf{d_1}$ and $\mathbf{d_2}$ as shown in Fig.26. Given π_0, we can apply *Algorithm 5.1* in scheduling the computation nodes and segment the 2-D iteration space along the orientation of π_0 as in Fig.26. Note that the step size for segmentations is determined by the length of the projection of $\mathbf{d_1}$ or $\mathbf{d_2}$ on the orientation of π_0 for Cycle 1 or Cycle 2 respectively. To put it simply, the computation nodes on the cycles of Fig.25 in each lattice point (corresponding to an iteration) of the iteration space can be executed concurrently within every segmentation.

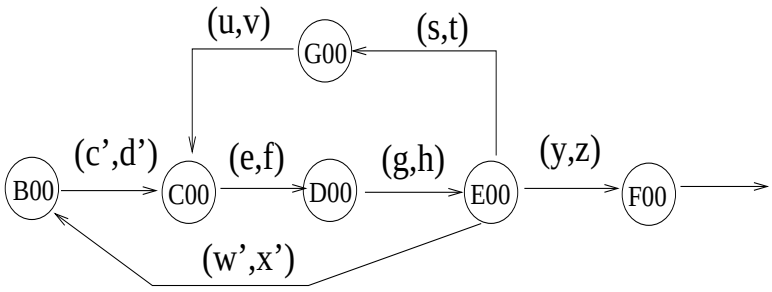


Fig. 25. The subgraph of the single-rate MDFG containing cycles only

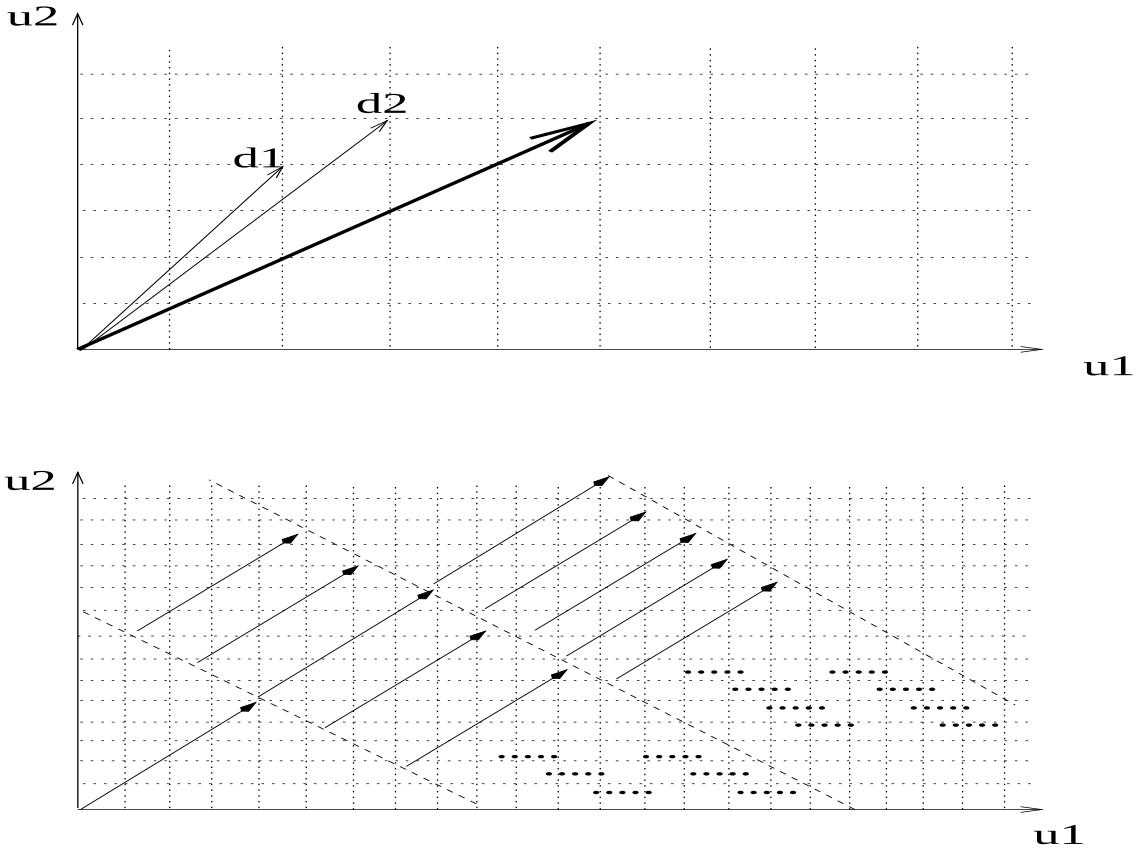


Fig. 26. An illustration of the implementation of the SFQ algorithm in 2-D iteration space

CHAPTER VI

CONCLUSION

In this dissertation, we have conducted theoretical work and proposed several design methodologies on computer architectural designs for high-performance computing of multidimensional multirate digital signal processing applications with inter- and intra- iteration parallelism being exploited. Wavelet-based digital algorithms are typical instances of such digital signal processing applications. Experimental results and design examples for the applications of the design methodologies are provided. The results have demonstrated better performance in terms of hardware cost and the execution time than other methods and designs in existing literature.

Chapter II has proposed a methodology for non-RAM-based architectural designs of wavelet algorithms based on novel nonlinear I/O data space transformations. Exploiting common features of computation locality and multirate signal processing within general wavelet-based algorithms, Chapter II proposes a series of novel nonlinear transformations in I/O data space analysis and obtains regularized and/or merged structures of dependence graphs for any wavelet-based algorithms. Such nonlinear transformations for newly-modeled data dependence graphs lead to non-RAM-based architectures for hardware implementations of general wavelet-based algorithms.

The series of nonlinear I/O data space transformations are proposed as a theoretical basis of architectural designs for generalized wavelet-based algorithms, but it is infeasible to introduce designs for all complex wavelet-based algorithms due to the space limit in Chapter II. We use the zerotree construction algorithm as the representative and propose a non-RAM-based design in which the input image is recursively decomposed by DWT and the zerotrees are constructed simultaneously. In contrast with the reported architectures ([18][19][20]) for wavelet zerotree construc-

tions that use large off-chip RAMs in building zerotrees and employ either memory address pointers or data rearrangement, our architectures only need much smaller on-chip FIFOs leading to the elimination of off-chip communications and the increase of processing rates.

The philosophy underlying our proposed design in Chapter II is a full exploitation of the locality of the computation. The computation of wavelet-based zerotree coding is strongly featured by the computation locality in that the calculations of coefficients on a certain zerotree only depend on the same local sub-area of the 2-D inputs. This desirable feature has been fully exploited in Chapter II by concurrent calculations of children and their parent in the rearranged DWT, so that the necessary storage for intermediately calculated data is reduced to a great extent. Thus some items of intermediate data need not be held for future calculations if the coefficients on the corresponding zerotrees are scheduled to be calculated together and earlier. This primary approach based on our novel nonlinear I/O data space transformations is not only used to derive designs for the algorithm of zerotree construction, but for many other general complex wavelet-based digital systems.

The technique of MD retiming has been proposed in literature to improve the circuitry performance of MD single-rate DSP systems. However, the theoretical analysis in Chapter III has demonstrated that this technique can not be extended to ARBITRARY MD multirate DSP systems. The class of rate-balanced MD multirate DSP algorithms in which all wavelet-based algorithms and most of other practical multirate applications are included has been identified theoretically in Chapter III. We theoretically generalize the MD retiming operations from MD single-rate DSP systems to rate-balanced MD multirate DSP systems, where all operations in single-rate systems can be taken as special cases of operations in multirate systems. An important new technique applied in iteration space prepared for fully retiming rate-

balanced MR-MDFG's, MD intercalation, is proposed in Chapter III. Based on it, a complete UNIFIED formulation of retiming operations for MD multirate DSP algorithms represented by rate-balanced MR-MDFG's is proposed, where clock period constraint, cost minimization, causality constraint, and arbitrarily linear processing order are addressed. The complete formulation constructs the theoretical basis for optimally exploiting the intra-iteration parallelism in rate-balanced multirate DSP systems in Chapter III. Finally, a design example of optimally retiming the MR-MDFG that represents the 3-DWT algorithm is proposed in Section B of Chapter V as the application of this methodology.

Chapter IV has proposed the methodology of inter-iteration parallelism exploitation by combining (1) the multidimensional intercalation, (2) the multidimensional unfolding, (3) the translation of a multirate multidimensional data-flow graph into a single-rate multidimensional data-flow graph, and (4) the cyclic MDFG shrinking. Based on these approaches, we have shown how the inter-iteration parallelism is optimally exploited against precedence constraints within multirate multidimensional DSP algorithms in multiprocessor implementation. As a measurement to achieve the full inter-iteration parallelism, an upper bound on the number of processors is given, which is derived from (1) the topology and weights of the MR-MDFG and (2) the shape of the iteration space. Any other multiprocessor implementations with a number of processors beyond this bound do not lead to further improvement.

While this dissertation is mainly directed to a theoretical understanding of parallel processing implementations and a proposal of novel methodologies in architecture designs for MD multirate DSP applications, the work in this dissertation also helps discussing general topics regarding the high-level synthesis of MD DSP systems, on which interested readers are referred to for details in our classified and specialized discussions. The proposal of multidimensional intercalation leads to a unified formu-

lation of hardware designs for multidimensional DSP algorithms by exploring intra- and inter-iteration parallelism. A direct mapping of multirate DSP algorithms to hardware would require data to move at different rates on the chip, which involves complicate routing and synchronization of multiple clock signals. The methodologies of multidimensional unfolding and translating an MR-MDFG into an MDFG lead to mapping a multirate DSP algorithm into a single-rate VLSI architecture, where the entire system operates with the same clock signal. No sub-clocks are necessary and the hardware efficiency is improved significantly.

REFERENCES

[1] M.. Cotronei, L. B. Montefusco and L. Puccio, "Multiwavelet analysis and signal processing," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, pp. 970-987, Aug. 1998.

[2] F. G. Meyer, A. Z. Averbuch and J. O. Stromberg, "Fast adaptive wavelet packet image compression," *IEEE Trans. on Image Processing*, vol. 9, pp. 792-800, May 2000.

[3] H. Sava, M. Fleury, A. C. Downton and A. F. Clark, "Parallel pipeline implementation of wavelet transforms," *IEE Proc. on Vision, Image and Signal Processing*, vol. 144, pp. 355-360, Dec. 1997.

[4] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 1, pp. 191-202, June 1993.

[5] A. Grzeszczak, M. K. Mandal and S. Panchanathan, "VLSI implementation of discrete wavelet transform," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 4, pp. 421-433, Dec. 1996.

[6] Seung-Kwon Pack and Lee-Sup Kim, "2D DWT VLSI architecture for wavelet image processing, *Electronics Letters*," vol. 34, pp. 537-538, March 1998.

[7] G. Knowles, "VLSI architecture for the discrete wavelet transform," *Electronics Letters*, vol. 26, pp. 1184-1185, 19 July 1990.

[8] T. C. Denk and K. K. Parhi, "VLSI architectures for lattice structure based orthonormal discrete wavelet transforms," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, pp. 129-132, Feb. 1997.

[9] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: from single chip implementations to mappings on SIMD array computers," *IEEE Trans. on Signal Processing*, vol. 43, pp. 759-771, March 1995.

[10] M. Cotronei, D. Lazzaro, L. B. Montefusco and L. Puccio, "Image compression through embedded multiwavelet transform coding," *IEEE Trans. on Image Processing*, vol. 9, pp. 184-189, Feb. 2000.

[11] Gang Lin and Ze-Min Liu, "The application of multiwavelet transform to image coding," *IEEE Trans. on Image Processing*, vol. 9, pp. 270-273, Feb. 2000.

[12] F. Kurth and M. Clausen, "Filter bank tree and M-band wavelet packet algorithms in audio signal processing," *IEEE Trans. on Signal Processing*, vol. 47, pp. 549-554, Feb. 1999.

[13] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Signal Processing*, vol. 41, pp. 3445-3462, 1993.

[14] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, pp. 243-250, June 1996.

[15] Zixiang Xiong, K. Ramchandran and M. T. Orchard, "Wavelet packet image coding using space-frequency quantization," *IEEE Trans. on Image Processing*, vol. 7, pp. 892-898, June 1998.

[16] Zixiang Xiong, K. Ramchandran and M. T. Orchard, "Space-frequency quantization for wavelet image coding," *IEEE Trans. on Image Processing*, vol. 6, pp. 677-693, May 1997.

[17] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding, Prentice Hall*, pp. 112-130, June 1995.

[18] Li-Minn Ang, Hon Nin Cheung and K. Eshraghian, "VLSI architecture for significance map coding of embedded zerotree wavelet coefficients," *Proc. of 1998 IEEE Asia-Pacific Conference Circuits and Systems*, vol. 1, pp. 627-630, June 1998.

[19] Jongwoo Bae and V. K. Prasanna, "A fast and area-efficient VLSI architecture for embedded image coding," *Proc. of International Conference on Image Processing*, vol. 3, pp. 452-455, June 1995.

[20] J. M. Shapiro, "A fast technique for identifying zerotrees in the EZW algorithm," *Proc. of 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp. 1455-1458, June 1996.

[21] J. Vega-Pineda, M. A. Suriano, V. M. Villalva, S. D. Cabrera and Y. C. Chang, "A VLSI array processor with embedded scalability for hierarchical image compression," *1996 IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 168-171, July 1996.

[22] T. Acharya and Po-Yueh Chen, "VLSI implementation of a DWT architecture," *Proc. of the 1998 IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 272-275, July 1998.

[23] J. Fridman and E. S. Manolakos, "Discrete wavelet transform: data dependence analysis and synthesis of distributed memory and control array architectures," *IEEE Trans. on Signal Processing*, vol. 45, pp. 1291-1308, May 1997.

[24] J. N. Patel, A. A. Khokhar and L. H. Jamieson, "On the scalability of 2-D

wavelet transform algorithms on fine-grained parallel machines," *Proc. of the 1996 International Conference on Parallel Processing*, vol. 2, pp. 24-28, June 1996.

[25] Kwok-Wai Cheung, Chun-Ho Cheung and Lai-Man Po , "A novel multiwavelet-based integer transform for lossless image coding," *Proc. of the 1999 International Conference on Image Processing*, vol. 1, pp. 444-447, March 1999.

[26] G. Lafruit, F. Catthoor, J. P. H. Cornelis and H. J. De Man, "An efficient VLSI architecture for 2-D wavelet image coding with novel image scan," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 7, pp. 56-68, March 1999.

[27] V. Strela, P. N. Heller, G. Strang, P. Topiwala and C. Heil, "The application of multiwavelet filterbanks to image processing," *IEEE Trans. on Image Processing*, vol. 8, pp. 548-563, April 1999.

[28] K. N. Lalgudi and M. C. Papaefthymiou, "Retiming edge-triggered circuits under general delay models," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 1393-1408, Dec. 1997.

[29] B. S. Amrutur and M. A. Horowitz, "Speed and power scaling of SRAM's," *IEEE Journal of Solid-State Circuits*, vol. 35, Issue 2, pp. 175-185, Feb. 2000.

[30] Seung-Jong Choi and J. W Woods, "Motion-compensated 3-D subband coding of video," *IEEE Trans. on Image Processing*, vol. 8, pp. 155-167, Feb. 1999.

[31] T. C. Denk and K. K. Parhi, "Systematic design of architectures for M-ary tree-structured filter banks," *IEEE Signal Processing Society Workshop on VLSI Signal Processing,* vol. 8, pp. 157-166, Sept. 1995.

[32] T. C. Denk and K. K. Parhi, "Two-dimensional retiming," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 7, pp. 198-211, June 1999.

[33] T. C. Denk and K. K. Parhi, "Synthesis of folded pipelined architectures for multirate DSP algorithms," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 6, pp. 595-607, Dec. 1998.

[34] F. Fernandez, A. Sanchez and A. Duarte, "An optimal software-pipelining method for instruction-level parallel processors based on scaled retiming," *Proc. of the 2nd International Symposium on Image and Signal Processing and Analysis,* vol. 1, pp. 405-410, June 2001.

[35] F. Fernandez and A. Sanchez, "Application of multidimensional retiming and matroid theory to DSP algorithm parallelization," *Proc. of 25th EUROMICRO Conference,* vol. 1, pp. 511-518, Sept. 1999.

[36] J. Horstmannshoff and H Meyr, "Optimized system synthesis of complex RT level building blocks from multirate dataflow graphs," *Proc. of 12th International Symposium on System Synthesis,* vol. 1, pp. 38-43, Nov. 1999.

[37] H. Khalil, A. F. Atiya and S. Shaheen, "Three-dimensional video compression using subband/wavelet transform with lower buffering requirements," *IEEE Trans. on Image Processing*, vol. 8, pp. 762-773, June 1999.

[38] Jun Ma, K. K. Parhi and E. F. Deprettere, "Derivation of parallel and pipelined orthogonal filter architectures via algorithm transformations," *Proc. of the 1999 IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 347-350, June 1999.

[39] N. L. Passos, E. H.-M. Sha and S. C. Bass, "Optimizing DSP flow graphs via

schedule-based MD retiming," *IEEE Trans. on Signal Processing*, vol. 44, pp. 150-155, Jan. 1996.

[40] N. L. Passos, E. H.-M. Sha and Liang-Fang Chao, "MD interleaving for synchronous circuit design optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 146-159, Feb. 1997.

[41] N. L. Passos and E. H.-M Sha, "Achieving full parallelism using MD retiming," *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, pp. 1150-1163, Nov. 1996.

[42] V. Sundararajan and K. K. Parhi, "Synthesis of folded, pipelined architectures for multi-dimensional multirate systems," *Proc. of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5, pp. 3089-3092, Jul. 1998.

[43] J. Gong and Chang Wu, "Optimal FPGA mapping and retiming with efficient initial state computation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1595-1607, Nov. 1999.

[44] J. Wang and K Huang, "Medical image compression by using three-dimensional wavelet transformation," *IEEE Trans. on Medical Imaging*, vol. 15, pp. 547-554, Aug. 1996.

[45] Michael Weeks and Magdy Bayoumi, "3-D discrete wavelet transform architectures," *IEEE International Symposium on Circuits and Systems (ISCAS '98)*, vol. 4, pp. 57-60, June 1998.

[46] V. Zivojnovic and R. Schoenen, "On retiming of multirate DSP algorithms,"

IEEE International Conference Proc. on Acoustics, Speech, and Signal Processing, vol. 6, pp. 3310-3313, May 1996.

[47] N. Maheshwari and S. Sapatnekar, "Efficient retiming of large circuits," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 6, pp. 74-83, March 1998.

[48] P. Y. Calland, A. Mignotte, O. Peyran, Y. Robert and F. Vivien, "Retiming DAGs [direct acyclic graph]," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 1319-1325, Dec. 1998.

[49] N. L. Passos and E. H.-M. Sha, "Synchronous circuit optimization via multidimensional retiming Signal Processing," *IEEE Trans. on Circuits and Systems II: Analog and Digital*, vol. 43, pp. 507-519, July 1996.

[50] T. Soyata, E. G. Friedman and J. H. Jr. Mulligan, "Incorporating interconnect, register, and clock distribution delays into the retiming process," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 105-120, Jan. 1997.

[51] P. Y. Calland, A. Darte and Y. Robert, "Circuit retiming applied to decomposed software pipelining," *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, pp. 24-35, Jan. 1998.

[52] Liang-Fang Chao and E. H.-M. Sha, "Scheduling data-flow graphs via retiming and unfolding," *IEEE Trans. on Parallel and Distributed Systems*, vol. 8, pp. 1259-1267, Dec. 1997.

[53] V. Sundararajan and K. K. Parhi, "Synthesis of folded, pipelined architectures for multi-dimensional multirate systems," *Proc. of the 1998 IEEE International*

Conference on Acoustics, Speech and Signal Processing, vol. 5, pp. 3089-3092, 1998.

[54] D. Y. Chao and D. T. Wang, "Iteration bounds of single-rate data flow graphs for concurrent processing, " *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, pp. 629-634, Sept. 1993.

[55] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding, " *IEEE Trans. on Computers*, vol. 40, pp. 178-195, Feb. 1991.

[56] Weijia Shang, M. T. O'Keefe and J. A. B. Fortes, "On loop transformations for generalized cycle shrinking, " *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, pp. 193-204, Feb. 1994.

[57] W. Shang and J. A. B. Fortes, "Time optimal linear schedules for algorithms with uniform dependencies, " *Proc. of the 1988 International Conference on Systolic Arrays*, vol. 1, pp. 393-402, 1988.

VITA

Dongming Peng was born in Heilongjiang, P. R. China. He received both his B.S. and M.S. degrees in Electrical Engineering, in 1993 and 1996, from Beijing University of Aeronautics and Astronautics, Beijing. He taught at Beijing University from 1996 to 1997. Since spring 1998 he has been working toward the degree of Doctor of Philosophy in Computer Engineering in the Department of Electrical Engineering at Texas A&M University. He is currently an Assistant Professor with the Department of Computer and Electronics Engineering, University of Nebraska, Lincoln. His permanent address is: PKI 201A, 1110 South 67th Street, Omaha, NE 68182-0572

The typist for this thesis was Dongming Peng.