STORAGE, RETRIEVAL AND PROCESSING OF VIDEO

by

Radhika S. Grover

Advisor: Prof. Qiang Li

DISSERTATION

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Engineering
in the School of Engineering
of Santa Clara University

June 2003

Santa Clara, California
Abstract

STORAGE, RETRIEVAL AND PROCESSING OF VIDEO

by

Radhika S. Grover

June 2003

Doctor of Philosophy in Computer Engineering

Santa Clara University

Advisor: Qiang Li

We present the design of a video-on-demand system called Storage Area network Multimedia server (SAM). SAM manages a set of storage devices connected by a storage area network, and can be scaled to service thousands of clients with performance guarantees. The architecture, data layout, admission control scheme and support for interactive operations are described. An inter-node data layout scheme called constrained random striping (CRS) is proposed for SAM. CRS uses a novel placement of the stripe units across nodes that are connected on a logical loop, which reduces the start-up latency of a client. The start-up latency is the smallest time duration for which a client must wait between initiating a new request and starting display, in order to prevent jitter. CRS can provide a reduction of up to about 40% in the start-up latency compared to other placement schemes. SAM provides multi-resolution video to clients with different levels of quality. An intra-node placement scheme called Partial-Bundled is developed for variable bit rate multi-resolution video. This scheme can support from 10% to 100% more clients than other schemes depending on system parameters such as round time. We
address the issue of limiting client jitter to within acceptable limits with an admission control scheme. In this scheme, deterministic guarantees are provided for minimum layers of the multi-resolution video, and statistical guarantees are provided for higher layers of multi-resolution video. SAM supports VCR operations such as fast-forward and rewind. We show that the statistical multiplexing of available resources can reduce the resources required to support interactive service on a parallel server. An analytical model is developed to predict the probability with which stream requirements can be met for a given node buffer level. SAM also continues to provide uninterrupted service in the event of a storage device failure in the system. Two heuristic algorithms are proposed to provide for graceful degradation in the presence of disk failure. The heuristic algorithms attempt to maximize the rewards (related to average quality of a stream) while reducing the resolution of clients.

Since SAM uses compressed video, it must be processed in video coders and decoders. The basic operation in the discrete cosine transform (DCT) and its counterpart (IDCT), which are used in MPEG coding and decoding respectively, is matrix multiplication. Two ways of implementing matrix multipliers are using Distributed Arithmetic (DA) structures and systolic arrays. We present multiplier designs using serial and parallel DA look-up table and accumulator structures, on Xilinx FPGAs. In these designs, an \( n \)-bit carry chain, where \( n \) is the word length, is broken into smaller \( r \)-bit chains, \( 1 \leq r < n \). A design space exploration of these designs shows the cost/performance tradeoffs with various carry-chain lengths. A design for a systolic-array based multiplier is also presented. The design procedure is shown in detail by presenting the dependence graph, time and space mappings of the design. We explore \( d-r \) DA structures for DCT on Xilinx FPGAs, where \( d \) is the digit size and \( r \) is the length of the carry chain in the critical path, for \( 1 \leq r \leq n \) where \( n \) is the width of the internal data paths in the design. It is shown that the designs provide a variety of cost/performance trade-offs.
Dedicated to

God, my husband Puneet, mother Deepa and father Lokesh,

and children Rohan and Pranav
Acknowledgements

First and foremost, I express my deepest gratitude to my advisor Prof. Qiang Li for his patience, support and guidance. He allowed me to work at my pace, which enabled me to complete this work successfully. I am grateful to him for the many illuminating discussions on the research presented in this dissertation. This work would not have been possible without his tremendous help and encouragement.

I also express my deepest gratitude to Prof. Weijia Shang for being on my doctoral committee. I thank her for taking out endless hours to discuss new ideas, thoroughly reviewing my papers and offering valuable advice, all of which have helped me complete this work successfully. I also thank her for her consideration, friendship and help.

My sincere thanks to Prof. David Gustavson, Prof. Nam Ling and Prof. George Fegan for being on my doctoral committee. They have provided insights and ideas that have contributed greatly to my research. I am very grateful to Prof. George Fegan for helping me with probability. I thank Prof. Gustavson for his constructive critical comments on my presentation style, and Prof. Ling for his valuable help and advice. Many thanks are due to Prof. Shobha Krishnan for expressing an interest in my work and many useful discussions. I thank Prof. Peter Dommel for his encouragement and help. I also thank Prof. Daniel W. Lewis for providing me support in the form of Teaching Assistantships.

I am deeply grateful to my husband Puneet and my children, Rohan and Pranav, for their support, and allowing me the time to pursue my research, which made this work possible. I am indebted to my husband for helping me in more ways than I can describe here. I also thank my sister Divya and brother-in-law Navneet for the immense help they have accorded me.
My parents have played a critical role with their support and love and I cannot thank them enough. None of this work would have been possible without them.
# TABLE OF CONTENTS

**LIST OF TABLES**...........................................................................................................xi

**LIST OF FIGURES**.....................................................................................................xiii

1. **INTRODUCTION** 1

   1.1 **BACKGROUND** .......................................................................................1
   1.2 **PROBLEM STATEMENT** ..........................................................................5
   1.3 **PROPOSED SOLUTION** ...........................................................................8
   1.4 **LITERATURE REVIEW** ..........................................................................12
       1.4.1 Data Placement Schemes .................................................................13
       1.4.2 Admission Control ............................................................................16
       1.4.3 VCR Operations ...............................................................................17
       1.4.4 Fault tolerance ..................................................................................19
   1.5 **ORGANIZATION OF THE DISSERTATION** .......................................24

2. **ARCHITECTURE AND DATA LAYOUT SCHEME FOR SAM** 26

   2.1 **BACKGROUND** .......................................................................................26
   2.2 **STORAGE AREA NETWORK MULTIMEDIA SERVER (SAM)** ..........27
       2.2.1 Data layout .........................................................................................29
   2.3 **SCHEDULING SCHEME** .........................................................................34
3.7 SUMMARY ........................................................................................................... 64

4. SUPPORT FOR INTERACTIVE OPERATIONS 65

4.1 BACKGROUND ........................................................................................................... 65

4.2 RELATED WORK ...................................................................................................... 66

4.3 PROPOSED APPROACH ........................................................................................... 67

4.4 ANALYTICAL MODEL .............................................................................................. 69

4.5 MODEL VALIDATION ............................................................................................... 79

4.6 SUMMARY ............................................................................................................... 81

5. MATRIX MULTIPLICATION WITH DISTRIBUTED ARITHMETIC 83

5.1 BACKGROUND ......................................................................................................... 83

5.2 TRADITIONAL DALUT ARCHITECTURE .................................................................. 85

5.3 PROPOSED DESIGNS ............................................................................................... 87

5.4 COST AND PERFORMANCE ANALYSIS .................................................................. 93

5.5 IMPLEMENTATION RESULTS .................................................................................. 99

5.6 SUMMARY ............................................................................................................... 101

6. SYSTOLIC-ARRAY BASED MATRIX MULTIPLICATION 103

6.1 BACKGROUND ......................................................................................................... 103

6.2 MATRIX MULTIPLIER DESIGN ............................................................................... 105

6.2.1 Background ........................................................................................................... 105
List of Tables

Number | Page
---|---
Table 1-1. A comparison of different parallel video servers. | 21
Table 2-1. Video traces used in simulation | 37
Table 3-1. Sample characteristics of movies in database | 60
Table 3-2. Values of $N_{det}$ and $N_{stat}$ for the placement schemes. | 61
Table 4-1. Values of $p_{ov}$ and $p_{glitch}$ for different $N$, $kF$, $R$ and $g$. | 77
Table 4-2. Video traces used in simulation | 80
Table 4-3. Variation of $p_{ov}$ with the number of FF/Rew streams on 5 nodes, and no additional buffer | 80
Table 4-4. Variation of $p_{glitch}$ with additional buffer slots for 25 FF/Rew streams on 5 nodes | 81
Table 5-1. CLB counts and logic delays in various components, as a function of word size $n$, on the XC4000 series. | 94
Table 5-2. Delay parameters for XC4000 (-3) series. | 95
Table 5-3. One LUT-based serial DA design for four-product MAC. | 100
Table 5-4. Two LUT-based serial DA design for eight-product MAC. | 100
Table 5-5. Two LUT-based parallel DA design for four-product MAC. | 101
Table 6-1. Computation of PEs in Figure 6-2 at each cycle. | 111
Table 7-1. Values for logic and routing delays and number of slices for dct on

xcv300e-8pq240 ......................................................... 137
List of Figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2-1. A prototype architecture for SAM</td>
<td>28</td>
</tr>
<tr>
<td>Figure 2-2. Client start-up latency with in order and out of order delivery of video blocks</td>
<td>30</td>
</tr>
<tr>
<td>Figure 2-3. Layout for a VBR video file using CRS. Each block is numbered as (chunk #, block #), and the number of blocks in a chunk may be variable</td>
<td>33</td>
</tr>
<tr>
<td>Figure 2-4. Transmission sequence in each sub-round using CRS-SS with four clients served concurrently, using the data layout scheme shown in Figure 2-3</td>
<td>35</td>
</tr>
<tr>
<td>Figure 2-5. Effect of number of clients on mean response time for (a) round time: 0.2 second and (b) round time: 1 second</td>
<td>39</td>
</tr>
<tr>
<td>Figure 2-6. Effect of $d_{limit}$ on mean response time for (a) round time: 0.2 second and (b) round time: 1 second</td>
<td>40</td>
</tr>
<tr>
<td>Figure 2-7. Variation of the mean response time with the buffer size for (a) round time: 0.2 second and (b) round time: 1 second</td>
<td>41</td>
</tr>
<tr>
<td>Figure 2-8. Effect of number of nodes on mean response time for (a) round time: 0.2 second and (b) round time: 1 second</td>
<td>42</td>
</tr>
<tr>
<td>Figure 3-1. Different ways in which a chunk may be placed on a node. (a) Zonal (b) Bundled (c) Partial-Bundled</td>
<td>49</td>
</tr>
</tbody>
</table>
Figure 3-2. Number of clients admitted for different placement schemes and round
durations, with $N_{det}$ and $N_{stat}$ values as shown in Table 3 and with (a) $\alpha=0.5$ (b)$\alpha=0.2$. The average service time (in msec) of the disks in a round for different
placement schemes and with (c) $\alpha=0.5$ and (d) $\alpha=0.2$. ................................. 62

Figure 3-3. Performance comparison of HAL1 and HAL2 algorithms......................... 63

Figure 3-4. Performance comparison of HAL2 and Optimal algorithms...................... 64

Figure 4-1. Different ways in which released slots ($R$) and FF/REW requests ($F$) of 2
streams may be distributed across 2 nodes............................................................ 68

Figure 4-2. Buffer requirement for different combinations of the released slots ($R$)
and FF/REW requests ($F$). ............................................................................. 69

Figure 4-3. Probability of meeting all stream requests in a round with no additional
buffer allocated for FF/Rew requests.............................................................. 74

Figure 4-4. Probability of meeting all FF/Rew stream requests in a round with (a)
$DF$ buffer slots ($D =$ number of nodes) allocated on each node (b) $NF/4$ buffer
slots ($N =$ number of streams) allocated on each node. .............................. 75

Figure 5-1. Traditional LUT-based serial DA for a four-product MAC. ................ 86

Figure 5-2. Proposed ($l$-bit) serial DA one-LUT design for a four-product MAC... 88

Figure 5-3. Proposed ($r$-bit) serial DA one-LUT design for a four-product MAC... 89

Figure 5-4. Traditional serial DA two LUT-based design for an eight-product MAC.
...................................................................................................................... 89

Figure 5-5. Proposed serial DA two-LUT design for an eight product MAC........... 90

Figure 5-6. Traditional 2-bit parallel DA for a four product MAC....................... 91
Figure 5-7. Proposed 2-bit parallel DA for a four product MAC. .................................. 92

Figure 5-8. XC4000 dedicated carry logic............................................................... 95

Figure 5-9. Cost-performance analysis of SDA for a four product MAC with
different word sizes and carry chain lengths  (a) cost vs. word size (b)
performance vs. word size (0 routing delay assumed)................................. 97

Figure 6-1. Modified Baugh Wooley method for multiplying two words with word
length $p = 3$. (b) Bit-level addition of partial product bits for $w_{11} = \sum_{i=1}^{2} x_{r} y_{r}$ with $p$
$= 3$. ............................................................................................................... 108

Figure 6-2. Processor array structure to generate one word $w_{11}$ of product matrix $W$
for $r = 4$ and $p = 3$.................................................................................... 109

Figure 6-3. Index set and dependence structure of a one-dimensional algorithm... 115

Figure 6-4. Bit-level dependence structure for $r = 4, p = 3$. ......................... 116

Figure 6-5. Entire layout of the two’s complement bit-level array for matrix
multiplication with even $r$. ........................................................................... 126

Figure 6-6. Schematic for Cell A and its symbolic notation............................. 127

Figure 6-7. Schematic for cell B and its symbolic notation.............................. 128

Figure 6-8. Schematic for cell C and its symbolic notation.............................. 128

Figure 6-9. Schematic for cell D and its symbolic notation......................... 128

Figure 7-1. Traditional 1-D DA DCT................................................................. 132
Figure 7-2. (a) Design of an 8-bit (or larger) shift accumulator, which shifts one bit at a time (a) using 4-bit components. (b) using standard 5-bit library components.............................................................. 133

Figure 7-3. Design of an 8-bit (or larger) shift accumulator, which shifts two bits at a time, using 5-bit components. ................................................................. 135

Figure 7-4. Design of an 8-bit (or larger) shift accumulator, which shifts three bits at a time, using 5-bit components................................................................. 135

Figure 7-5. r-bit adder used in 2-bit and 3-bit DA DCT........................................ 136
Chapter 1

Introduction

1.1 Background

Interactive systems give users the flexibility of selecting and receiving particular information with the click of a mouse or a TV remote control. The user is no longer a passive recipient of the information, but rather controls it actively. The information must be stored, retrieved and processed, in a timely manner, before it is delivered to the user.

Applications such as video-on-demand, interactive news television, distance learning, e-commerce, interactive video games, and scientific visualization store and retrieve large amounts of real-time data over a network. Video is a sequence of plain images occurring periodically, and audio is a sequence of samples with periodic behavior. These media are called continuous media [1]. Among all types of multimedia data such as images, 3D graphics, video, audio etc., continuous media are the most challenging ones. Continuous media are characterized by stringent real-time constraints, and must be made available by specified deadlines. Failure to do so can result in disruptions and delays, also called jitter, which can compromise the quality of the presentation at the client end. In order to provide good quality of service to the clients, it is necessary to reduce jitter to a very low level.

Video services can be classified into the following categories, based on the amount of interactivity allowed [2]:
• *Broadcast* (No-VOD) services similar to broadcast TV, in which the user is a passive participant and has no control over the session.

• *Pay-per-view* (PPV) services, in which the user signs up and pays for specific programming.

• *Near video-on-demand* (N-VOD) services, in which functions like forward and reverse are simulated by transitions in discrete time intervals (on the order of 5 minutes). This capability can be provided by multiple channels with the same programming skewed in time [3].

• *True video-on-demand* (T-VOD or simply VOD) services, in which the user is provided with total interaction capability. The user has full-function VCR (virtual VCR) capabilities, including forward and reverse play, freeze, and random positioning. T-VOD needs only a single channel for a client; multiple channels become redundant.

In this dissertation, we present the design of a T-VOD system called *Storage Area network Multimedia server* (SAM). SAM manages a set of parallel storage devices connected by a storage area network and is geared towards providing multiple concurrent services such as interactive video on demand and video games. We present the architecture, data layout, admission control schemes and support for interactive operations for SAM. The primary goal of the server is to serve a large heterogeneous client population. SAM can provide *Quality of Service* (QoS) guarantees to its clients by limiting client jitter to within acceptable limits. SAM also continues to provide uninterrupted service in the event of a *node* (storage device attached directly to a network) failure in the system. VCR operations such as fast-forward/rewind are also supported on SAM.
Video and audio have very stringent requirements on real-time processing and enormous storage and transportation requirements. For example, MPEG-1 and MPEG-4 streams typically have bit rates of 1.5Mbits/s and 9-40Kbits/s respectively. The storage requirements are also considerable – a 90-minute movie encoded in 1.5Mbits/s MPEG-1 format with similar image quality as Video CD takes up about 1 GB of storage space.

A single server such as a PC with attached storage is commonly used to deliver videos to clients. However, the capacity of a single server is limited. Therefore this solution is not scalable and restricts the number of clients that can be supported. When the demand exceeds the server’s capacity the data may have to be replicated on another server. This doubles the storage requirement of the system. Another possibility is to partition the videos into disjoint subsets and store each subset on different servers. Studies have shown that the video retrievals are skewed [4], i.e. some videos are more popular than others and also vary with time. This can result in some of the servers not storing the popular movies being under-utilized, whereas a server that holds a popular video may be overloaded.

A better solution is to divide each video into small blocks and place them so that each video is spread over many servers. This results in parallel servers where video data are striped, rather than replicated or partitioned, across multiple servers. Many different designs of parallel servers have been proposed, which will be discussed later, but all of them consist of multiple independent servers connected by an interconnection network. Each server has its own disk storage and network interface. This architecture allows one to scale up the system capacity to more concurrent users by adding more servers and redistributing data across them.

SAM is a parallel server in which the various devices are connected using a storage area network. A storage area network (SAN) is a separate network dedicated to storage and optimized for the movement of data from server to disk and tape [73].
There are several possible network architectures/protocols that could possibly meet the requirements of the SAN. The majority of SANs use Fibre Channel as a transport using serial SCSI protocol to move data to and from disk. Fibre Channel SANs provide high bandwidth (100MBps) and long distances (to 10 km). Another candidate, which is the focus of industry effort, is the upcoming InfiniBand [5] network architecture. InfiniBand is a bottom-up interconnect architecture that provides operating system independent communication over a switched network fabric.

The basic feature of a SAN, as compared to a conventional IP net, is its support for the direct transmission of low level I/O commands (SCSI) over the network media, thus eliminating the need for intermediate IP based servers, TCP stacks, NFS, etc. In most storage systems today data must move through the server machine before it reaches the network. Thus, a store-and forward copy through the server machine is required. Multimedia data possess different caching properties from text data: they are data-intensive and have a ravenous appetite for space and are relevant only for a very small duration from the time of their retrieval. In a SAN, the network-attached storage devices are connected directly to the network and can bypass the server overhead. Server striping is not well supported by any of the existing popular distributed file systems, which limits their scalability. In a SAN the storage devices are shared, and therefore, server striping can be done more effectively. Also, the file access computation and network transfer bandwidth can be coupled to each drive in a SAN. Hence, the aggregate transfer bandwidth scales with drive rather than server memory and network bandwidth.

SAM supports *VBR* (variable bit rate) *multi-resolution* video. For storage efficiency, video must be compressed before writing to the disk. However, the resulting bit rate of compressed video varies in time. Multi-resolution compressed video consists of a compressed full rate video bit stream whose various subsets
correspond to different resolutions or rates of the same video sequence. It also provides several advantages in interconnected heterogeneous networks with various bandwidth characteristics, multimedia applications and video-on-demand services [6]: For example, when high speed fiber optics networks with Gigabit per second capability are connected to lower bandwidth networks such as ISDN and POTS, it is highly desirable to “filter” a compressed video bit stream originating from a source to various destinations with different bandwidth connection capabilities. Another example is in multimedia systems where it is desirable to provide each user with a video bit rate proportional to the window sizes that he chooses to open. In this scenario use of scalable compression video has several advantages. First, the user’s decoder does not necessarily need to decode a full resolution bit stream before choosing the desired bit rate/resolution; second, by providing a subset of the full resolution video bit stream the server can serve a larger number of clients; finally, the need for storing multiple copies of the same movie at different bit rates/resolutions (to implement interactive operations) is eliminated. Another use of scalability is during periods of resource shortage in which the video can undergo graceful degradation, i.e. a partial loss of the bit stream results in only partial loss in quality.

1.2 Problem Statement

SAM is a parallel SAN-based video server storing multi-resolution video. SAM aims to deliver video efficiently to a large number of concurrent clients with the provision of QoS guarantees. Therefore, the design of SAM encompasses several issues:

1. **Inter-node data layout**: In a parallel video server data are striped across the various servers (also called nodes) instead of being stored on a single server. The dissertation looks at the problem of striping the data so the system start-up latency can be lowered. The start-up latency is the smallest time duration for
which a client must wait between initiating a new request and starting display, in order to prevent jitter. It is important to keep the start-up latency small to efficiently support interactive operations such as fast-forward and rewind. Various factors decide the start-up latency – the length of time the request is queued in a node before it is processed, node service time, and the time spent in the node buffer until the retrieved data can be transferred on the interconnect. The striping scheme must also provide scalability, reliability and a good system throughput.

2. **Intra-node data layout**: The intra-node data layout scheme decides how the files should be stored on a particular node. This scheme has important implications on the provision of QoS guarantees, such as bounded jitter, for clients that are being serviced. When dealing with multi-resolution files many questions are raised – should the different resolution layers in a stripe unit of a file be placed contiguously on a node or should they be distributed across the node? Intuitively, the former must provide a higher throughput because the seek overhead in a node can be reduced with contiguous placement; on the other hand, the latter scheme may provide other advantages, such as better QoS. This dissertation takes a close look at this problem.

3. **Admission Control**: How should the server decide whether or not to admit a new client? A fundamental problem in developing a VOD system is one of storage and network I/O bandwidth management. A VOD server has a finite amount of resources, and when a new client arrives it must decide whether or not to admit the client. The service provider desires to generate the maximum revenue from the services by admitting the largest number of clients possible. At the same time, it must meet the QoS parameters of the clients that are admitted by reserving sufficient resources for them. The server must balance these two
conflicting goals with the provision of an admission control scheme so that resources are not over-utilized or under-utilized.

4. **Node Failure**: How should the server handle the situation when a node failure occurs so that video delivery is uninterrupted? Two basic schemes that are used are parity-based and mirroring based. But these schemes can increase the load on the remaining nodes, which may adversely affect the QoS provided to the clients during admission. Therefore, a scheme is required that will provide graceful degradation, so that there is only partial loss in bit stream quality, and QoS guarantees are preserved.

5. **Interactive Operations**: Another question is how the server can support features such as rewind, pause and fast-forward. Operations such as fast-forward and rewind typically consume large amounts of bandwidth, which can make resources unavailable to other clients. The main problem in handling fast-forward/rewind is that the amount of resources required is not precisely predictable, and depends on the interactions that a user may choose to take.

6. **Support heterogeneous clients**: How can service be provided to heterogeneous clients? For example, there is a growing demand for enabling multimedia applications on mobile devices. We are finding a great diversity of these devices such as personal digital assistants (PDAs), hand-held computers (HHC), smart phones and automotive computing devices. These devices are typically constrained along a number of dimensions, such as processing power, local storage, display size, connectivity and communication bandwidth. These limitations make it difficult to enable image and video access by these devices.

7. **Cost-Performance Tradeoffs in design of video coders and decoders**: Different applications have different performance and area requirements. The question is how can one vary the length of carry chains to meet specified requirements.
Specifically, what are the cost-performance tradeoffs in designs for matrix multiplication and DCT using variable length carry chains in the critical path?

1.3 Proposed Solution

Storage Area network Multimedia server (SAM) is a parallel VOD server that can provide multi-resolution video with different levels of quality to its clients. SAM consists of IBM Ultrastar 18ES storage devices connected on a storage area network. SAM provides small response time, bounded jitter, high throughput and can be scaled to service thousands of heterogeneous clients. Some of the other features of SAM include provision of interactive operations such as fast-forward and rewind, and graceful degradation in the presence of node failure.

1. **Inter-node data layout**: A new inter-node data layout scheme called constrained random striping (CRS) is proposed for SAM [7]. The main feature of this scheme is that it uses a novel placement of the stripe units across nodes, which reduces the start-up latency of a client. A logical loop model is considered, which consists of nodes connected on a shared interconnect that transmit bursts of data in a staggered manner. The primary idea behind CRS is to partition the data into fragments, and place the fragments so that those needed earlier are placed on higher priority nodes that can transmit earlier in a round. Using a simulation model, the performance of this scheme is compared with several other data layout strategies. The simulation results show that the CRS scheme provides the smallest start-up latency for newly arrived clients compared to other schemes.

2. **Intra-node data placement**: A new intra-node placement scheme called Partial-Bundled is developed for VBR multi-resolution video [8]. In this scheme, the minimum resolutions layers of the multi-resolution video are bundled together and placed contiguously on one portion of the node. The advantage of this
scheme is that it can provide higher throughput than other existing schemes when the QoS parameter is jitter. Simulation results show that this scheme can support from 10% to 100% more clients than other schemes depending on system parameters such as round time.

3. **Admission Control**: A new admission control scheme is also presented which can provide QoS guarantees so that the client jitter is bounded [8]. The basic idea here is that minimum or necessary layers must not be dropped as that can result in unacceptably large glitch in the client video stream, while some loss in the higher layers can be tolerated. E.g., in MPEG video coding, frames are coded in one of three modes: intraframe (I), predictive (P) or bidirectionally-predictive (B). These modes provide intrinsic layering in that an I-frame can be independently decoded, while P-frames require I-frames, and B-frames generally require I- and P-frames to decode. In each temporally scalable MPEG trace the following layering scheme is used: I- and P-frames make the base layer, and B-frames make the enhancement layer. In an MPEG-encoded video stream, discarding an I-frame eliminates all the succeeding P- and B-frames until the next I-frame is accessed. If an I-frame is lost, a perfect picture cannot be displayed again until the next I-frame is received, which is typically every 15 frames or half-second. A half-second loss of video is noticeable and unacceptable. For MPEG, if an I-frame is lost, the QOS may be inadequate. Therefore, in the proposed scheme, deterministic guarantees are provided for minimum layers of the multi-resolution video, and statistical guarantees are provided for higher layers of multi-resolution video. An analytic node model is used to determine the amount of node resources that can be reserved.

4. **Node Failure**: When a node failure occurs, the streams executing on the failed node must be supported on the remaining node(s) (depending on the replication scheme used for recovery). In the proposed solution, on each node, a certain
fraction of the buffer slots are reserved in order to accommodate the minimum resolution layers of streams on the failed node. In normal operation, however, these reserved slots may be used by the higher resolution layers, in order to make more efficient usage of resources. After node failure, the resolution of the higher resolution layers of the admitted streams is degraded, such that the QoS guarantees are upheld. The optimal solution to the resolution degradation problem is shown to be NP-complete. Two heuristic algorithms [8] are proposed to provide for graceful degradation in the presence of disk failure. The heuristic algorithms attempt to maximize the rewards (related to average quality of a stream) while reducing the resolution of clients.

5. Interactive Operations: The server also provides statistical QoS guarantees for interactive operations such as fast-forward, rewind and pause. We show that the statistical multiplexing of available resources can reduce the resources required to support interactive service on a parallel server. An analytical model is developed to predict the probability with which stream requirements can be met for a given node buffer level. A fast-forward/rewind (FF/Rew) stream on a node can use the resources released by other FF/Rew streams on the same node, if any are available, without requiring any additional buffer. It is proved that the probability of satisfying all interactive stream requests in a round, by sharing resources between streams and using no additional buffer, lies in the interval (0.5, 1). When a small amount of additional buffer is allocated for the FF/Rew streams, the probability of satisfying interactive requests increases depending on the amount allocated.

6. Support heterogeneous clients: SAM supports multi-resolution video and provides different levels of service with QoS guarantees – this is an important feature in SAM, which enables it to service a heterogeneous client population.
An overview of the motivation and design principles behind SAM is presented in [9].

7. **Cost-Performance Tradeoffs in design of video coders and decoders**: In SAM compressed video is used, and a second part of this dissertation looks at the design of video coders and decoders. The basic operation in the discrete cosine transform (DCT) and its counterpart (IDCT) is matrix multiplication. The cost-performance tradeoffs of matrix multiplication with variable length carry chains are discussed for different designs.

Distributed Arithmetic (DA) is an important technique to implement digital signal processing (DSP) functions in FPGAs (field-programmable gate arrays). Multiplier designs are implemented for serial and parallel DALUT and accumulator structures in which an $n$-bit carry chain, where $n$ is the word length, is broken into smaller $r$-bit chains, $1 \leq r < n$. The implementation is on a Xilinx chip XC4028XL-3-BG256 using Xilinx Foundation tools v 3.1i. The results show that the proposed designs can achieve speedup by a factor of at least 1.5 over traditional DA designs in some cases [10]. These designs are used in the implementation of DA-based DCT. New designs are presented for $d$-$r$ DA structures for DCT on Xilinx FPGAs, where $d$ is the digit size and $r$ is the length of the carry chain in the critical path, for $1 \leq r \leq n$ where $n$ is the width of the internal data paths in the design. For example, 1-16 represents bit-serial distributed arithmetic designs for DCT, with a 16-bit carry chain in each shift accumulator. Similarly, a 3-8 design implies a 3-bit serial distributed arithmetic DCT design, with two 8-bit carry chains in each 16-bit shift accumulator and adder. The designs are implemented on a Virtex-E chip (xcv300e-8pq240). The area, latency, and throughput are determined for various designs with different $d$ and $r$. 

Matrix multiplication can also be implemented on systolic arrays. A systolic array based matrix multiplier on a Xilinx FPGA chip is described in [11, 12] where each processing element does a simple operation of adding three to six bits to generate one partial sum bit and one to two carryout bits. The bit-level matrix multiplier was implemented on a Xilinx FPGA chip and compared to a word-level matrix multiplier composed of highly optimized multiplier and adder macros available in the Xilinx Core generator library. The results show that speedup by a factor of 2 can be obtained in practice. The design procedure is shown in detail by presenting the dependence graph, time and space mappings of the design in [13]. In addition, the time optimality and conflict-free properties of the design are also proven.

1.4 Literature Review

Video servers can organize the storage of continuous media objects on disk in terms of media blocks whose playback durations are equal called rounds. Each admitted client is serviced once in every round, and a fixed number of media units (e.g. video frames or audio samples) are retrieved for each client.

Many designs for VOD servers have been proposed. They can be grouped under one of the three main architectures: single, parallel and distributed server. The single server model comprises small-scale systems made from a standard PC up to massively parallel supercomputers for large-scale systems. Examples of the single server model consist of [21,22,81,14] and others.

In parallel servers video data are striped, rather than replicated or partitioned, across multiple servers [15]. VOD servers that are based on clusters of workstations fall under this category [16,17,20]. Another example of a parallel server model is a storage-area-network (SAN)-based VOD server [54]. Cluster-based servers are composed of whole computers, each with its own processor, memory, OS,
applications etc, whereas SAN-based servers are generally composed of network-attached storage devices or nodes (NAS).

In the distributed server architecture multiple local servers are placed close to user pools, and according to their local demands, dynamically cache the contents streamed from the video repository.

1.4.1 Data Placement Schemes

The data placement scheme is one of the most important factors that affect the performance of a system. Different video servers employ many different policies to break a video file into small pieces and distribute those pieces across the storage nodes. Before discussing the data layout policies, some terms are presented.

**Chunk**: A video file is broken up into chunks $F_i$ such that the video file $F = \{F_1 \cup F_2 \cup \ldots F_n\}$, $F_i \cap F_j = \emptyset$ and $i, j \in 1, 2, \ldots n$. A chunk is the data retrieved for each stream during a round and it contains a fixed number of media units. A chunk has constant time length (CTL) in terms of playback duration, but the size of a chunk can vary from round to round if each video is compressed using a VBR compression algorithm.

**Block**: Each chunk is broken into fixed-size blocks (or fragments) where a block is the maximum amount of logically contiguous data stored on a single device. A chunk is composed of varying number of blocks for VBR videos and an equal number of blocks for CBR videos. A block has constant data length (CDL). The advantage of a block is that it simplifies storage and buffer management because all blocks have the same size. The trade-off is that it is more difficult to schedule the transmission of blocks since they have varying deadlines.

The chunks/blocks in a video file are interleaved across nodes; node interleaving is also called *striping*. The number of disks across which the data is
striped is the degree of interleaving, or the *striping width*. The degree of interleaving can be further classified as *single*, *narrow* and *wide* [19].

If the chunk/block is confined to a single device, the degree of interleaving is termed single. The Tiger video server [16] uses single striping. A distributed scheduling strategy is used in which data retrieval for a particular stream cycles through all disks, in successive cycles, in a round robin manner. All streams move in lockstep across the disks, i.e. all streams served on a given cycle are transferred to the next logical disk in the following cycle. MARS [17] is an example of a parallel storage server that uses single striping. It consists of storage nodes on a switched ATM interconnect. A central manager, in MARS, implements a token-based distributed scheduling scheme between unsynchronized storage nodes. A token is sent out periodically on the ATM interconnect, and this determines when a particular storage node can transmit. RIO [55] is a general-purpose server that uses random single striping. They show that RIO has a cost/performance competitive to that of conventional round robin striping techniques, with the advantage of supporting much more general workloads. They also show that RIO outperforms round robin striping if some replication is used, even if replication is only partial. SPIFFI [18] consists of 64 nodes/ 64 disks on an Intel Paragon. The Intel Paragon is a supercomputer that is capable of scaling to thousands of nodes. The nodes are connected by a high performance mesh network, which enables processes at distinct nodes to exchange messages and data efficiently. SPIFFI uses single round robin striping.

If the chunk/block encompasses only a fraction of the devices in the system, the degree of interleaving is termed narrow. In [19] it was shown that wide and narrow striping methods have a lower *response time* than single striping methods, where response time is the *maximum* elapsed time between the arrival of a new client and the transmission of the first block for this client. It is important to keep the
response time small to efficiently support interactive operations such as fast-forward and rewind. However, wide striping is not popular in distributed servers because of high buffer requirements. Here, the amount of buffer is proportional to the number of disks (assuming a fixed block size). Since the number of streams is also proportional to the number of disks, the amount of required buffer grows quadratically with the number of supported streams; a problem which is usually referred to as the buffer explosion problem. Mitra [20] is a parallel VOD server that is geared towards very high bandwidth multimedia data. The data layout uses a narrow data layout scheme called staggered striping. In staggered striping, streams can access multiple blocks on consecutive disks in each cycle. As in traditional striping, the set of accessed disks is shifted in consecutive cycles. By allowing streams to access a different number of blocks per cycle, multiple data rates can be supported, but still each stream is assumed to have constant bit rate.

If the chunk/block is placed across all the devices in the system, the degree of interleaving is termed wide. Examples of servers that use wide striping are [53,21,22]. In [53] the Server Array is described which is a parallel server that uses sub-frame striping i.e. one frame is split evenly across all the storage nodes. This offers perfect load balancing for VBR videos but a high disk overhead for retrieval.

Another classification depends on the order in which the chunks/blocks are placed: round robin and random. In round robin striping the successive chunks/blocks are placed on successive devices, and after the last device is reached the placement is wrapped around to start from the first device. Examples of video servers in which round robin striping is used are [16,17,20,24,53,56]. In random striping the successive chunks/blocks are placed on randomly selected devices e.g. [55,54].

In addition, each movie may be placed across all the disks (Wide) in the system or on a subset of the disks (Narrow). Parallel servers generally use wide
striping. E.g., the Tiger Shark Video server [56] is a high performance video server in which wide striping is employed, and each video file can be striped evenly on more than 65000 disks.

1.4.2 Admission Control

An admission control scheme ensures that the resources are not over-utilized by allowing too many streams, beyond the system capacity, into the server. A system overload can result in a poor QoS to clients, for example, by increasing jitter. On the other hand, if the admission control scheme is too restrictive, the number of streams admitted to the system will be small, and the system will be under-utilized. Therefore, it is important to provide good admission control so that QoS guarantees can be provided, and the highest possible system throughput can be obtained. Three schemes for admission control are deterministic, predictive and statistical. A deterministic service policy [23] guarantees specified QoS requirements of existing customers and admits a new client only if its service demand does not affect the present clients. In this case, the worst-case assumptions about stream resource requirements must be made when admitting the stream, so it is highly restrictive. In [24] a deterministic admission control scheme is described which guarantees normal playout and interactive operations separately. When a normal playout changes to an interactive operation it is treated as a new request.

A predictive admission control policy [25, 26] monitors the server utilization over a time window, and admits a client if the recent history indicates that the server can meet the requirements of the client.

With statistical admission control, deadlines are guaranteed to be satisfied with a certain probability, for example, guarantee that 95% of deadlines will be met during a round. In [58], a statistical admission control scheme is described in which a new client is admitted only if sum of the average rates of admitted streams is below server capacity. In DAVS [62], an adaptive admission control scheme is
described for multi-resolution video. In this scheme, each connection request generates two layer vectors – minimum quality $R_{min}$ and maximum quality $R_{max}$. The scheme admits as a function of $R_{min}$ and $R_{max}$ i.e. $f(R_{min}, R_{max})$, where $f()$ is chosen on promotion/demotion heuristics. If all streams are admitted at $R_{min}$, the number of streams that can be admitted to the system is a maximum but the quality is low, whereas if all streams are admitted at $R_{max}$ the quality is high, but the number of connections is limited at low loads. The adaptive scheme provides a tradeoff between these two extremes by maximizing the number of connections admitted while still maximizing the average marginal quality of those connections. However, they do not conduct any experiments to check how the fluctuating video quality affects user perception of a particular video. In [27,28], statistical guarantees are provided to clients retrieving VBR scalable video. In [28], the admission control ensures that the total number of frames discarded by the multimedia server during overflow rounds will not exceed the cumulative loss tolerance of all the clients. In order to ensure that the individual service requirements of the clients are not violated, the server distributes the discarded frames among all the clients with the goal of minimizing the cumulative frame loss. In [83], the authors propose a scheme to deal with disk overload by dropping appropriate frames in the MPEG video. They propose a number of classes of service, corresponding to various probabilities of loss. Requests with a higher probability of loss threshold are degraded before those with lower thresholds.

1.4.3 VCR Operations

A VOD server must provide interactive operations such as fast-forward, rewind, pause etc. in order to efficiently replace the VCR. To illustrate, once a video conference is archived, a participant may want to browse a stored clip for a specific discussion or demonstration. This may be achieved by a FF/FR to the point of interest. Designs that implement FF/FR functions must be scalable, i.e. the number of displays doing FF/FR increase linearly with increasing bandwidth.
There are two basic paradigms for implementing VCR functions: *Online* and *Offline* processing [36]. In *online processing*, a single normal-speed movie is processed accordingly in real-time during playback. In *offline processing*, a clip is pre-processed by the content provider with separate files for FF/FR operations. The tradeoff between these two approaches is I/O bandwidth vs. storage space. Online techniques require more bandwidth (disk, network), whereas offline methods need more storage space.

An online processing technique is sub-sampling at the client side in which separate server bandwidth is reserved for VCR interactions, using statistical methods [29]. This approach allocates $n$ times the playback bandwidth for $n$ times FF/FR viewing, using statistical QoS guarantees. When bandwidth is scarce due to a high system load, service is either delayed or provided by sacrificing resolution.

Other online processing techniques use sub-sampling at the server side [30,31] and segment skipping [32,33]. Sub-sampling at the server side is conceptually similar to sub-sampling at the client side, except that the real-time frame reduction is done at the server side. This approach does not increase the network bandwidth. Segment skipping skips a fixed number of blocks to achieve the desired playback rate by controlling the placement of blocks on disk drives. For example, to provide three times fast forward, this method displays one block out of three consecutive blocks. Segment skipping also has good scalability since it does not require extra network or disk bandwidth to support various fast rates. However, since the granularity of skipping is at the block level and not the frame level, it results in unusual previewing. A similar scheme for grouping MPEG frames into segments, wherein no frames are discarded during fast playback with segment skipping due to unavailability of other frames needed for their decoding, is given in [34].
In order to provide FF/FR functionality, offline processing techniques require separate, pre-processed versions of the normal file [35, 36]. For example, to create a three times fast-forward file, every third frame is selected from the original movie before compression. This collection of frames is encoded in the regular manner (e.g., using MPEG) and stored in a separate file. Cross-references among the different files are maintained so that a user can switch between versions. This method does not require extra network or disk bandwidth, but needs extra storage space to store the additional versions.

Other approaches use inspection of past system behavior for resource allocation [37,38, 39]. An analytical model for the provision of VCR functionality in which different data sharing techniques (such as batching, buffering adaptive piggybacking) are combined is described in [40]. Another way to provide interactive service is to use readmission at interaction points i.e. for each user interaction, a new admission request is initiated at the server [41]. In [42], two schemes to restart playback after an interactive operation with a low start-up latency are described. FF/Rew operations that are limited to data present within the client buffer for client buffer size of 25MB or more is described in [43]. A similar method of using caching in the client buffer is used in [44]. An interactive version of stream tapping is presented in [45]. In [46], clients selectively prefetch segments from multicast channels based on the observation of the play point in their local buffer. Other similar schemes are described in [47, 48].

1.4.4 Fault tolerance

The server should be able to sustain continuous video playback even if one or more of its disks fail. Therefore, the server needs to store redundant data that can be used to deliver video reliably even if some of its disks become inoperable. There are two main models for fault tolerance: 1) replication-based and 2) parity-based.
Replication-based methods consist of replicating the data on all or a subset of the disks so that the data from the failed disk can be recovered from the others.

Two replication-based schemes are mirroring [87] and chained declustering [88]. Mirroring consists of replicating each video object. Thus the storage space needed for a mirrored object doubles. However mirroring avoids the increase of the I/O bandwidth when failure occurs that may be observed in parity-based schemes [49]. The replica of each disk can be uniformly distributed over all the remaining disks of the server. This method is called the doubly striped approach and was proposed in [50]. Doubly striping does not tolerate more than a single disk failure. To tolerate multiple disk failures a different approach was proposed in which the secondary copy is distributed over a fraction of the remaining disks [16]. Chained declustering is another replication-based scheme.

Another approach to maintaining reliability is by using parity-based reliability schemes. In these schemes the parity data is stored in addition to the existing original video data. Some RAID systems use this approach to protect against disk failures. When a single disk failure occurs the parity information can be used to reconstruct the missing data [51]. In RAIS, constant data length striping algorithms similar to RAID are employed to achieve server-level fault tolerance [52]. A similar approach is adopted in PRESTO [54].

A different approach is taken in Server Array [53]. Here sub-frame striping (portions of a frame distributed across disks) may be used. This lends itself to the application of error concealment schemes that interpolate missing data from neighboring parts in the image.

In Table 1-1 the important features of various parallel servers are summarized. The data placement scheme is described by the tuple <stripe unit type, video placement across disks, striping width>. Thus, the stripe unit type can be CTL
or CDL; the video placement describes how the video is placed across all the disks – e.g. if a single title is distributed across all the disks it is called Wide, and finally how a block/chunk is placed across the nodes (single, narrow or wide). The default placement scheme is round robin, unless mentioned otherwise. The servers are also classified on the type, purpose, platform, kind of admission control scheme used, and whether VCR operations are supported. The field is left empty when no information is provided, or the scheme has not been implemented.

Table 1-1. A comparison of different parallel video servers.

<table>
<thead>
<tr>
<th>Server</th>
<th>Type</th>
<th>Purpose</th>
<th>Components/Platform</th>
<th>Data placement</th>
<th>Admission Control</th>
<th>VCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
<td>SAN</td>
<td>General</td>
<td>Storage nodes on a SAN</td>
<td>CDL, Wide, Narrow, (CRS)</td>
<td>Deterministic and Statistical</td>
<td>Yes</td>
</tr>
<tr>
<td>TIGER [16]</td>
<td>Cluster</td>
<td>VOD</td>
<td>Pentium PCs on an ATM network</td>
<td>CDL, Wide, Single</td>
<td>Deterministic</td>
<td>-</td>
</tr>
<tr>
<td>MITRA [20]</td>
<td>Cluster</td>
<td>VOD</td>
<td>HP9000 workstations connected with an ATM switch</td>
<td>CDL, Wide, Narrow</td>
<td>Deterministic</td>
<td>Yes</td>
</tr>
<tr>
<td>PRESTO [54]</td>
<td>SAN</td>
<td>General</td>
<td>Storage nodes connected via routers in butterfly network</td>
<td>CDL, Wide, Narrow (Random)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RIO [55]</td>
<td>Cluster</td>
<td>General</td>
<td>Low cost PCs</td>
<td>CDL, Wide, Single (Random)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TIGER SHARK [56]</td>
<td>Cluster</td>
<td>General</td>
<td>RS/6000s that access disks via switched network</td>
<td>CDL, Wide, Single</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SERVER ARRAY [53]</td>
<td>Cluster</td>
<td>VOD</td>
<td>Storage nodes on a switched network</td>
<td>CTL, Wide, Wide</td>
<td>Deterministic</td>
<td>-</td>
</tr>
<tr>
<td>MARS [17]</td>
<td>Cluster</td>
<td>VOD</td>
<td>Pentium PCs on an ATM network</td>
<td>CTL, Wide, Single</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Tewari [57]</td>
<td>Cluster</td>
<td>VOD</td>
<td>RS/6000 machines on a CDL network</td>
<td>CDL, Wide, Single</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Richard H. Veith provides an interesting historical context for the shift from dedicated video-on-demand systems to Internet-based media-on-demand systems in [64]. There are some commercial VOD and N-VOD systems being deployed today. The infrastructure needed for the delivery of VOD to consumers is available. There are a number of high speed data offerings such as Cable (6-30 Mbps downstream, 320 Kbps to 10 Mbps upstream), DBS for dish, and DSL (1.54-28 Mbps). The company Time Warner Cable has selected nCUBE to launch VOD in select cities in the U.S. recently. The service offers digital cable customers movies and other programming content when they want it with complete ability to pause, rewind, fast-forward, and play using their cable television remote control. Some of the other companies that provide VOD are Concurrent Computer Corporation and AT&T. Microsoft offers VOD through its Windows Media Technologies IP-based delivery systems. Outside the U.S., many companies are offering streaming solutions. These include U.K.-based telecommunications companies such as NTL and Telewest, Israel-based EMBLAZE systems Inc., and others. However, the internal details of most of the commercial offerings are not known.
A second part of the dissertation deals with the efficient processing of video in video coders. The discrete cosine transform and its inverse are commonly used in video compression and decompression [65]. Since Field Programmable Gate Arrays (FPGAs) provide reconfigurable hardware and flexible interconnect, they are commonly used for the prototyping of DCT. However, applications like real-time video compression are a challenge to implement on FPGA because of their large computational complexity and large data throughput requirement.

In [66], a 2D DCT is implemented on a Xilinx XC6264 FPGA. They show that a high quality circuit implementation is possible by using distributed arithmetic and exploiting parallelism and pipelining. A throughput rate of $1.536 \times 10^7$ pixels per second is obtained which corresponds to processing colour images of VGA resolution (640 x 480 pixels) at 25 frames per second, with the utilization of 30% resources of the FPGA. Naviner et al. describe an implementation of the DCT on the Altera Flex10k FPGA in [67]. The main features of their implementation include using a special encoding, resource sharing for multiplications, additions and accumulations of partial inner products, and usage of distributed arithmetic. Their implementation uses 2040 logic elements and runs at 27 MHz. In [68], DCT is implemented on a Xilinx 4036XL-09 FPGA. Using 16-bit coefficients a 1-D transform consumed 334 CLBs and, according to the post-place & route timing analysis tool, operated at 101 MHz. With 8-bit coefficients the implementation occupied 196 CLBs and operated at 110 MHz. Three pipeline stages were added to prevent the critical path from propagating through each functional unit. The performance trade-offs among a class of three different 8-point 1-D DCT architectures are presented in [69]. The architectures chosen for implementation are the Distributed Arithmetic (DA) architecture with digit sizes (1 and 3), digit-serial flow graph and systolic architecture. In the DA technique, instead of calculating the inner product with multipliers, the multiplication is performed using memories (ROMs) and shift accumulators [70,71]. Their experiments show that the DA
architecture is the best in terms of area, speed and latency. Design exploration for pipelined IDCT is reported in [72]. Since efficient pipelining is important to obtain good performance in the decoder, designs are pipelined in different ways to obtain a variety of cost/performance trade-offs.

1.5 Organization of the Dissertation

The dissertation is organized as follows:

In Chapter 2, the architecture and data layout schemes for SAM are presented. An *inter-node* data layout scheme called constrained random striping (CRS) is described. Using a simulation model, the performance of this scheme is compared with several other data layout strategies.

In Chapter 3, an admission control scheme that provides QoS guarantees is presented. An *intra-node* data placement scheme called Partial-Bundled is proposed and compared with other data placement strategies. Two heuristic algorithms to provide for graceful degradation in the presence of disk failure are described.

In Chapter 4, a new scheme to support fast-forward and rewind operations is described. An analytical model for statistical resource sharing is presented.

In Chapter 5, new designs are provided for distributed arithmetic based matrix multipliers with variable length carry chains. Designs are described for serial and parallel DALUT and accumulator structures in which an $n$-bit carry chain, where $n$ is the word length, is broken into smaller $r$-bit chains, $1 \leq r < n$. A cost-performance analysis of the designs is presented.

In Chapter 6, a novel design of a systolic array based matrix multiplier in which individual bits of a word do not have to be processed as a unit is presented.
The design procedure is shown in detail by presenting the dependence graph, time and space mappings of our design.

In Chapter 7, new designs for $d$-$r$ DA structures for DCT on Xilinx FPGAs are described, where $d$ is the digit size and $r$ is the length of the carry chain in the critical path, for $1 \leq r \leq n$ where $n$ is the width of the internal data paths in the design.

In Chapter 8, the conclusion is presented.
Chapter 2

Architecture and Data Layout Scheme For SAM

2.1 Background

In this chapter, we propose the architecture and data layout scheme for SAM, a SAN-based multimedia server. SAM consists of high bandwidth storage devices or nodes on a storage area network. A new data layout scheme called CRS is proposed for SAM that can provide smaller start-up latency for nodes on a logical loop.

In most parallel servers [19,20,55,57,61,60,59] the inherent architecture is one in which the nodes can transmit concurrently to the clients. A second model, which we call the logical loop model, consists of nodes that are connected using a shared interconnect and transmit bursts of data in a staggered manner e.g. MARS [17]. In this system, each node prefetches data into its buffer during the time that one of the other nodes is transmitting on the shared interconnect. For the transmitting node, the transfer from the buffer to the interconnect takes place in a single burst.

In this chapter, a new data striping scheme called constrained random striping (CRS) is proposed, which provides a lower start-up latency than other placement schemes for the logical loop model. In interactive servers, it is very important to have small start-up latency, which is the smallest time duration for which a client must wait between initiating a new request and starting display, in
order to prevent jitter. The response time is defined to be the time interval from the arrival of the new client request until the time when the first piece of requested data is transmitted by the node. The response time includes the buffering delay at the node that services the request. The basic idea behind CRS is to fragment the data that must be transmitted in a round, and place it on different nodes, so that portions of the data that are needed earlier are placed on nodes that transmit earlier. It will be shown that this in-order delivery scheme can reduce start-up latency. Consider a request that has come in at the start of a round, and is serviced at the node that is the last to transmit in a round. Then, the response time of this request is larger than one that could be satisfied by nodes that transmit their data earlier in the round. This scheme is different from that used in MARS, where the single striping data layout scheme is used, i.e., all the data that is transmitted in a round is placed on a single node. In the next section, the architecture and data placement scheme in SAM are discussed in detail.

2.2 Storage Area Network Multimedia Server (SAM)

Figure 2-1 shows the proposed architecture for SAM. It consists of several storage nodes and a controller on a network segment. A network segment, or segment, consists of an interconnect whose bandwidth is shared amongst the nodes that are on the segment. An example of a network segment is a fibre channel arbitrated loop (FC-AL). Another example is a link connecting an InfiniBand switch to a gateway. A node is a storage device such as a single disk, RAID array, JBOD (just a bunch of disks) or any other. Adding new nodes on each port of a switch increases the aggregate data retrieval bandwidth, whereas adding new nodes to a network segment further divides the shared network bandwidth.
The controller is responsible for the data layout and initiating data transfer from the nodes. The data retrieved from the nodes is sent through switch 1 to the clients. Switch 2 facilitates communication between controllers regarding scheduling decisions. When a new client request arrives, each host uses admission control to determine if the request can be feasibly scheduled with those being serviced. The hosts communicate the results to each other via switch 2 and determine when to service the new client. The hosts can also access robotic tape systems via switch 2. It is assumed that the network between the server and the clients has sufficient bandwidth.
2.2.1 Data layout

A traditional video-on-demand server can concurrently transmit multiple, distinct VBR streams to clients across a high-speed network. For each stream, data is retrieved from the disk subsystem according to a retrieval schedule and stored temporarily in server memory. The server network processing moves the data from the in-memory buffers to the client, according to some server transmission schedule. The transmission schedule can be very different from the retrieval schedule. For example, data may be retrieved from disks in blocks according to some optimized disk retrieval strategy. However, a simple transmission schedule may transmit this data periodically, in varying amounts, to the client at, say, its frame rate.

In a SAN, there is no centralized intermediate buffer to store the data before it is sent over the network to the clients. This can introduce the problem of out-of-order delivery of video frames, where, a frame with a later playback time is received before a frame with an earlier playback time. The client display will experience a glitch or jitter if a frame is not received in time to be played out. Consider a general model of a multimedia server that interleaves videos across disks by storing successive blocks of a video on consecutive disks in a round robin manner. The server services multiple clients in periodic rounds, and retrieves a fixed number of media units (e.g. video frames or audio samples) for each client. A video file is broken up into chunks \( F_i \) such that the video file \( F = \{ F_1 \cup F_2 \cup \ldots \cup F_n \} \), \( F_i \cap F_j = \emptyset \) and \( i, j \in 1, 2, \ldots, n \). A chunk is the data retrieved for each stream during a round and it contains a fixed number of media units. A chunk has constant time length in terms of playback duration, but the size of a chunk can vary from round to round if each video is compressed using a VBR compression algorithm. Each chunk is broken into fixed-size blocks, where a block is the maximum amount of logically contiguous data stored on a single node. A chunk is composed of varying number of blocks for VBR videos.
Figure 2-2 shows that in-order delivery provides lower start-up latency at the client side. With narrow striping, the blocks in a chunk are placed on different nodes, and are retrieved into their respective node buffers by the start of the transmission round. Blocks that are transmitted in the order of their temporal requirements can take advantage of the overlap between the network delay (includes the transmission, propagation and queueing delays) and the round time, and provide lower start-up latencies.

If ordering is not imposed, the first block to be displayed may possibly be transmitted only at the end of the round, leading to higher start-up latencies. Another advantage of in order delivery is that it can provide service transparent to server characteristics, such as round size. Otherwise, a client would need to buffer a round’s worth of data, so that it could be reordered before display. This is important
in devices such as PDAs with limited buffer sizes of 3-4 MB or less, much of which may be occupied with other software applications. For example, with a 1 second round duration and using double buffering at the client side, an MPEG-2 movie with a 10Mbps bit rate would require about 2.5 MB of buffer.

A data layout scheme in SAM called constrained random striping is presented next. This scheme provides a solution to the out-of-order delivery problem from devices on a logical loop, such that the frames in each chunk can be delivered to the client in a timely and ordered manner.

2.2.1.1 Constrained random striping

The constrained random striping (CRS) scheme is a data layout scheme for VBR video in which the blocks are distributed in a random manner across disks, subject to an ordering and placement constraint. In this scheme, each node on the network segment is assigned a priority. The blocks in every chunk are distributed across the nodes in accordance with the following ordering constraint: a block with an earlier playback time is placed on a higher priority node. In addition, there is a placement constraint for jitter-free delivery of video: each subsequent block must be placed so that it is transmitted within the display time of the previous block. The reason for including a placement constraint can be explained as follows. In VBR video, constant sized blocks have a variable display time, since there are a variable number of frames in each block. If the client has finished playing one block and the succeeding block does not arrive there is jitter at the client display.

The layout scheme is explained with the following example: a chunk with blocks \( b_0, b_1, b_2 \ldots b_n \) (listed in the order in which they have to be played back) is to be distributed across nodes \( d_0, d_1 \ldots d_m \) (listed in order of decreasing priorities). Suppose that the time period is \( T \) and each node transmits once on the segment during this time. Then each node can transmit for a time \( T_d \leq T/n \). Further, suppose that each
block $b_i$ has a display time of $r_iT_d$, where $r_i$ is an integer. Then block $b_0$ can be placed on a node $d_j$ randomly selected from $d_0$ to $d_{n-1}$, block $b_1$ can be placed on any node from $d_0$ to $d_{(r_1+r_2)-1}$ and so on. The start-up latency can be restricted by imposing an additional restriction on the first block in a chunk that it must be placed within $d_0$ to $d_{\text{limit}}$ nodes. A request for a new client must wait until the beginning of the next node service round before it can be processed. In the worst case this takes a time period $T$. Client display must be further delayed by $(1 + d_{\text{limit}}) \times \left(\frac{T}{n}\right)$, which is the time to transmit the first block of the chunk from buffer into the network from node $d_{\text{limit}}$. Therefore, the total start-up latency for CRS is

$$T + (1 + d_{\text{limit}}) \times \left(\frac{T}{n}\right) \quad (2.1)$$

Any other scheme would give a start-up latency of $2T$, since the first block of a chunk can be placed on the last node. In CRS, the number of nodes must be greater than or equal to the number of blocks in the chunk i.e. $m \geq n$. The scheme can be similarly extended to non-integral values of $r_i$.

A node in a network segment has to gain control of the network in order to transfer data from its buffer. When several nodes in a segment contend with each other for control of the network, the node with the highest priority wins access first. The transmission takes place in the order of decreasing priorities. The CRS layout also introduces some degree of randomization, which is a well-known strategy for load balancing. Otherwise, the highest priority node may become a hot-spot and the lowest priority node may be idle most of the time.
Figure 2-3 shows how four chunks (belonging to different video files), each with a varying number of blocks, may be distributed on four disks using CRS, with $d_{\text{limit}} = \text{node 1}$. This implies that the first block of any chunk cannot be placed on any node with id greater than 1. Each rectangle in the figure represents a block of a video data. The blocks are numbered as (chunk number, block number). Thus, chunk 1 has 3 blocks shown as (1,1), (1,2) and (1,3). Suppose that the node 0 has the highest priority and the node 3 has the lowest priority. The blocks (1,1), (1,2), (3,2), (3,3), (4,1) and (4,2) have a display time $T/4$ and the others have a display time $T/2$, where round time is $T$. It can be easily verified that the distribution of all blocks across the nodes follows the ordering and placement constraints for CRS.

The CRS scheme can be used in conjunction with a very simple and general scheduling scheme (SS) for timely and in-order delivery of the blocks to the client. SS is not specific to CRS, and can be used with any other data layout scheme. SS is discussed in the next subsection along with an example that describes the working of CRS-SS.
2.3 Scheduling Scheme

The server services all streams simultaneously by proceeding in periodic rounds, retrieving a chunk of data into its buffer for every stream, in each round, according to a retrieval schedule. This data is transmitted in the next round from the node buffers in bursts of blocks. Since there is no centralized intermediate buffer to store the blocks retrieved from the various nodes, a scheme must be developed to coordinate the timely delivery of data from the various nodes. The scheduling scheme (SS) enables periodic retrieval and transmission of data for all streams from the storage nodes.

In SS, every round of duration $T$ is divided into sub-rounds with duration $T_d$ given by,

$$T_d \leq T / n$$

where $n$ is the number of nodes on the segment. The nodes prefetch all the data into their buffers in the previous round. This data is then paced out from the nodes in a staggered manner.

Figure 2-4 shows the transmission sequence using CRS-SS with four clients served concurrently. Clients 1, 2, 3 and 4 require chunks 1, 2, 3 and 4 respectively. The data are laid out using CRS as shown in Figure 2-3. The blocks were prefetched in the previous round into their respective node buffers. The devices transmit in the order of their priorities, with node 0 transmitting first in the round for time $T_d$, where $T_d \leq T / n$, with $n = 4$. The start-up latency is $3T / 2$ (since it takes time $T$ to prefetch the data and time $T/2$ to transmit all the first blocks of requested chunks, as the position of the first blocks is restrained to nodes 0 or 1 with $d_{limit} = 1$). For other placement schemes, the first block of a chunk can be placed on any node including the last node, which gives a start-up latency of $2T$. 

34
The CRS scheme can be seamlessly integrated with the Fibre Channel arbitration scheme on an FC-AL by using AL_PAs for node priorities. Arbitrated Loop is a commonly deployed technology for FC SANs. Like shared Ethernet or Token Ring segments, the functional bandwidth available to any individual loop node is determined by the total population of the segment and the level of activity. A node may contain multiple ports, or N_Ports. The attached Arbitrated Loop ports are referred to as Node Loop ports or NL_Ports and each NL_Port has an Arbitrated Loop Physical Address, or AL_PA. Arbitrated Loop assigns priority to AL_PAs, based on the numeric value. The lower the numeric value, the higher the priority. Contention between ports for access to arbitrated loop involves two components: the priority of a loop node’s AL_PA and an access variable that is toggled when a node wins arbitration. An arbitrate primitive, which contains the AL_PA of the arbitrating node, is transmitted whenever the node needs loop access. If two or more nodes are arbitrating at the same time, the NL_Port with the highest priority AL_PA will win.
An NL_Port that observes fairness in sharing the loop will, after it has won arbitration, reset its access bit and be able to arbitrate again only when no other nodes are arbitrating. This scheme allows even low-priority nodes on the loop to win arbitration and prevents starvation of any port. For a detailed description of the FC-AL refer to [73].

With complex topologies such as trees and butterfly networks a token-passing scheme may be used to enforce transmission priority between unsynchronized nodes in the CRS-SS scheme. A token may be generated at the start of every sub-round, and passed from higher to lower priority devices. However, protocols are needed to handle token-loss. Further discussion of this scheme is beyond the scope of this dissertation.

2.4 Performance Evaluation

This section evaluates the performance of the CRS layout scheme with other schemes by using a simulation model. The system consists of 16 nodes containing 50 videos on a network segment. The IBM Ultrastar18ES is used, which is representative of the type of high-performance disks available commercially. The device characteristics are given in the Appendix. The mean response time and throughput obtained with the various schemes are measured.

2.4.1 Simulation Model

A trace-driven simulation model was developed consisting of several disks and a controller on a network segment. Data is transferred directly into the client buffer from the network segment; delays through a gateway or an intermediate network are not modeled in the simulation. The user can specify different storage policies and a network architecture with an arbitrary number of devices. The user can also specify the number and display rates of the stored video files and supply a trace file containing the frame sizes. The movie database consisted of 50 movies.
The characteristics of 5 sample movies is presented in Table 2-1. Further information on the video characteristics can be found in [74]. The simulator is implemented using the CSIM18 package [75]. The various simulation objects are described in further detail next.

Table 2-1. Video traces used in simulation.

<table>
<thead>
<tr>
<th>Video</th>
<th>Peak bit rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jurassic Park</td>
<td>1.01</td>
</tr>
<tr>
<td>Star Wars</td>
<td>4.24</td>
</tr>
<tr>
<td>Mr. Bean (3 TV episodes)</td>
<td>1.76</td>
</tr>
<tr>
<td>Car Race (TV sports event)</td>
<td>3.24</td>
</tr>
<tr>
<td>The Silence of the Lambs</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Each frame consists of one slice.

GOP pattern:

Frame rate: 25 frames per

Number of frames per sequence: 10000.

The controller contains a storage manager that is responsible for storing files on devices. The storage manager distributes data of each file according to one of the data placement schemes described above. When a new client is admitted the storage manager informs the devices on the network segment to start servicing the client. In the current implementation, an admission control policy is not used, and clients are admitted using a FCFS policy. The storage devices modeled are the IBM Ultrastar 18ES disks. The model is based on techniques described in [76,77]. The effect of VBR zones is also modeled. The model is described in further detail in the Appendix.

Client interarrival times are exponentially distributed and each client selects a video using a Zipf distribution [78]. The Zipf distribution, which is characterized
by parameter $\zeta \geq 0$, corresponds to a highly localized request pattern. It has been observed that the requests for movies in video rental stores and Video-on-Demand systems follow a Zipf distribution with $\zeta$ around one [79].

The CRS striping scheme is compared with three other schemes described in literature, namely, *single random* ($S_{\text{Random}}$), *narrow round robin* ($NRR$) and *single round robin* ($SRR$). In the $S_{\text{Random}}$ scheme, each chunk is assigned to a single device in an arbitrary manner. In $SRR$ placement, successive chunks are assigned to devices in a round robin manner as follows: the first chunk is assigned to an arbitrarily selected device, say device $n$. Then the next chunk is placed on device $(n+1) \% MAX\_DEVICES$, where $MAX\_DEVICES$ is the maximum number of devices on the segment, etc. The $NRR$ striping is similar to above, with the difference that blocks of successive chunks are placed in a round robin fashion, with each block of a chunk on a different device.

Unless otherwise specified, the default model contains 16 IBM 18ES Ultrastar devices on a network segment, Zipfian parameter $\zeta = 1$ and the interarrival rate of client requests is exponential with parameter 0.5.

### 2.5 Simulation Results

The variables in the simulation include: (a) the total number of client requests during the entire simulation period; (b) the number of nodes across which the first block of a chunk may be striped, $d_{\text{limit}}$; (c) the total buffer size; and (d) the number of nodes. In order to reduce the complexity of the simulation only one variable was varied during each simulation while keeping the others fixed. The response time is measured as the time interval from the arrival of the new client request until the time when the first block of the requested chunk is transmitted by the node. This includes the retrieval time and the waiting time spent in the node buffer.
2.5.1 Effect of number of clients

The number of clients affects the response time as there is a limit on the number of concurrent users that a system can support. Figure 2-5(a) and (b) compare the performance of the various schemes as the number of clients in the system is increased. The value of $d_{\text{limit}}$ in the CRS scheme is set to 7. The start-up latency for the CRS scheme with $d_{\text{limit}}=7$ is calculated to be 0.3 second and 1.5 seconds, using eqn. (2.1), for round durations of 0.2 second and 1 second respectively. The other schemes have a higher start-up latency of 0.4 second and 2 seconds for round durations of 0.2 second and 1 second respectively. However, the tradeoff is that CRS supports smaller number of clients than the other schemes since the mean response time increases exponentially as the number of clients is increased beyond 300 for 0.2
second round duration, and 1000 for 1 second round duration. The maximum number of clients that can be supported depends on the value of $d_{\text{limit}}$, and the effects of varying $d_{\text{limit}}$ are considered next.

2.5.2 Effect of varying $d_{\text{limit}}$

Figure 2-6(a) and (b) show the effect of varying $d_{\text{limit}}$ for different round durations. As $d_{\text{limit}}$ increases, the start-up latency also increases, according to eqn. (2.1). However, the mean response time drops initially, and then increases. The reason for this is that the load imbalance is largest when $d_{\text{limit}}$ is set to a node at either end. For example, if $d_{\text{limit}}$ is 0, the first blocks of all chunks reside on the first node, and therefore, this node becomes a hot spot. As $d_{\text{limit}}$ is increased, the load on this node reduces and the response time reduces. As $d_{\text{limit}}$ is moved further towards the lowest priority node (with id 15), the number of blocks that reside on nodes $d_{\text{limit}}$ to 15 is increased, which creates load imbalance and increases the response time of the system. From Figure 2-6(a), when $d_{\text{limit}} = 3$, start-up latency = 0.25 seconds for CRS – a reduction of 37% over other schemes.
2.5.3 Effect of varying the total buffer size

In the other simulations there is sufficient buffer provided on each node so that all client demands can be met. Figure 2-7(a) and (b) show the effects of a limited buffer for round times of 0.2 second and 1 second respectively. In general, the CRS scheme gives the lowest mean response time among all schemes as the buffer size is increased. A buffer size of 0.5 MB is sufficient to support 200 streams, when the round duration is 0.2 sec, and a size of about 6.5 MB is sufficient to support 1000 streams.

Figure 2-7. Variation of the mean response time with the buffer size for (a) round time: 0.2 second and (b) round time: 1 second.

2.5.4 Effect of varying number of nodes

To evaluate the scalability of the server, the response time is measured as the number of nodes is increased. Figure 2-8(a) and (b) show the change in the mean...
response time as the number of nodes is increased from 14 to 30. In general, it is observed that the CRS scheme has the lowest mean response. For CRS, the values of \(d_{\text{limit}}\) increase from 6 to 14 as the number of nodes increase from 14 to 30, i.e., \(d_{\text{limit}} = 6\) when the number of nodes is 14. All the schemes show a large reduction in the mean response time as the number of nodes is increased. The node queues become smaller as the number of nodes is increased so that the mean response time decreases and the system throughput increases.

![Graphs showing mean response time vs. number of nodes for different schemes.](image)

Figure 2-8. Effect of number of nodes on mean response time for (a) round time: 0.2 second and (b) round time: 1 second.

### 2.6 Summary

A new data layout scheme called CRS is presented for a SAN-based video server called SAM. The basic idea behind CRS is to fragment the data that must be transmitted in a round, and place it on different nodes, so that portions of the data
that are needed earlier are placed on nodes that transmit earlier. CRS provides a solution to the problem of potential out-of-order delivery of video blocks for devices on a logical loop, and gives the smallest start-up latency among all schemes. The simulation results show that CRS can give a reduction of up to about 40% in the start-up latency compared to other schemes. In addition, CRS exhibits good scalability as the number of nodes in the system increases.
Chapter 3

Admission control with VBR scalable video

3.1 Background

A video server must service multiple concurrent clients with specified QoS guarantees. The resource requirements of clients are specified by QoS parameters, and different applications have different QoS parameters. For instance, a shared whiteboard service requires an interactive, error free and low bandwidth transmission, whereas a VOD system requires low jitter, low delay and higher bandwidth but can tolerate some transmission errors. Due to the high bandwidth requirements of multimedia data it is generally stored in a compressed format, which is inherently VBR. When VBR video is used storage and retrieval of video are complicated since the resource requirements of VBR streams are highly variable.

Multi-resolution or layered coding techniques split signals into components of varying importance [80,81]. The aggregation of these components reconstructs the original data, but subsets of the data can also provide varying degrees of approximation to the original signal. The layered streams can be used to serve a variety of clients with different decoding capabilities. Another advantage is that it also allows a video-on-demand service provider to provide customers with different levels of service.
Consider the storage subsystem storing VBR multi-resolution video. It is possible that the QoS requirements of client(s) may be violated in one of the following situations: 1) The server may suffer from periods of transient overload resulting from a high degree of customer interactivity. 2) In the event of a failure, streams from the failed node must be reassigned to the other nodes with minimal disruption in service. The basic requirement of a solution to these two problems is that the QoS guarantees provided to the clients at admission time must be upheld.

In [82], the authors propose a QoS negotiation scheme for efficient failure recovery in multi-resolution video servers. The proposed scheme exploits the multi-resolution property of video streams. A spectrum of resolution levels is defined that permits acceptable performance when a disk fails by using a smaller amount of resources. However, they consider only constant bit rate (CBR) scalable video and provide deterministic guarantees to both minimum and higher resolution layers.

In [83,28], statistical guarantees are provided to clients retrieving VBR scalable video. In [83], the authors propose a scheme to deal with disk overload by dropping appropriate frames in the MPEG video. They propose a number of classes of service, corresponding to various probabilities of loss. Requests with a higher probability of loss threshold are degraded before those with lower thresholds. In [28], the admission control scheme ensures that the total number of frames discarded by the multimedia server during overflow rounds will not exceed the cumulative loss tolerance of all the clients. In order to ensure that the individual service requirements of the clients are not violated, the server distributes the discarded frames among all the clients with the goal of minimizing the cumulative frame loss.

Our work differs from theirs in that deterministic guarantees are provided for minimum layers in conjunction with statistical guarantees for higher layers of each client, and disk failure is handled. The reason for providing deterministic guarantees to the minimum layers is that the effects of a loss in this layer can propagate
adversely to other layers. In MPEG video coding, frames are coded in one of three modes: intraframe (I), predictive (P) or bidirectionally-predictive (B). These modes provide intrinsic layering in that an I-frame can be independently decoded, while P-frames require I-frames, and B-frames generally require I- and P-frames to decode. In each temporally scalable MPEG trace the following layering scheme is used: I- and P-frames make the base layer, and B-frames make the enhancement layer. In an MPEG-encoded video stream, discarding an I-frame eliminates all the succeeding P- and B-frames until the next I-frame is accessed. If an I-frame is lost, a perfect picture cannot be displayed again until the next I frame is received, which is typically every 15 frames or half-second. A half-second loss of video is noticeable and unacceptable. Therefore, for MPEG, if an I-frame is lost, the QOS may be inadequate.

In this chapter, we discuss overload tolerance and graceful degradation in case of disk failure with VBR scalable video. We present an admission control scheme that provides deterministic guarantees for minimum or necessary layers, and statistical guarantees for higher layers of client streams. The basic idea here is that minimum layers must not be dropped, while some loss in the higher layers can be tolerated. The disks must service minimum layers before the higher layers in each round, and it may require up to two complete disk sweeps in a round to service all the layers, which is factored in the admission control algorithm. A new data placement policy called Partial-Bundled is described and compared with other strategies. The advantage of this placement scheme is that each disk can service the minimum and higher resolution layers with only one sweep in each round. We also propose two heuristic algorithms to provide for graceful degradation in the presence of disk failure. The heuristic algorithms attempt to maximize the rewards (related to average quality of a stream) while reducing the resolution of clients.
3.2 Multi-resolution Video System Model

A multi-resolution video system model is presented here for variable spatial resolution streams.

3.2.1 Multi-resolution video

A chunk is the data retrieved for each stream during a round. A chunk has constant time length in terms of playback duration, but the size of the chunk can vary from round to round. Let us assume that a given video $V$ has $g$ chunks such that

$$V = \left\{ S_0 \cup \ldots \cup S_i \ldots \cup S_{g-1} \right\}.$$ 

Also, each chunk $S_i$ has up to $Q$ display resolutions of data such that

$$S_i = \left\{ S_i^0 \cup \ldots \cup S_i^j \ldots \cup S_i^{Q-1} \right\}$$

where, $S_i^j$ represents the $j$th resolution layer ($0 < j \leq Q-1$) for the chunk. In order to display a video stream with resolution $j$, it requires reading of data from the base or $0$th layer up to the $j$th layer. For example, to view resolution $k$ of $S_i^j$, we need to retrieve $\{ S_i^0 \cup S_i^j \ldots \cup S_i^k \}$. Let the number of bits in layer $j$ of the $i$th chunk be $n_i^j$.

The number of bits varies across chunks of the same layer as well as different layers in each video. Each chunk is broken into fixed-size blocks where a block is the maximum amount of logically contiguous data stored on a single device.

3.2.2 Client streams

The QoS requirements of clients can be expressed in terms of the minimum and maximum resolution levels. Each client has a maximum resolution level, which is the resolution level that is provided to the client in normal mode. Each client also has a minimum resolution level, which the client can lower the resolution to in degraded mode. Resolutions below the minimum resolution level are not allowed. For example, client $i$ has resolution levels $min_i$ and $max_i$ such that,
\[0 \leq \text{min} \leq \text{min}_i \leq \text{max}_i \leq \text{max}\]

Two models of client streams are considered – the \textit{fixed resolution} model and the \textit{variable resolution} model. In the \textit{fixed resolution} model each video provides a fixed minimum resolution level. Thus, the values of \text{min}_i are the same for all clients that access a particular video. In the \textit{variable resolution} model, the minimum resolution level varies for different streams that access a particular video. In addition, each layer is associated with a numerical value termed \textit{reward}. The idea is to consider the priority of the client [82]. Thus, each client is assigned several rewards that indicate its value to the system when the stream is serviced with the assigned QoS level successfully.

### 3.2.3 Data Placement

A video file is broken up into chunks \(F_i\) such that the video file \(F = \{F_1 \cup F_2 \cup \ldots F_n\}\), \(F_i \cap F_j = \emptyset\) and \(i, j \in \{1, 2, \ldots n\}\). The chunks are distributed in a round robin manner across the different nodes [81]. In this data placement strategy, each chunk is assigned to a single node, and the chunks in a video are distributed on successive nodes. The placement of a chunk on a node (a large bandwidth disk) can take place in one of the following ways:

**Zonal placement:** This strategy is similar to the one discussed in [84], and divides each disk into different zones, one for each type of layer. Each layer is placed in the zone designated for that type. The placement within each zone proceeds in a sequential manner, in the order in which blocks of each video are allocated. The minimum resolution layers are placed in the outer (higher bandwidth) zones, and the higher resolution layers are progressively placed toward the inner (lower bandwidth) zones. In Figure 3-1(a) the Zonal placement for two videos with four resolution layers is shown.
Figure 3-1. Different ways in which a chunk may be placed on a node. (a) Zonal  (b) Bundled  (c) Partial-Bundled.

**Bundled placement:** Figure 3-1(b) shows how the different layers in a chunk of a video are placed sequentially on the same disk. A placement strategy similar to this is described in [84].

**Partial-Bundled placement:** A new placement scheme called Partial-Bundled is proposed in this chapter, and is an extension of the Bundled placement scheme. It is used when all clients follow the fixed resolution model. All the minimum resolution levels use the bundled placement, and are placed sequentially on disk on the outer zones. The remainder of the disk is divided into zones and the remaining layers in each chunk are placed according to the zonal placement strategy. Alternately, the remaining layers also use bundled placement and are placed sequentially. This is shown in Figure 3-1(c), assuming that there are two minimum resolution layers – 0 and 1.
Since the minimum layers are provided deterministic guarantees they must be retrieved first in a round for each client, before any of the higher resolution layers. Otherwise, the requests for minimum resolution layers may be dropped due to insufficient round time, which is a possibility if the higher resolution requests are serviced earlier. We assume that the requests are available at the start of a round, and can be serviced in a given order using an elevator disk scheduling algorithm. In the Bundled, Random and Zonal placement strategies this implies that the disk may have to be scanned twice in order to service all the layers. Note that this may be the case even when clients use the fixed resolution model. But in the Partial-Bundled disk placement scheme this extra disk sweep can be avoided by scanning in a single direction from the minimum resolution layers towards the higher resolution layers. The Partial-Bundled placement can also take advantage of contiguous placement for chunks in the minimum resolution layer, which further reduces the seek time.

Random: The different layers are distributed randomly across the disk.

3.3 Admission Control

In this section a new admission control scheme is described. In the proposed scheme, deterministic service is provided for the minimum resolution layers and statistical guarantees are provided for the remaining layers for each stream. The deterministic admission control scheme is similar to the Instantaneous Maximum and Future-Max schemes described in [85, 86], which are modified to handle the cases of multi-resolution video and node failure. The statistical admission control scheme admits streams so that the statistical estimation that there is no overload can be met. The advantage of the Partial-Bundled scheme over the others is demonstrated by eqn. (3.5). These schemes are discussed in greater detail next.
3.3.1 Deterministic Admission control

A stream block schedule is created for each stream, which contains a set of vectors describing the resource requirements of the stream for the different layers on a particular node. Each vector in this schedule contains one entry per slot, which is the number of blocks that must be read for a given layer in a particular round. Let $BS_{\text{min}}^k(i)$ represent the sum of the vector schedules for the minimum resolution layers of a stream $k$ (i.e. base to $\min_k$) at the $i$th slot. The guaranteed number of blocks that can be read in a slot is called $N_{\text{det}}$. This number is calculated by running a calibration program that determines the maximum number of blocks that can be read when the blocks are located on the disk under the worst conditions. Let $K$ be the number of streams currently being served, and the starting intervals of the streams be represented as $I_{\text{start}}^k$. Then the condition to admit or reject a new stream with vector schedule $BS_{\text{min}}^{K+1}$ can be formulated as shown in eqn. (3.1).

$$\gamma(l) = \left( \sum_{k=1}^{K} BS_{\text{min}}^k(l - I_{\text{start}}^k) + BS_{\text{min}}^{K+1}(l - I_{\text{start}}^{K+1}) > N_{\text{det}} \right)$$

(3.1)

where $\gamma(l)$ is a boolean function that is evaluated from the starting slot until the slot at which the playback of the new stream ends. The acceptance or rejection is based on the following expression –

Reject: $\exists l \ s.t. \ \gamma(l) = True$

Accept: $\forall l \ \gamma(l) = False$

(3.2)

In the event of a node failure, the load on the failed node is distributed across the functional nodes. Therefore, the load on each node is increased by a fraction $\alpha N_{\text{det}}$, where $\alpha$ depends on the data replication scheme used. For example, $\alpha = \frac{1}{2}$ with
mirroring [87], and $\alpha = \frac{1}{D}$ with chained declustering [88]. Therefore, eqn. (3.1) is modified to handle disk failure as shown in eqn. (3.3),

$$\gamma(l) = \left( \sum_{k=1}^{K} BS_{\min}^k (l-l_{\text{start}}^k) + BS_{\min}^{K+1} (l-l_{\text{start}}^{K+1}) > N_{\text{det}} (1-\alpha) \right)$$ (3.3)

Eqns. (3.2) and (3.3) must be satisfied before a new stream can be admitted. They provide deterministic guarantees to the minimum resolution layers of a video even in the case of node failure.

3.3.2 Statistical Admission control

In this scheme, the mean and variance of the service time for a single block, when the elevator disk scheduling algorithm is used, are determined using techniques described in [89]. Then, using the Central Limit Theorem it is shown that the probability distribution of the service time follows an approximately normal distribution [83,90]. This can be used to obtain a statistical estimate of the number of blocks $N_{\text{stat}}$, for each placement scheme, that can be retrieved from a disk in a round such that the probability of disk overload does not exceed a given value $p_{\text{miss}}$. The values of $N_{\text{stat}}$ for different $p_{\text{miss}}$ are determined offline and pre-stored in a lookup table for runtime use.

The statistical condition to admit or reject a new stream with vector schedule $BS_{\text{max}}^{K+1}$ can be formulated as shown in eqn. (3.4), where, $BS_{\text{max}}^k$ represents the sum of the vector schedules for all layers (base to $\text{max}_k$) of stream $k$.

$$\gamma(l) = \left( \sum_{k=1}^{K} BS_{\text{max}}^k (l-l_{\text{start}}^k) + BS_{\text{max}}^{K+1} (l-l_{\text{start}}^{K+1}) > N_{\text{stat}} \right)$$ (3.4)

Reject: $\exists l \text{ s.t. } \gamma(l) = \text{True}$
The expected service time for one block on a disk is given as

$$E(\text{Service Time}) = E(\text{Positioning Time}) + E(\text{Transfer Time})$$

For a zoned disk, which has different transfer rates in different zones, the average transfer rate $r^\text{avr}_t$ is determined as a weighted sum of transfer rates across zones. The expected positioning time is given by the sums of expected seek time and rotational latency.

The expected seek time varies for the different placement schemes. It can be approximated by considering the $N$ requests as a random sample size $N$ between 0 and Maximum Seek Distance. This creates $N+1$ random points on the interval $(0, \text{Maximum Seek Distance})$. The $N+1$ random points create $N+2$ intervals whose lengths have the same distribution [91]:

$$\Pr(\text{Seek Distance} \geq z) = (1 - \frac{z}{\text{Maximum Seek Distance}})^{N+1}$$

The minimum resolution layers are provided deterministic guarantees, and must be serviced before the higher resolution layers. Therefore, in order to service all the requests in a round, the disk may have to be scanned twice. However, in the Partial-Bundled placement scheme, with the fixed resolution model, only one sweep on the disk is required in a round. Therefore, the Maximum Seek Distance is given by

$$\text{Maximum Seek Distance} = c^* \text{ Maximum Cylinders},$$

$$c = \begin{cases} 
1, & \text{Partial-Bundled placement} \\
2, & \text{other placement strategies}
\end{cases}$$

(3.5)
This can be substituted in the following equation, with \( z \) expressed as a function of seek time obtained from the disk seek-time vs. distance function, as shown in [89]:

\[
E(\text{Seek Time}) = \int_{0}^{\infty} \Pr(\text{Seek Distance} \geq x)dx
\]

Also, in all placement schemes with the exception of Random, the seek time decreases when the request has spatial locality in the form of runs of sequential requests, assuming the runs are uniformly distributed [89]. The seek time in this case is

\[
E[\text{Seek Time}] = \frac{\text{Number of consecutive Blocks}}{}
\]

The variance in the service time to retrieve one block from the disk can be obtained similarly. Using the Central Limit Theorem, for sufficiently large \( N \), the pdf of the service time \( T \), can be approximated by a normal distribution [83]. Given the mean and variance of this distribution and the round time, the value of \( N_{stat} \) can be calculated for a given overload probability \( p_{\text{miss}} \).

There are several ways to calculate the overload probability \( p_{\text{miss}} \). One possible way is to consider the number of frames that are retrieved for a stream in normal rounds and overload rounds and allow this number to be sufficiently large as the following QoS requirement, given that \( a \) is the percentage of frames that must be retrieved for a given stream, at any resolution level:

\[
p_{\text{miss}}F_{\text{min}} + (1 - p_{\text{miss}})F \geq aF \tag{3.6}
\]

where \( 0 \leq p_{\text{miss}} \leq 1 \). Here \( F_{\text{min}} \) is the smallest number of frames of minimum resolution guaranteed to be retrieved in overload rounds (using the deterministic admission control scheme), and \( F \) is the number of frames that are retrieved in other rounds at a particular resolution. For a given set of streams that have been admitted,
or are waiting to be admitted to the server, the smallest overload probability or the average overload probability could be selected to calculate $N_{stat}$.

### 3.4 Buffer constraint

To schedule the video streams the dual buffering method (i.e., the buffer space is chosen to be twice as large as that needed to accommodate the data retrieved in one round) is used. Let $Buf$ be the total buffer size available on a node. In order to ensure that sufficient buffer is available to service streams with minimum resolution levels even in degraded mode, the following condition must hold:

$$2(N_{det} * B) \leq Buf$$  \hspace{1cm} (3.7)

In the normal mode, there should be sufficient buffer to service the maximum resolution levels, as shown in eqn. (3.8).

$$2(N_{stat} * B) \leq Buf$$  \hspace{1cm} (3.8)

If eqn. (3.8) is satisfied, then eqn. (3.7) is also guaranteed to be true.

### 3.5 Admission control decision

The server admits the client if the admission control conditions given by eqns. (3.2), (3.3) and (3.4) can be satisfied within the client’s acceptable delay time. Otherwise, the client is rejected.

#### 3.5.1 Reward-based multi-resolution schemes

When a node failure occurs, the streams executing on the failed node must be supported on the remaining nodes. This may require that the resolution of the admitted streams be degraded. In this section, heuristic algorithms are developed to adjust the resolution of the admitted clients when a node failure occurs. In [82], heuristic algorithms were presented for CBR streams, which are not applicable to the VBR streams considered in our design.
With Eqn (3.3) it can be guaranteed that sufficient bandwidth is reserved for minimum resolution levels to handle the load increase in the case of node failure. A new vector schedule is created for each of the streams on the failed disk. Suppose that the number of streams executing on a node is increased to $M$ in degraded mode. Let $BS^k_{\text{max}}(i)$ represent the sum of the new vector schedules for the maximum resolution layers of a stream $k$ at the $i$th slot. In order to support the maximum resolutions for each stream eqn. (3.9) must be satisfied on each node:

$$\gamma(l) = \left( \sum_{k=1}^{M} BS^k_{\text{max}}(l - l^k_{\text{start}}) > N_{\text{stat}} \right), \forall l: \gamma(l) = False$$  \hspace{1cm} (3.9)$$

Otherwise, the resolutions of streams must be degraded from the maximum resolution level. Suppose that the resolution of the $k$th stream is degraded to $mid_k$, where, $min_k \leq mid_k \leq max_k$. Then values for $mid_k$ must be selected so that eqn. (3.10) can be satisfied on each node:

$$\gamma(l) = \left( \sum_{k=1}^{M} BS^k_{\text{mid}}(l - l^k_{\text{start}}) > N_{\text{stat}} \right), \forall l: \gamma(l) = False$$  \hspace{1cm} (3.10)$$

If eqn. (3.10) is not violated then the buffer constraint given by eqn. (3.8) is also met. It can be easily verified that the QoS requirement given by eqn. (3.6) is satisfied for the streams whose resolutions are degraded to $mid_i$, when the number of frames to be retrieved in degraded mode per round, $F$, is reduced, for a given value of parameter $a$. In the reward-based scheme there is a reward $c^a_i$ associated with the $a$th layer of video $i$, where video $i$ consists of $l_i$ layers. The goal is to drop layers such that the reward associated with the remaining layers can be maximized and the statistical guarantees are not violated. This can be formalized in the following manner:
The resolution degradation problem is to find a resolution mid for every stream \( k \) that maximizes the reward \( \sum_{k=1}^{n} c_k^{mid} \) while satisfying eqn. (3.10) on each node. The reward \( c_i^u \) is the value associated with the \( u \)th layer of video \( i \), and \( n \) is the number of streams in the system.

The above problem is reduced to a multiple choice knapsack problem which is known to be NP-complete [92]. Two heuristic algorithms are presented for this problem. First two heuristic functions \( \phi_1(u) \) and \( \phi_2(u) \) are defined as

\[
\phi_1(u) = \frac{c_i^{max} - c_i^{u-1}}{\sum_{k=1}^{N_{avg}^k}}
\]

where \( N_{avg}^k \) is the average number of blocks requested for the \( k \)th layer of video \( i \) in a slot, and

\[
\phi_2(u) = \frac{c_i^{max} - c_i^{u-1}}{\sum_{k=1}^{N_{tot}^k}}
\]

where \( N_{tot}^k \) is the number of blocks requested by the stream \( k \)th layer of video \( i \) in the maximum overload round. The maximum overload round, \( \psi^{max} \), is the round with the largest overload (i.e., \( N - N_{stat} \) is maximum), where \( N \) is the cumulative number of blocks that must be retrieved by these streams in that round. The numerator of both the heuristic functions represents the loss in rewards from dropping layers \( u \) to \( max_i \) of video \( i \). The denominators of \( \phi_1(u) \) and \( \phi_2(u) \) are proportional to the average and maximum bandwidth respectively that is returned to the server from dropping layers \( u \) to \( max_i \). A stream with a lower value of \( \phi_1(u) \)
should be degraded first, since it returns more average bandwidth to the server, with smaller loss in rewards. We may use $\phi^i_j(u)$ to break ties between streams that have the same value of $\phi^i_j(u)$, or break ties arbitrarily.

In the Partial-Bundled placement scheme, the higher resolution layers may be placed according to the Zonal placement scheme. If one or more layers can be removed completely, it may result in seek time saving since the disk head has to sweep over a shorter distance. This may allow a node to service more streams if it is allowed to work ahead. In the first heuristic algorithm (HAL1), blocks are dropped layer by layer for the different streams, starting from the highest resolution layer. For example, suppose a number of clients have $\min_i < \max_i = \max$ (maximum resolution). The resolutions of these clients are decreased in the order of increasing $\phi^i_j(\max)$, by dropping layer $\max$, until eqn. (3.10) is satisfied. If eqn. (3.10) is not satisfied, the resolutions of clients have to be reduced further by dropping layer $\max - 1$ if possible, and so on. In addition, only the resolution of those clients that execute in some overload round is reduced. The resolution of client $i$ is represented as $R(i)$. The pseudocode for HAL1 is as follows, given $M$ streams in the system:

**Heuristic algorithm 1 (HAL1):**

1. $u = \max$;
2. while eqn. 3.10 is not satisfied
3. Create $LIST1 = \{\phi^i_1(u), \phi^i_2(u), \ldots, \phi^i_j(u), \ldots\}$, $i$ executes in $\psi^\max$, $R(i) \geq u$, $\min_i < u$;
4. if $LIST1 == \phi$
5. $u = u - 1$;
6. if $u == min$
7. break;
8. endif
9. else
10. Select smallest $\phi^i_j(u)$ from $LIST1$;
11. Reduce resolution of client $i$ to $u - 1$;
12. endif
13. endwhile
In the second heuristic algorithm *HAL2*, the heuristic functions for different layers are compared together, instead of layer by layer. The resolutions of these clients are decreased in the order of increasing $\phi_i(u)$, where $\min u \leq \max$, until eqn. (3.10) is satisfied. This may allow consecutive layers of a client to be dropped together, instead of one layer at a time as in *HAL1*. This is useful when the higher resolution layers are placed according to the Bundled placement scheme. The pseudocode for *HAL2* is given next.

**Heuristic algorithm 2 (HAL2):**

1. Create $\text{LIST1} = \{\phi_1^1(u),\phi_1^2(u),\ldots,\phi_1^i(u),\ldots\}$, $(\min < u \leq \max)$;
2. $\textbf{while}$ eqn. 3.10 is not satisfied
3. Select smallest $\phi_i^1(u)$ from $\text{LIST1}$, $i$ executes in $\psi^{\max}, R(i) \geq u$, $\min_i < u$;
4. Set current resolution of client $i$ to $u$;
5. Remove $\phi_i^1(u)$ from $\text{LIST1}$;
6. $\textbf{endwhile}$

In the next section the above schemes are compared.

### 3.6 Simulation Results and Analysis

A trace-driven simulation model is developed to compare the performance of the data layout schemes and the HAL1 and HAL2 algorithms. The simulation model was developed using the CSIM18 package [75]. The user can specify different storage policies, a network architecture with an arbitrary number of nodes, the number and display rates of the stored video files, and supply a trace file containing the frame sizes. The movie database consisted of 40 2-layer temporally scalable MPEG-4 videos striped on 5 nodes with a block size of 16KB.

The characteristics of 3 sample movies, for different quantization levels ($QL$), are presented in Table 3-1. The average quality of a layer (given by its PSNR value) is used as the reward for that layer. Further information on the video characteristics can be found in [93].
Table 3-1. Sample characteristics of movies in database.

<table>
<thead>
<tr>
<th>Movie</th>
<th>Base layer mean bitrate (kbits/s)</th>
<th>Base layer mean quality (db)</th>
<th>Enhancement layer mean bitrate (kbits/s)</th>
<th>Base + Enhancement layer mean quality (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citizen Kane (QL:3)</td>
<td>256.0</td>
<td>26.5</td>
<td>26</td>
<td>34.2</td>
</tr>
<tr>
<td>Silence of lambs (QL:2)</td>
<td>128.0</td>
<td>27.2</td>
<td>31.4</td>
<td>32.9</td>
</tr>
<tr>
<td>Silence of lambs (QL:3)</td>
<td>256.0</td>
<td>27.5</td>
<td>29.7</td>
<td>34.31</td>
</tr>
</tbody>
</table>

Each node contains two buffers to hold incoming requests. The first buffer called *minimumResolutionBuffer* holds data for requests for the minimum resolution layers and the second buffer called *higherResolutionBuffer* is for the remaining layers. The node first schedules all the requests from the *minimumResolutionBuffer*, in each round, using the CLOOK scheduling algorithm. The requests in the *higherResolutionBuffer* are then scheduled in the remaining round time, and those that could not be retrieved by the end of the round are discarded. This scheme ensures that the requests for minimum resolution layers are not dropped due to insufficient round time, which is a possibility if the higher resolution requests are serviced earlier. The node characteristics are presented in the Appendix. The admission control module checks if eqns. (3.2) to (3.4) are satisfied, otherwise it rejects the client.

Table 3-2 shows the values of $N_{det}$ and $N_{stat}$, calculated using the results in Section 3, for the different placement schemes, assuming overload probability $p_{miss} \leq 0.03$. The minimum resolution layers in the Partial-Bundled placement occupy no more than half of each node. Since these layers can be restricted to a portion of the node, the worst case seek time is smallest here as compared to the
other schemes; hence, the Partial Bundled scheme has the largest value of \( N_{det} \). The value of \( N_{stat} \) is calculated using the disk model described earlier. The Partial-Bundled scheme also has the largest value of \( N_{stat} \). The reason for this is that only one sweep of the node is required to service all layers in a round, which results in lower seek times for Partial-Bundled. All the placement schemes except Random take advantage of the fact that at least 2 contiguous blocks are retrieved for rounds of duration 2 seconds or more.

Table 3-2. Values of \( N_{det} \) and \( N_{stat} \) for the placement schemes.

<table>
<thead>
<tr>
<th>Round time (sec)</th>
<th>Random</th>
<th>Bundled</th>
<th>Zonal</th>
<th>Partial Bundled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( N_{det} )</td>
<td>( N_{stat} )</td>
<td>( N_{det} )</td>
<td>( N_{stat} )</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>110</td>
<td>42</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>134</td>
<td>230</td>
<td>244</td>
<td>414</td>
</tr>
<tr>
<td>3</td>
<td>226</td>
<td>350</td>
<td>413</td>
<td>620</td>
</tr>
</tbody>
</table>

Figure 3-2(a) shows the number of clients that can be admitted using the different placement schemes, with \( \alpha = 0.5 \). The clients follow the fixed resolution model. In all cases, half the clients were provided deterministic guarantees for at least the base layer, and the remaining half were provided deterministic guarantees for both the base and enhancement layers. For a round duration of 1 second the Partial-Bundled placement scheme can admit more than double the number of clients admitted by other schemes. It can also admit over 10% more clients than the other streams with round durations of 2 seconds or more.
Figure 3-2. Number of clients admitted for different placement schemes and round durations, with $N_{det}$ and $N_{stat}$ values as shown in Table 3 and with (a) $\alpha=0.5$ (b) $\alpha=0.2$. The average service time (in msec) of the disks in a round for different placement schemes and with (c) $\alpha=0.5$ and (d) $\alpha=0.2$.

Figure 3-2(c) shows that the node utilization is low, between 10 - 25%. The provision of stringent deterministic guarantees and higher bitrate of the base layer is the limiting factor in the number of streams that are serviced. Figure 3-2(b) and Figure 3-2(d) show the number of clients that can be admitted using the different
placement schemes, with $\alpha = 0.2$, and the corresponding node utilization respectively.

The heuristic algorithms HAL1 and HAL2 were also simulated. The performance of both was identical with two-layered scalable video. Therefore, pseudo-traces of video with 3-4 levels were generated with random rewards to compare their performance. In order to compare the two algorithms fairly, they were run starting from identical system states at the point of disk failure. Figure 3-3 shows the sum of rewards of the clients after their resolutions were degraded. The results show that HAL2 is able to degrade the resolutions of the admitted streams more effectively since it provides higher overall reward for the admitted streams. Therefore, HAL2 performs better than HAL1 for round durations from 1 to 3 seconds.

![Figure 3-3. Performance comparison of HAL1 and HAL2 algorithms.](image)
Figure 3-4 shows the comparison of HAL2 with Optimal for 30 clients and a 1 second round duration. The Optimal algorithm enumerated all feasible solutions and selected the one with the highest sum of rewards.

![Graph showing comparison of HAL2 and Optimal algorithms.]

Figure 3-4. Performance comparison of HAL2 and Optimal algorithms.

### 3.7 Summary

In this chapter, we presented an admission control scheme to provide deterministic admission to the minimum or necessary layers and statistical admission to the higher layers of a client video stream accessing VBR multi-resolution video. A data placement scheme called Partial-Bundled is proposed and compared, using simulation, with other schemes. For a round duration of 1 second the Partial-Bundled placement scheme can admit more than double the number of clients admitted by other schemes. It can also admit over 10% more clients than the other placement strategies with round durations of 2 seconds or more. We have presented two heuristic algorithms, HAL1 and HAL2, to degrade the resolutions of the streams in case of disk failure while maximizing the server-perceived rewards. The simulation results show that HAL2 can degrade the resolution effectively.
Chapter 4

Support For Interactive Operations

4.1 Background

Interactive multimedia presentations are an important element of many application domains such as video on demand, news on demand, education and training, home shopping and others. It is necessary to efficiently support interactive operations such as fast-forward (FF), rewind (Rew) and pause with Quality of Service (QoS) guarantees on video servers that provide these services. The main problem in handling FF/Rew is that the amount of resources required is not precisely predictable, and depends on the interactions that a user may take. In addition, interactive operations have high data rates and strict temporal constraints.

We show that the statistical multiplexing of available resources can reduce the resources required to support interactive service on a parallel server. An analytical model is developed to predict the probability with which stream requirements can be met for a given node buffer level. A FF/Rew stream on a node can use the resources released by other FF/Rew streams on the same node, if any are available, without requiring any additional buffer. In addition, a small amount of buffer may also be reserved for use exclusively by the FF/Rew streams. Since the server composed of distributed nodes, it is possible that all resources may be released and requested from different nodes. In this case FF/Rew streams cannot leverage the unused resources for their own requirements. However, it shall be shown that the probability of this worst-case scenario occurring is very small. Simulation results using the given scheme are also presented.
4.2 Related work

A queuing model for statistical resource sharing for interactive operations was discussed in [29]. However, in their model they consider a single node with a single shared buffer for all streams. Therefore, any bandwidth released is always available for all streams to use. This work is different in that the buffers of the different nodes are distributed, and buffers that are released on one node are not accessible to the streams on the other nodes.

Two schemes (prefetching approach and grouping approach) are proposed for scheduling interactive operations on clustered constant-bit-rate video servers in [61]. Their approach allocates the video files in a round-robin manner and uses precise allocation and scheduling to reduce resource requirements. In the prefetching approach, a buffer is required to store some video data blocks and a browsing operation has a start-up delay. In the grouping approach, different operations such as normal play, fast-forward and fast-backward are clustered into different groups with different paces. The primary difference between their approach and the one presented here is that the former provides deterministic guarantees whereas the latter provides statistical guarantees to clients. One advantage of the proposed scheme is that a browsing operation can start immediately without any start-up delay. Also, some video servers such as Rio [55] use random data placement. Browsing with random data placement is supported in the proposed scheme, but the grouping and prefetching approaches require a round-robin allocation.

A lower resolution component is used for interactive operations in [31,94]. However they do not consider resource sharing between streams. Chen et al. [32] propose a segment skipping scheme for interactive operations, where a segment can be a set of Group of Pictures (GoP) of an MPEG video. This scheme is used in SAM for providing interactive operations.
4.3 Proposed Approach

Since a multimedia server has only limited resources (e.g. the available number of nodes, processors and buffers), it can serve only a restricted number of clients. Therefore, the number of clients that can be serviced concurrently is limited by means of an admission control mechanism, in order to avoid service degradation. A resource is reserved for each stream on any one node during every round to ensure that QoS guarantees can be provided during normal play. This resource is the number of buffer slots that must be reserved for that stream so that stream requirements can be met during normal play. Streams switch from fast-forward mode to normal and vice-versa at random times. Every stream that goes into interactive mode releases the resource that was reserved during normal play and requires a new resource from possibly another node. In the proposed approach, an interactive mode stream can utilize the resources that are released by other streams on the node that it is accessing currently, if any are available. In this section, an analytical model is developed that relates the statistical probability of satisfying all stream requests with a given number of streams, nodes and buffer levels.

The following assumptions are made in the derivation of the model. Firstly, it is assumed that the data blocks are striped using single random striping, where a segment (data required by a stream in a round) is placed on a single node. Therefore, only one node is accessed in each round for a normal play stream request. Secondly, the segment sampling method [32] is used for FF/Rew, so that only a single node is accessed by a FF/Rew stream during a round. Thirdly, a low-resolution component of the stream may be used to satisfy FF/Rew stream requests, so that $F \leq R$, where $F$ is the number of buffer slots required by a fast-forward stream, and $R$ is the smallest number of slots released by a stream. Lastly, it is assumed that all nodes are equally likely to be selected by a stream during normal play and FF/Rew.
Example 1. The different ways in which 2 FF/Rew streams and released slots may be distributed across 2 nodes in a round is shown in Figure 4-1.

<table>
<thead>
<tr>
<th>Node 1</th>
<th></th>
<th>Node 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-1. Different ways in which released slots (R) and FF/REW requests (F) of 2 streams may be distributed across 2 nodes.

In Figure 4-1 the released slots for the 2 streams are shown as R, and the resources required during FF/Rew are shown as F. In the first two rows of Figure 4-1, both the FF/Rew requests can be satisfied without requiring any additional
buffer. The reason is that the released slots of both the streams lie on the same node as the FF/Rew requests, and can be used to satisfy the FF/Rew requests (since \( F \leq R \)). However, in the third row, the FF/Rew requests and the released slots are placed on different nodes. The buffer requirement in this case is \( 2F \) slots in order to service both FF/Rew streams.

The buffer requirements for the different combinations are summarized in Figure 4-2. Clearly, the released slots \( R \) and \( RR \) have a buffer requirement of 0. The combination \( RFF \), which consists of one released slots and two fast-forward requests, has a buffer level of \( F \) since only one of the two requests can use the released slot.

![Figure 4-2. Buffer requirement for different combinations of the released slots (R) and FF/REW requests (F).](image)

### 4.4 Analytical Model

In this section, the analytical model for the proposed scheme is presented. In this model the statistical probability of satisfying stream requests for a given number of nodes, streams and buffer level is determined.

**Theorem 1**: Given that \( n \) FF/Rew streams execute on \( D \) nodes, the probability of satisfying all stream requests with a buffer level of \( kF \) in a round is given by
\[
\sum_{i=0}^{n} \sum_{j=0}^{i} P(i, j) + \sum_{r=1}^{k} \sum_{i=0}^{n-r} P(i, i+r) \quad \forall k = 0, 1, \ldots, n
\]

where \( P(i, j) = \frac{C(N-i-1, n-i) \cdot C(N-j-1, n-j)}{C(N, n)^2} \) \hspace{1cm} (4.1)

and \( N = n + D - 1 \)

Proof: Let \( X \) and \( Y \) be independent identically distributed random variables representing the number of released and requested buffer slots in a round on a node. The probability distribution function (pdf) of \( X \) and \( Y \) is derived as follows. First consider the problem of counting the number of ways in which \( n \) identical streams can be distributed among \( D \) distinct nodes. The number of solutions is equal to the number of \( n \) combinations with repetition allowed from a set with \( D \) elements (stars and bars problem), which is equal to \( C(n + D - 1, n) \) \[95\]. Next, it is required to determine the number of combinations in which there are exactly \( i \) streams on a node. This can be obtained by fixing \( i \) streams on a node, and determining the number of combinations of the remaining streams on the remaining nodes. This is equal to the number of \( n-i \) combinations with repetition allowed from a set with \( D-1 \) elements, which is equal to \( C(n + D - i - 2, n - i) \). Therefore, the pdf of \( X \) and \( Y \) is given by

\[
P(X = iR) = P(Y = iF) = \frac{C(n + D - i - 2, n - i)}{C(n + D - 1, n)}
\]

Since \( X \) and \( Y \) are independent (because a FF/REW stream accesses a node independently of where its reservation existed), their joint pdf is given by the product of their individual pdfs \[96\],

\[
P(X = iR, Y = jF) = \frac{C(n + D - i - 2, n - i) \cdot C(n + D - j - 2, n - j)}{C(n + D - 1, n)^2} \quad \text{(4.2)}
\]
For example, using eqn. (4.2), with \( n = D = 2, i = 2, j = 1 \), \( P(X = 2R, Y = F) = 1/9 \). This is the probability of the occurrence of \( RRF \), as seen in Figure 4-1.

It can be noted from Figure 4-2 that all combinations in which \( X \geq Y \) require no additional buffer. Then the probability that all stream requests can be satisfied with no additional buffer is given by

\[
P(\text{Buffer} = 0) = \sum_{i=0}^{n} \sum_{j=0}^{i} P(i, j) \quad (4.3)
\]

Similarly, the probability that streams require a buffer level of exactly \( kF \) is given by

\[
P(\text{Buffer} = kF) = \sum_{i=0}^{n-k} P(i, i+k) \quad k = 1, \ldots, n \quad (4.4)
\]

For example, with \( k = 1 \), it can be noted from Figure 4-2 that this is the number of occurrences of the combinations \( RRF \) and \( F \) in Figure 4-1. Using eqn. (4.4), it can be verified that \( P(\text{Buffer} = F) = 2/9 \). Combining eqns. (4.3) and (4.4), the probability that all stream requests can be satisfied with a buffer level of \( kF \) is given by

\[
P(\text{Buffer} \leq kF) = \sum_{i=0}^{n} \sum_{j=0}^{i} P(i, j) + \sum_{r=1}^{k} \sum_{i=0}^{n-r} P(i, i+r)
\]

Q.E.D.

**Lemma 1**: Given that \( n \) \( FF/Rew \) streams execute on \( D \) nodes, the probability of satisfying all stream requests with a buffer level of 0 in a round lies in the interval \((0.5, 1)\).
Proof: The symmetrical combinations where $X = Y$ have a buffer requirement 0, since the number of released slots is at least as large as the number of FF/Rew requests. Using eqn. (2), it is seen that the probability of a symmetrical combination is given by

$$P_{\text{sym}}(i,i) = \sum_{i=0}^{n} \frac{C(n+D-i-2,n-i)^2}{C(n+D-1,n)^2}$$

Eqn. (3) can be rewritten as

$$P(\text{Buffer} = 0) = \sum_{i=0}^{n} \sum_{j=0}^{i-1} P(i,j) + P_{\text{sym}}(i,i)$$

Since $P(i,j) = P(j,i)$, it can be easily seen that

$$2 \sum_{i=0}^{n} \sum_{j=0}^{i-1} P(i,j) + P_{\text{sym}}(i,i) = 1$$

(4.5a)

Hence,

$$P(\text{Buffer} = 0) = 0.5 + \frac{P_{\text{sym}}(i,i)}{2}$$

(4.5b)

Expanding the term $P_{\text{sym}}(i,i)$ gives

$$\left[\frac{D-1}{D-1+n}\right]^2 + \left[\frac{D-1}{D-1+(n-1)}\left(1-\frac{D-1}{D-1+n}\right)\right]^2 + \ldots$$

$$+ \left[\frac{D-1}{D-1+(n-n)}\left(1-\frac{D-1}{D-1+n}\right)\left(1-\frac{D-1}{D-1+(n-1)}\right)\ldots\left(1-\frac{D-1}{D-1+(n-(n-1))}\right)\right]^2$$

(4.5c)
When \( n = D \), taking the limit of the first term in the above series gives

\[
\lim_{D \to \infty} \left[ \frac{1}{1 + n/(D-1)} \right]^2 = 1.
\]

All the remaining terms contain the factor

\[
\left( 1 - \frac{1}{1 + n/(D-1)} \right), \text{ where } \lim_{D \to \infty} \left( 1 - \frac{1}{1 + n/(D-1)} \right) = 0.
\]

Hence, for \( n = D \), \( \lim_{D \to \infty} P_{\text{sym}}(i,i) = 1 \) and \( \lim_{D \to \infty} P(\text{Buffer} = 0) = 1 \). \hspace{1cm} (4.6)

Analogously to eqn. (4.5c), expanding \( P_{\text{sym}}(i,i) \) and grouping terms differently gives

\[
\left[ \frac{D-1}{D-1+n} \right]^2 + \left[ \frac{1}{n} \right] \left( \frac{D-1}{1 + \frac{D-1}{n}} \right) \left( \frac{D-1}{1 + \frac{D-2}{n}} \right) \ldots \left( \frac{D-1}{1 + \frac{D-2}{n}} \right) \right]^2 + \ldots
\]

\[
+ \left[ \frac{1}{n} \right] \left( \frac{D-1}{1 + \frac{D-1}{n}} \right) \left( \frac{D-1}{1 + \frac{D-2}{n}} \right) \ldots \left( \frac{D-1}{1 + \frac{D-2}{n}} \right) \right]^2
\]

Suppose that \( D = n \), then \( \lim_{n \to \infty} \left[ \frac{D-1}{D-1+n} \right]^2 = 0 \). All the other terms in the above series contain the factor \( \left( \frac{1}{n} \right) \), where \( \lim_{n \to \infty} \left( \frac{1}{n} \right) = 0 \).

Therefore, for \( D = n \), \( \lim_{n \to \infty} P_{\text{sym}}(i,i) = 0 \) and \( \lim_{n \to \infty} P(\text{Buffer} = 0) = 0.5 \) \hspace{1cm} (4.7)

Therefore, \( P_{\text{sym}}(i,i) \) cannot lie outside \((0,1)\). \hspace{1cm} (4.8)

This proves lemma 1 (using eqns. (4.6), (4.7) and (4.8)). \hspace{1cm} Q.E.D.
A graphical description of Lemma 1 is shown in Figure 4-3.

Figure 4-3. Probability of meeting all stream requests in a round with no additional buffer allocated for FF/Rew requests.

Figure 4-4 shows the probability of meeting all requests in a round when additional buffer slots are provided for the FF/Rew streams on each node. Given $N$ FF/Rew streams, the overload probability $p_{ov}(N,k)$ for not being able to serve the requests of all $N$ streams for a given buffer level in a round is obtained as

$$p_{ov}(N,k) = 1 - P(\text{Buffer} \leq kF)$$
Figure 4-4. Probability of meeting all FF/Rew stream requests in a round with (a) $DF$ buffer slots ($D = \text{number of nodes}$) allocated on each node (b) $NF/4$ buffer slots ($N = \text{number of streams}$) allocated on each node.

For e.g., consider $N = 100$, $D = 16$ and the buffer on each node is $10F$ slots, then $p_{ov}(100,10) = 0.1$. In other words, we can guarantee with a probability of at least $1 - p_{ov}(100,10) = 0.9$ that all FF/Rew $N = 100$ requests in one round can be served. If the goal is to guarantee the timely service of $N$ requests with a probability of at least 0.95, then the value of $N$ must be lowered, or the buffer allocated to the FF/Rew streams must be increased. For $N = 68$ or a buffer level of $14F$ slots, we obtain an overload probability of 0.05, which would achieve the goal. In general, for a given value of $D$, $N$ and $k$, and a given threshold $\delta$, the maximum number of admissible concurrent streams $N_{max}$ can be derived as

$$N_{max} = \max\{N \mid p_{ov}(N,k) \leq \delta\}$$

It is important to note that the above analysis assumes an equiprobable sample space – therefore, fast-forward segments must be short, and evenly
distributed across all nodes. The latter condition can be satisfied easily by selecting the number of nodes and the FF/Rew speed (i.e., number of blocks skipped between two consecutive FF/Rew requests of a stream) to be relatively prime. Otherwise, groups of requests may form repetitive cycles, which may cause severe overload on one or more nodes. It also suggests that a random distribution of data may be preferable to schemes such as round-robin.

If there is insufficient buffer for the stream on some node in a round, the data segment for the stream cannot be retrieved for that round, which results in a glitch or jitter at the client display. The upper bound on the glitch probability for a stream is $p_{ov}(N,k)$, the reason for which can be explained as follows. In an overload round, there may be some released slots available (but not a sufficient number for all streams). It is assumed that the streams that can use these slots are selected independently among the rounds. In the worst-case, an unlucky stream may never be able to grab a released slot in an overload round. Then the probability for one stream suffering $g$ glitches in $R$ rounds is

$$P(\text{stream has } g \text{ glitches in } R \text{ rounds}) = \binom{R}{g} (p_{ov}(N,k))^g (1 - p_{ov}(N,k))^{R-g}$$

This is the binomial distribution, which is computationally expensive to compute for large $R$. However, the following Chernoff bound, derived in [97], can be applied for $g / R > p_{ov}(N,k)$:

$$p_{glitch} = P(\text{number of glitches in stream in } R \text{ rounds } \geq g) \leq \left( \frac{Rp_{ov}(N,k)}{g} \right)^g \left( \frac{R - Rp_{ov}(N,k)}{R - g} \right)^{R-g}$$

(4.9)
The probability that an individual stream suffers more than a given number of glitches can be obtained using eqn. (4.9) for given $R$, $N$ and $k$. For example, with $N = 100$, $kF = 34$, and $D = 16$, the probability of overload in a round $p_{ov} = 0.0012$ (from eqn. (4.1)), and the probability that a stream suffers more than 8 glitches in 1500 rounds (approximately 0.5% of 1500), is $3.2 \times 10^{-3}$. Table 4-1 shows the various values of $p_{ov}$ and $p_{glitch}$ for different $N$ and $kF$ on a system with 16 nodes.

Table 4-1. Values of $p_{ov}$ and $p_{glitch}$ for different $N$, $kF$, $R$ and $g$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$kF$</th>
<th>$R$</th>
<th>$g$</th>
<th>$p_{ov}$</th>
<th>$p_{glitch}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>18</td>
<td>100</td>
<td>2</td>
<td>.0012</td>
<td>$2.3 \times 10^{-2}$</td>
</tr>
<tr>
<td>50</td>
<td>15</td>
<td>50</td>
<td>2</td>
<td>.0038</td>
<td>$5.3 \times 10^{-2}$</td>
</tr>
<tr>
<td>100</td>
<td>34</td>
<td>60</td>
<td>1</td>
<td>.0012</td>
<td>$1.8 \times 10^{-1}$</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>1500</td>
<td>3</td>
<td>.0447</td>
<td>$6.2 \times 10^{-1}$</td>
</tr>
<tr>
<td>100</td>
<td>34</td>
<td>1500</td>
<td>8</td>
<td>.0012</td>
<td>$3.2 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

The model that is presented in this section, for provision of FF/Rew service, is sufficiently general in that it can be used for any parallel server. It can also be extended to support the pause operation. We shall briefly describe here how the model can be adapted to provide FF/Rew/Pause in SAM. Since FF/Rew/Pause change the manner in which resources are consumed, the admission control scheme needs to be modified. When there is no support for interactive operations, the resources are reserved on a round basis, as described in the previous chapter. However, when a stream may perform FF/Rew/Pause, the largest number of blocks retrieved by the minimum and maximum resolution layers of the streams are reserved for each stream in every round.

Let $MBS_{\text{min}}^k = \text{maximum} \ BS_{\text{min}}^k (l - l_{\text{start}}^k), \ \forall l$ of stream $k$. 

77
\( MB_{\text{max}}^k = \text{maximum} \ B_{\text{max}}^k (l - l_{\text{start}}^k), \forall l \text{ of stream } k \)

One scheme by which FF/Rew may be provided is by reserving a fraction of \( N_{\text{det}} \) slots for FF/Rew use. Therefore, if \( \beta N_{\text{det}} \) slots are reserved for FF/Rew streams, eqn. (3.3) for admission control can be modified to

\[
\sum_{k=1}^{K} MB_{\text{min}}^k + MB_{\text{min}}^{K+1} \leq N_{\text{det}} (1 - \alpha - \beta)
\]

\[
\sum_{k=1}^{K} MB_{\text{max}}^k + MB_{\text{max}}^{K+1} \leq N_{\text{stat}}
\]  \hspace{1cm} \text{(4.10)}

Both of the above conditions must be true for stream \( K+1 \) to be admitted. An alternate scheme is to use the \( \alpha N_{\text{det}} \) slots, which are reserved for recovery from node failure, for the FF/Rew service. The advantage of the scheme is that it can admit more streams than those admitted by (4.10), but the FF/Rew service becomes unavailable when a node failure occurs. Eqn. (3.3) for admission control can be modified to

\[
\sum_{k=1}^{K} MB_{\text{min}}^k + MB_{\text{min}}^{K+1} \leq N_{\text{det}} (1 - \alpha), \alpha > \beta
\]

\[
\sum_{k=1}^{K} MB_{\text{max}}^k + MB_{\text{max}}^{K+1} \leq N_{\text{stat}}, \alpha > \beta
\]  \hspace{1cm} \text{(4.11)}

or

\[
\sum_{k=1}^{K} MB_{\text{min}}^k + MB_{\text{min}}^{K+1} \leq N_{\text{det}} (1 - \beta), \beta > \alpha
\]
\[ \sum_{k=1}^{K} MBS_{\text{max}}^k + MBS_{\text{max}}^{K+1} \leq N_{\text{stat}}, \beta > \alpha \] (4.12)

The pause operation can be supported by reserving the resources for an extended period of time, which depends on how long the pause may be. For example, to support pause of 5 minutes on a server with a round duration of 1 second, the resources given by eqns. (4.10)-(4.12) must be reserved for an additional \(5 \times 60 = 300\) rounds. If the total pause time while viewing the movie exceeds 5 minutes, the client stream needs to be readmitted after the reservation time has elapsed.

### 4.5 Model Validation

To validate the analytical model, we compared the predictions of the model with results obtained from detailed simulations. A trace-driven simulation model was developed consisting of several disks and a controller on a network segment, using the CSIM18 package [75]. The model consists of 5 nodes containing 50 videos on a network segment. The IBM Ultrastar18ES is used, which is representative of the type of high-performance disks available commercially. The device characteristics are given in the Appendix. The user can specify different storage policies and a network architecture with an arbitrary number of devices. The user can also specify the number and display rates of the stored video files and supply a trace file containing the frame sizes.

The movie database consisted of 30 movies. The characteristics of 2 sample movies are presented in Table 4-2. Further information on the video characteristics can be found in [93]. The videos are laid out in a random manner across the nodes. During normal play, each stream retrieves one block from successive nodes in consecutive rounds. During FF/Rew with speed \(f\), a block is retrieved after skipping \(f-1\) blocks. The number of nodes in the system is 5, and the FF/Rew speeds are...
randomly selected to be one of 2, 3, 4, 6 and 7. The duration of a FF/Rew clip is randomly selected to be between 10-500 seconds.

Table 4-2. Video traces used in simulation.

<table>
<thead>
<tr>
<th>Movie</th>
<th>Mean bit rate (kbits/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citizen Kane (base layer, 64)</td>
<td>64</td>
</tr>
<tr>
<td>Baseball (base layer, QL:1)</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 4-3 shows the probability of overload in a round on a node, $p_{ov}$, when no additional buffer is allocated for the FF/Rew streams. Thus, for example, the system is configured with 20 buffer slots on each of the 5 nodes, in order to support 100 normal play streams without jitter. When all 100 streams perform interactive operations, $p_{ov}$ is 0.475. For each point on the graph, the simulations were repeated several times with different randomizer seeds, and the largest value of $p_{ov}$ was recorded. It is observed that the results from the analytical model provide a good upper bound on the simulation results.

Table 4-3. Variation of $p_{ov}$ with the number of FF/Rew streams on 5 nodes, and no additional buffer.

<table>
<thead>
<tr>
<th>Number of FF/Rew streams</th>
<th>Analytic results</th>
<th>Simulation results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$p_{ov}$</td>
<td>$P_{ov}$</td>
</tr>
<tr>
<td>1</td>
<td>0.16</td>
<td>0.015</td>
</tr>
<tr>
<td>2</td>
<td>0.24</td>
<td>0.032</td>
</tr>
<tr>
<td>10</td>
<td>0.4083</td>
<td>0.236</td>
</tr>
<tr>
<td>50</td>
<td>0.4782</td>
<td>0.259</td>
</tr>
<tr>
<td>100</td>
<td>0.4888</td>
<td>0.475</td>
</tr>
<tr>
<td>150</td>
<td>0.4925</td>
<td>0.489</td>
</tr>
</tbody>
</table>
Table 4-4 shows the probability that a stream has 2 or more glitches in 10 rounds, although other glitch parameters may be selected. In general, FF/Rew streams have short durations and are more tolerant to glitches than normal play streams. As shown, the analytical model always overestimates the probability of a glitch, but on the other hand, the results are sufficiently accurate to be usable for selecting the number of buffer slots for FF/Rew streams. For example, the results show that with a buffer of 15 slots per node for 25 FF/Rew streams (5 for normal play and 10 for FF/Rew), $p_{ov} = 0.0528$, and the probability of 2 or more glitches in 10 rounds is reduced to $p_{glitch}$ (analytic) = 0.09. Therefore, selecting the buffer to be 15 slots per node results in a 60% saving over the deterministic case, where 25 buffer slots per node are needed to support 25 FF/Rew streams without any glitches.

Table 4-4. Variation of $p_{glitch}$ with additional buffer slots for 25 FF/Rew streams on 5 nodes.

<table>
<thead>
<tr>
<th>Additional buffer</th>
<th>$p_{ov}$</th>
<th>$p_{glitch}$ (R = 10, g = 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Analytic</td>
</tr>
<tr>
<td>0</td>
<td>0.458</td>
<td>≈ 1</td>
</tr>
<tr>
<td>8</td>
<td>0.09</td>
<td>0.225</td>
</tr>
<tr>
<td>10</td>
<td>0.0528</td>
<td>0.09</td>
</tr>
<tr>
<td>15</td>
<td>0.0099</td>
<td>0.004</td>
</tr>
<tr>
<td>20</td>
<td>0.0007</td>
<td>0.017</td>
</tr>
</tbody>
</table>

4.6 Summary

It is shown that the statistical multiplexing of available resources can reduce the resources required to support interactive service on a parallel server. An analytical model is developed to predict the probability that stream requirements can be met for a given node buffer level. The results show that the proposed scheme can reduce buffer requirements significantly for a small amount of tolerable client glitch.
An upper bound on client glitches is also derived. The analytical model is validated using simulations.
Chapter 5

Matrix Multiplication With Distributed Arithmetic

5.1 Background

In the previous chapters, we discussed methods for the storage and retrieval of video. In Chapters 5 – 7, we develop new designs for processing video. Matrix multiplication operation is common in many image processing and signal processing environments. The parallel multiplier has been a basic building block for many algorithms. Many high performance algorithms and architectures have been proposed to accelerate multiplication. Two different methods of performing matrix multiplication are based on Distributed Arithmetic and systolic arrays.

Distributed Arithmetic (DA) [70,71] provides an approach for multiplier-less implementation of DSP systems. It is an algorithm that can perform multiplication with lookup table (LUT) based schemes (also called DALUT). DA specifically targets the sum of products (also referred to as the vector dot product) computation that is found in many of the important DSP filtering and frequency transforming functions. With improvements in capacity and performance and a decrease in cost, FPGAs have become a viable solution for making custom chips and programmable DSP devices. By mapping algorithms to FPGAs significant performance benefits can be achieved. Combined with Xilinx FPGA lookup table architecture, the DA algorithm was shown to produce very efficient filter designs [98].
Techniques for speeding up DALUT based applications have appeared in several papers, such as using matrix factorization methods and exploiting symmetry in matrices to speed up the computation procedure by reducing the number of addition and subtraction operations required. Image and signal processing techniques using DA have been discussed in [99,100,101,102] and others.

In this chapter, we present a novel architecture for speeding up DALUT applications. The distinguishing feature of the new design is that, unlike in traditional DALUT architectures, the carry chain delay is reduced or eliminated from the critical path. In the proposed scheme, the individual bits of a word do not have to be processed as a unit. Instead, the current iteration can start as soon as the LSB of the previous iteration is available, without waiting for the whole word from the previous computation. Designs in which an $n$-bit carry chain, where $n$ is the word length, is broken into smaller $r$-bit chains, $1 \leq r < n$, are described. Techniques to eliminate carry chain delay have appeared in other papers [11,103,104]. We apply a similar idea to DALUT-based structures on FPGAs to gain performance benefits. New designs for parallel and serial DALUT structures are proposed. A cost-performance analysis of the designs shows that the proposed designs with no carry chain delay have a lower cost-performance ratio than traditional DALUT designs as well as designs with $r$-bit carry chains. Also, it is shown that designs with 8-bit carry chains provide a good compromise between performance and cost. The designs are implemented on a Xilinx XC4028XL-3-BG256 using Xilinx Foundation tools v 3.1i. The results show that the new designs achieve speedup by a factor of at least 1.5 over traditional designs. The technique is general enough to be applicable to other families of chips as well.
5.2 Traditional DALUT architecture

A brief overview of traditional DALUT architecture is presented in this section. Let the input data and the transformed data be represented by two vectors \( X \) and \( Y \) of size \( N \), respectively. Then \( Y \) can be written as follows:

\[
Y = \sum_{k=0}^{N-1} A_k x_k
\]  

(5.1)

where \( A_k \) are constant coefficients. \( x_k \) is written in weighted format as shown in equation (5.2).

\[
x_k = -x_{k,n-1} + \sum_{m=1}^{n-1} x_{k,n-1-m} 2^{-m}
\]  

(5.2)

where \( x_{k,m} \) is the \( m \)th bit of \( x_k \) (which can be a zero or one), \( x_{k,n-1} \) is the sign bit and \( n \) is the word size. Substituting equation (5.2) in (5.1),

\[
Y = \sum_{k=0}^{N-1} A_k \left( -x_{k,n-1} + \sum_{m=1}^{n-1} x_{k,n-1-m} 2^{-m} \right)
\]

\[
= -\sum_{k=0}^{N-1} A_k x_{k,n-1} + \sum_{m=1}^{n-1} \left( \sum_{k=0}^{N-1} A_k x_{k,n-1-m} \right) 2^{-m}
\]

Define \( Z_{n-1-m} = \sum_{k=0}^{N-1} A_k x_{k,n-1-m} \) (\( m \neq 0 \))

and \( Z_{n-1} = -\sum_{k=0}^{N-1} A_k x_{k,n-1} \) (\( m = 0 \))

The output result is given by

\[
Y = \sum_{m=0}^{n-1} Z_{n-1-m} 2^{-m}
\]  

(5.3)
Let \( c = n - 1 - m \) for convenience. Since \( Z_c \) is a function of \( x_{k,c} \), \( k = 0, N - 1 \), it has only \( 2^N \) possible values, which can be precomputed and stored in a LUT.

Figure 5-1 shows the traditional LUT-based serial DA design for a four-product \( (N = 4) \) multiplier-accumulator (MAC), which multiplies four pairs of numbers and sums the results. The LUT data is composed of all partial sums of the coefficients \( (A_0, A_1, A_2 \text{ and } A_3) \). The LSB (output from each parallel to serial converter (PSC) register) of the four data samples \( (x_0, x_1, x_2 \text{ and } x_3) \) addresses the LUT. The output of the LUT depends on the address formed by the combinations of these input bits. For example, if all four data bits are 1, then the output from the LUT is the sum of all four coefficients. Further details on DA can be found in [71,98]. In the next section, new designs for serial and parallel DALUT structures are presented.

![Figure 5-1. Traditional LUT-based serial DA for a four-product MAC.](image-url)
5.3 Proposed Designs

In Figure 5-1, the carry chain is present in the shift-accumulator. Figure 5-2 shows how the carry chain can be removed from the critical path in this design. The critical path is the delay between points A and B in both designs. The accumulator is composed of a series of bit-level processing elements (PEs). Each bit-level PE takes three input bits that are added to produce a sum bit and a carry bit that are registered in two flip-flops (FF) at each clock. Figure 5-2 shows a PE configured as a serial adder. The output of the LUT is connected to input \( I_1 \) of the PEs. The partial sum bit in the \( j \)th PE is connected to the input \( I_2 \) of the \((j-1)\)th PE. The carry bit \( C \), generated from adding three bits in each PE, is not propagated across the PEs; instead, it is registered within the PE at each clock, and added to the input bits in the next clock. One bit of final product is shifted out serially from the bit-level shift-accumulator at each clock into a set of shift registers. After the \( n \)th clock, where \( n \) is the word size, the residual partial sum and carry bits are summed in a carry-propagate adder to form the \( n-1 \) higher-order bits of the final product, which has \( 2^{n-1} \) bits.

The residual partial sum and carry bits are the bits left over in the PEs after \( n \) clocks, when no new outputs from the LUT are required to be added. When the sign bits arrive, a subtraction instead of an addition is done in the shift-accumulator. The bits of \( Z_e \) are inverted in the last cycle by setting \( INV \) to ‘1’ (corresponding to multiplication by the sign bit); however, the addition of a ‘1’ is delayed (in order to move the carry-propagate adder out of the critical path), and takes place, after the last clock cycle, by setting the \( IO \) bit to a ‘1’. The \( IO \) bit is called a compensating one, since it adjusts the final result to the correct value by compensating for the addition of ‘1’ that was omitted earlier in the computation. The resulting design, in Figure 5-2, takes the same number of clocks as the design in Figure 5-1, but runs much faster because the carry propagate adder has been moved out of the critical
path. The clock time of the design in Figure 5-1 has to be long enough to accommodate the carry chain; the clock time of the design in Figure 5-2 is the time needed to generate the partial carry and sum.

Figure 5-2. Proposed (1-bit) serial DA one-LUT design for a four-product MAC.

Figure 5-3 shows how this can be extended to the $r$-bit case, by using PEs with carry chain lengths of $r$ bits. The last cell in the $r$-bit PE sums 4 bits and generates two carries – $C$ (lower order carry) and $C_1$ (higher order carry). All the other cells are similar to the bit-level PE. The carries, $C$, between the cells inside the $r$-bit PE are not registered; instead, they propagate across the $r$-bit PE, with the exception of the two carries in the last cell. Thus, the carry chain is broken across PEs. Note that the last PE may have less than $r$ cells, depending on whether $n$ is a
multiple of $r$. The above scheme is extended to the multiple-product case. As the number of coefficients increases, the size of the LUT grows exponentially.

Figure 5-3. Proposed ($r$-bit) serial DA one-LUT design for a four-product MAC.

Figure 5-4. Traditional serial DA two LUT-based design for an eight-product MAC.
A large LUT is avoided by partitioning the circuit into smaller groups and combining the LUT outputs with adders, as shown in Figure 5-4. The adders are less costly than the larger LUT. However, each adder introduces a carry chain delay in the critical path.

Figure 5-5 shows how to remove the carry chain for an eight-product MAC, implemented with two LUTs, by using a simple bit-level adder array. The adder array is composed of bit-level PEs shown in Figure 5-2. The partial sum output bit of each PE in the bit-level adder array is connected to a PE input in the bit-level shift-accumulator. The partial sums and carries are stored separately in this adder array, and the carries are not propagated across the PEs of the adder array. When the
sign bits are input to the LUT, the outputs of both LUTs must be inverted and a ‘1’ added to each of the outputs. Instead, as in the one-LUT case, the addition of the two ‘1’s is delayed and takes place after the last clock, in the carry-propagate adder. The design in Figure 5-5 takes one extra clock compared to the traditional serial DA for two-LUT case so that the residual carries in the bit-level adder array are added in the bit-level shift-accumulator. After the last clock, the partial sums and carries in the shift-accumulator are added in the carry-propagate adder to form the final product. Again, the clock cycle time is reduced since the carry chains are removed from the critical path, at the cost of a single extra clock. To compensate for the two ‘1’s required to be added earlier, a suitably weighted compensating one, (i.e., in this case a ‘10’ string) is added in the carry-propagate adder at the appropriate position (by setting IO to a ‘1’ after the last clock as in Figure 5-5). The above idea can be extended similarly for serial DA (SDA) many-product cases and parallel DA.

Figure 5-6 shows the traditional LUT-based scheme for 2-bit parallel DA. Two LUTs are needed, one for the even order coefficients and the other for the odd order coefficients.

Figure 5-6. Traditional 2-bit parallel DA for a four product MAC.
Figure 5-7 shows the scheme for proposed 2-bit parallel DA for a four product MAC. The bit-level 2-bit scaling adder array and accumulator are used here, and there is no carry propagation across these components. The carry out $C$ from the $j$th PE is registered before it is sent to the $(j-1)$th PE, where it is added to input bits of the same weight. Unlike in SDA designs, the carryout in the bit-level PDA PE is not fed back into the same PE where it was generated. The bit-level 2-bit scaling accumulator shifts two bits at a time. As in the previous designs, carry propagation takes place after the last clock in the two carry propagate adders shown in Figure 5-7. The $IO$ bit is set to a “1” after the last clock to compensate for the “1”
that was required to be added earlier. The \( r \)-bit designs can be obtained directly by replacing \( r \) bit-level PEs with one \( r \)-bit PE in Figure 5-5 and Figure 5-7.

### 5.4 Cost and performance analysis

Gate count and gate-delay unit models used in VLSI designs are not useful for evaluating costs and performance of FPGA designs. In our study, instead of gate counts, cost is measured as the number of configurable logic blocks (CLBs) used. Performance is calculated as the inverse of the computation time (time to complete the DA operation). The computation time is estimated using a simple model that estimates the total delay in the critical path as a sum of the logic and routing delays. The number of logic levels in the design determines the logic delay, and the routing delay is estimated to be 50-100\% of the logic delay. The actual values of the computation times are then obtained from Xilinx timing-simulation software, instead of from the gate-delay models used for VLSI designs. The cost-performance ratio (CT) is calculated as cost multiplied by the computation time. In making comparisons, a DA design with a lower CT ratio has better performance.

Table 5-1 shows the CLB count and logic delays for various components in the traditional and proposed designs as a function of the word size \( n \), on the XC4000 chip. In the XC4000 series ripple-carry outputs are routed between CLBs on high-speed dedicated paths. This dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. In adders using XC4000 dedicated carry logic, delay estimation is possible [105]. The carry path in an adder uses dedicated interconnects between CLBs. These interconnects introduce a fixed delay, even when the carry passes from one CLB column to the next at the top or bottom of the array. This permits the routing delay to be incorporated into the CLB specifications published in the data book. Consequently, the propagation delay through an adder can be calculated directly from the data book specifications.
Table 5-1. CLB counts and logic delays in various components, as a function of word size \( n \), on the XC4000 series.

<table>
<thead>
<tr>
<th>Component</th>
<th>CLB count (C)</th>
<th>Delays (T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )-bit full adder (non-registered)</td>
<td>( C_{FA}(n) = \frac{n}{2} + 2 )</td>
<td>( T_{FA}(n) = T_{INCY} + \frac{(n-2)}{2} T_{BRP} + T_{SUM} )</td>
</tr>
<tr>
<td>( n )-bit shift accumulator (registered)</td>
<td>( C_{ACC}(n) = \frac{n}{2} + 2 )</td>
<td>( T_{ACC}(n) = T_{INCY} + \frac{(n-2)}{2} T_{BRP} + T_{SUMC} + T_{CK} )</td>
</tr>
<tr>
<td>( n )-bit invertor</td>
<td>( C_{INV}(n) = \frac{n}{2} )</td>
<td>( T_{INV} = T_{ILO} )</td>
</tr>
<tr>
<td>( n )-bit ROM-based LUT (depth ( \leq 16 ))</td>
<td>( C_{LUT}(n) = \frac{n}{2} )</td>
<td>( T_{LUT} = T_{ILP} )</td>
</tr>
<tr>
<td>( n )-bit Shift register</td>
<td>( C_{SR}(n) = C_{PSC}(n) = \frac{n}{2} )</td>
<td>( T_{PSC} = T_{CKO} )</td>
</tr>
<tr>
<td>1-bit Serial adder</td>
<td>1</td>
<td>( T_{SA} = T_{CK} )</td>
</tr>
</tbody>
</table>

The values of the delay parameters for the XC4000 (-3) family (shown in Table 5-2) are obtained from the Xilinx data sheet [106]. The total computation time to calculate the result is given by the product of the number of clocks and the clock cycle time. In Figure 5-1, the critical path is the delay through points A and B. The total delay is the sum of the logic delays from the output of the shift register, through the LUT, inverter and the \((n+2)\)-bit shift accumulator including the routing delay between these components. The cost is the sum of the number of CLBs in the shift registers, LUT, inverter and the \((n+2)\)-bit (signed) shift accumulator.
The computation time of the one LUT-based traditional design (Figure 5-1) is

\[ (n+1) \times (T_{PSC} + T_{LUT} + T_{INV} + T_{ACC}(n+2) + T_R) \]

\[ = (n+1) \times (T_{CKO} + 2T_{LIO} + T_{INCY} + \frac{n}{2}T_{BYP} + T_{SUMC} + T_{ICK} + T_R) \]

where \( T_R \) is the total routing delay between components in the critical path.

The cost of the traditional design (Figure 5-1) is

\[ = C_{LUT}(n+2) + C_{ACC}(n+2) + C_{SR}(n) + C_{PSC}(n) \times 4 \]

\[ ^1 \text{Obtained from the Xilinx timing analyzer.} \]
Substituting the values of the parameters from Table 5-1 and Table 5-2 gives the total computation time to be \((n+1)(10.23 + 0.13 n + T_R^r)\) and the cost to be \(3.5n + 4\).

Therefore, the cost-performance metric of the traditional design (Figure 5-1) is

\[
= (3.5n + 4)(n + 1)(10.23 + 0.13 n + T_R^r).
\]  \(5.4\)

The computation time of the one LUT-based proposed design (Figure 5-2) is

\[
(n+1) \times (T_{\text{PSC}} + T_{\text{LUT}} + T_{\text{INV}} + T_{\text{SA}} + T_{\text{R}}) + T_{\text{FA}}(n) = (n+1) \times (T_{\text{CKO}} + 2T_{\text{ILO}} + T_{\text{IC}} + T_{\text{R}}) + T_{\text{INCY}} + \frac{(n-2)}{2} T_{\text{BYP}} + T_{\text{SUM}}.
\]

The cost of the proposed design (Figure 5-2) is

\[
= C_{\text{LUT}}(n+2) + (n+2) \times C_{\text{SA}}(1) + C_{\text{INV}}(n+2) + C_{\text{SR}}(n) + C_{\text{PSC}}(n) \times 4
\]

Substituting the values of the parameters from Table 5-1 and Table 5-2 gives the total computation time to be \(6.53n + 10.94 + (n+1)T_R^r\) and the cost to be \(5n + 6\).

The cost-performance metric of the one LUT-based design (Figure 5-2) is

\[
= (5n + 6)(6.53n+10.94+(n+1)T_R^r).
\]  \(5.5\)

The cost and computation time of these designs with \(r\)-bit wide carry chains, where \(1 < r < n\), can be calculated similarly.

The cost-performance metrics are plotted for different values of \(r\), in the range \(1 \leq r \leq n\), for different values of \(n\) and different routing delays in Figure 5-9. The traditional design \((r = n)\) has the lowest cost (Figure 5-9(a)), but a higher computation time (Figure 5-9(b)) than the proposed designs. The proposed design \((r = 1)\) has the fastest computation time and the best cost-performance metric (Figure 5-9(c) and Figure 5-9(d)). The design with \(r = 8\) provides a good compromise between cost and performance.
Figure 5-9. Cost-performance analysis of SDA for a four product MAC with different word sizes and carry chain lengths (a) cost vs. word size (b) performance vs. word size (0 routing delay assumed)
Figure 5-9 continued. Cost-performance ratio vs. word size of SDA for a four product MAC with different word sizes and carry chain lengths (c) 0 routing delay (d) routing delay = logic delay
Figure 5-9(c) and Figure 5-9(d) show the variation in the cost-performance curves with zero routing delay and routing delay equal to the logic delay, respectively. The performance improvement in designs with \( r < n \) becomes more pronounced as the word size increases. The analysis given above can be extended similarly to other cases such as PDA and SDA with more than one lookup table. For lack of space we do not present other cases, but in general, the proposed designs outperform the traditional designs.

The bit-level designs are implemented on an XC4000xl chip, and compared with traditional DA designs. The implementation results are presented in the next section. It will be shown that the results of the simulation are consistent with the above discussion.

5.5 Implementation results

The designs described in the earlier section were coded using Verilog HDL and implemented on a Xilinx XC4820-4- BG256 chip using Xilinx Foundation Series 3.1i. The accumulator and adder for traditional designs were implemented from the Core Generator library, and contain dedicated carry logic. The designs presented were verified through extensive simulations. Constraints were entered in the user constraints file (ucf). After the place-and-route (PAR) tool completed the placement and routing, the timing for the design was obtained using the Timing Analyzer. The resources required for each design are given in terms of the total number of CLBs (Configurable Logic Blocks) that are used. The word size used for all the designs is 32 bits.

Table 5-3 compares the resources and computation time required by the traditional and proposed one-LUT-based serial DA designs for four-product MAC. The overall path delay has two components: delay through the logic gates (logic delay) and the route delay. After the designs were placed and routed by the place-
and-route tool, they were examined using the Timing Analyzer tool. The tool showed that the traditional design could be clocked at a period of 27ns, of which 13.4ns is logic delay and 13.6ns is routing delay. The logic delays and the number of CLBs used correspond closely with those predicted by the model in the previous section. The traditional design has a larger route delay because of the additional overhead of routing delay between the stages in a 32-bit accumulator or adder. The proposed design has a period of 15ns, of which 6.4ns is logic delay and 8.6ns is routing delay. This design has fewer logic levels, since there is no carry propagation, than the traditional design, resulting in lower logic delay. However, it consumes more resources because extra registers are required to hold the carry bits in the adder and accumulator, as shown earlier.

Table 5-3. One LUT-based serial DA design for four-product MAC.

<table>
<thead>
<tr>
<th>Design</th>
<th>CLBs</th>
<th>Clock cycles</th>
<th>Logic delay (ns)</th>
<th>Route delay (ns)</th>
<th>Computation Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>114</td>
<td>33</td>
<td>13.4</td>
<td>13.6</td>
<td>891.0</td>
</tr>
<tr>
<td>Proposed (1-bit)</td>
<td>165</td>
<td>33</td>
<td>6.4</td>
<td>8.6</td>
<td>527.0</td>
</tr>
</tbody>
</table>

Table 5-4. Two LUT-based serial DA design for eight-product MAC.

<table>
<thead>
<tr>
<th>Design</th>
<th>CLBs</th>
<th>Clock cycles</th>
<th>Logic delay (ns)</th>
<th>Route delay (ns)</th>
<th>Computation Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>231</td>
<td>34</td>
<td>13.2</td>
<td>13.3</td>
<td>902.3</td>
</tr>
<tr>
<td>Proposed (1-bit)</td>
<td>314</td>
<td>35</td>
<td>6.4</td>
<td>9.6</td>
<td>589.4</td>
</tr>
</tbody>
</table>
Table 5-4 shows the results for serial DA for eight-product MAC. This uses more CLBs than the serial DA with one LUT because of four extra PSC shift registers and a lookup table. However, the logic and routing delays are almost the same as in the previous case. The final addition introduces a combinational delay of 29.4 ns, through the adder, before the final result is available in the proposed design. The traditional design has a larger routing delay because of the additional routing overhead between stages. As the two tables show, the (1-bit) proposed designs run faster by a factor of over 1.5 over traditional design.

In Table 5-5 the traditional and proposed designs for 2-bit parallel DA for four-product MAC are compared. As before, the proposed design with carry chain length of 1 bit has a smaller computation time than traditional.

<table>
<thead>
<tr>
<th>Design</th>
<th>CLBs</th>
<th>Clock cycles</th>
<th>Logic delay (ns)</th>
<th>Route delay (ns)</th>
<th>Computation Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>151</td>
<td>17</td>
<td>13.2</td>
<td>13.5</td>
<td>480.6</td>
</tr>
<tr>
<td>Proposed (1-bit)</td>
<td>270</td>
<td>18</td>
<td>6.4</td>
<td>9.6</td>
<td>346.8</td>
</tr>
</tbody>
</table>

5.6 Summary

In this chapter, novel designs for serial and parallel DALUT and accumulator structures are proposed that reduce or eliminate the carry chain delay from the critical path. A cost-performance analysis of the SDA for a four-product MAC is performed. Instead of using gate counts and gate delays as in VLSI, the cost is measured as the CLB count and the performance measured as the inverse of the total computation time (dependent on the number of CLB logic levels and routing delays) to complete the DA operation. The cost-performance analysis shows that proposed
$r$-bit designs, $r < n$, have a better performance than traditional DA designs on the XC4000 series. The 1-bit designs have the best performance, but a high cost. The 8-bit designs provide a good compromise between cost and performance. Other families of chips can also be analyzed similarly. The designs are implemented using Xilinx Foundation series 3.1i on a Xilinx XC4028-3-BG256 chip. The results show that the new designs proposed show speedup by a factor of at least 1.5 over traditional DA designs. This can pave the way for faster DSP chips, since DA is used in many signal and image processing applications. We shall use the work on the DA architecture described here in DCT in Chapter 7.
Chapter 6

Systolic-array based matrix multiplication

6.1 Background

In this chapter, we present a novel design for a bit-level matrix multiplier. In a bit-level architecture speedup is possible because individual bits of a word do not have to be processed as a unit. Thus, the carry propagation delay is eliminated from the critical path. For example, in matrix multiplication, the carry propagate chain of multiplying two numbers can be broken by sending the partial sums and carries of the product to the accumulating operation, instead of the whole finished word. Consider matrix multiplication $W = X \cdot Y$ where $X$ is $q \times r$, $Y$ is $r \times s$ and word length is $p$. Each element of $W$ is a sum of $r$ intermediate products, i.e.,

$$w_{h,j} = \sum_{j=1}^{r} x_{h,j} y_{j,j} = X_{h} \cdot Y_{j}.$$  

In the proposed design, the least significant bits of intermediate products ($x_{h,j} y_{j,j}, j = 1, ..., r$) can be added before the final result is fully computed. The carries of each of the first $r-1$ intermediate products ($x_{h,j} y_{j,j}, j = 1, ..., r-1$) are not propagated in the last step of multiplication; instead, they are sent to the next product for accumulation until the last multiplication. In contrast, in a word-level matrix multiplier, a product of two words has to be computed completely before it can be added to the next word-level product. Bit-level arrays can exploit parallelism at the bit level that is impossible in word-level processor arrays.
It is proven that bit-level design is $O(\log p)$ times faster than the corresponding word-level architecture for matrix multiplication, where $p$ is the word length. A word-level processor array for $r \times r$ matrix multiplication can have either $O(r)$ processing elements (PEs) with total execution time $T_c = O(r^2)$, or $O(r^2)$ PEs with $T_c = O(r)$. Each word-level PE can use carry-save adders to implement word-level multiplication, which requires $O(p \log p)$ space and $O(\log p)$ execution time [107]. On the whole, a word-level processor array may need either $O(rp \log p)$ space to get $O(r^2 \log p)$ execution time, or $O(r^2 p \log p)$ space to get $O(r \log p)$ execution time. Using a design method developed in [108,109] we have designed the bit-level matrix multiplier, which is presented in this chapter, with an execution time of $O(r)$ (see Section 6.4 for details). Hence, the fastest bit-level design is $O(\log p)$ faster than the fastest word-level design.

In [110] a novel approach for bit-level architecture design is presented. The first step in their approach is to derive the dependence structure as a function of corresponding word-level dependence structure, the dependence structures of arithmetic algorithms implementing word-wise operations and algorithm expansions. A dependence structure is a graph representing the algorithm by one node per operation and one edge between dependent operations. The second step is to find a mapping of the algorithm into a processor array with some optimization criterion, often the total execution time. They illustrate their design using matrix multiplication as an example. In [11,12] a new design is presented that achieves a speedup by a factor of two over the design in [110]. However, both of these designs handle unsigned multiplication only. In [111] designs for bit-level matrix multiplication are presented, but these are slower than the one presented here.

In this chapter, we present a unique bit-level design to perform fixed-point two’s complement matrix multiplication. The distinguishing feature of this work
from our previous work in [11,12] is that two’s complement numbers can be handled as well. The Baugh-Wooley method [112] is used in our design. The bit-level dependence structures, space and time mappings of the algorithm into a processor array are presented. The time optimality and conflict-free properties of the design are proven.

6.2 Matrix Multiplier Design

In this section, the basic design idea of the bit-level two’s complement matrix multiplier is presented. Throughout this chapter, sets, matrices and row vectors are denoted by capital letters; column vectors are represented by lower-case symbols with an overbar; scalars are shown as lower-case letters. The transpose of a vector $\vec{v}$ is denoted as $\vec{v}^T$. Vector $\vec{0}$ denotes a row or column vector whose entries are all zeroes. The dimension of vector $\vec{0}$ and whether it denotes a row or column vector are implied in the context in which it is used. Symbol 0 denotes a matrix whose entries are all zero. For two vectors $\vec{v}$ and $\vec{u}$, $\vec{v} \geq \vec{u}$ means that every component of $\vec{v}$ is greater than or equal to the corresponding component of $\vec{u}$. The set of integers is denoted as $\mathbb{Z}$. A partial product bit is a product of a digit of the multiplicand and a digit of the multiplier. Two or more input bits are summed in each cycle to generate a partial sum bit and one or more carry bits, where each input bit may be a partial product bit, partial sum bit or carry bit.

6.2.1 Background

Multiplication of two words can be divided into three steps: 1) generating partial products, 2) summing up all partial products until only two rows remain, and 3) adding the remaining two rows of partial products by using a carry propagation adder. In a word-level matrix multiplier all three steps must be completed to form an intermediate product before it can be accumulated to the final matrix product.
In the first step, two methods are commonly used to generate partial products. The first method generates the partial product directly by using a 2-input AND gate. The second one uses radix-4 modified Booth encoding (MBE) to generate the partial products [113]. After generating the partial products, a partial product reduction tree is used to sum up all the partial products efficiently. The Wallace tree and the Carry-save tree were proposed to solve this problem [114,115].

In the last step, to generate the product in two’s complement format, a fast carry-propagation adder is required to add the final two rows of partial products from the partial product reduction tree.

Several techniques have been developed to eliminate or reduce the final adder delay, such as carry-lookahead adders, left-to-right-carry-free algorithm [116] and hybrid adder structure [117]. Carry-lookahead adders are not practical for any but the smallest adders due to the rapid increase in the fan-out and fan-in requirements as the adder size increases [107]. For large adders, multi-level structures of gates with limited fan-in may be used as a solution to the fan-in problem, and buffers may be used to obtain the required fan-out, but this involves additional delay – in general the operational time will be proportional to \( \log n \), rather than a constant. The left-to-right-carry-free algorithm proposed in [116] requires \( n \)-level conversions to generate \( n \)-bit MSB products. The hybrid adder structure, which consists of ripple-carry adder, carry-skip adder, and conditional-sum adder blocks was proposed in [117]. However, their empirical methodology is not general enough and requires many trials to determine the final adder partition boundary for different sizes of multiplier, thus increasing the design effort.

In the bit-level matrix multiplier proposed in this chapter, step 3, which requires a carry-propagation adder for the intermediate products, is eliminated. The products are not computed fully; instead, the carries are forwarded to the next intermediate product. The advantage is that the carry-chain can be eliminated in all
steps (except the last one) without requiring any specialized hardware. This greatly speeds up the computation process since the carry chain in the last step does not lie in the critical path that determines the multiplier speed.

We use the Baugh-Wooley method [112] to implement multiplication of two words. The Baugh-Wooley method is simplified by assuming that the multiplier and multiplicand have equal word length $p$. Let the multiplier be $x^{p-1}x^{p-2}...x^1x^0$ (where $x^0$ is the least significant bit and $x^{p-1}$ is the most significant bit), and the multiplicand be $y^{p-1}y^{p-2}..y^jy^0$; then the partial-product bits $x^{p-1}y^j$ and $y^{p-1}x^j$ where $j < p-1$ are inverted. In addition two “1”s are added at positions $2p-1$ and $p$ of the product $s_{2p-1}s_{2p-2}...s_1s_0$. The remaining partial-product bits are generated as in unsigned multiplication.

6.2.2 Example

Figure 6-1(a) shows the modified Baugh-Wooley method for multiplying two words $x_{11} = x_{11}^2x_{11}^1x_{11}^0$ and $y_{11} = y_{11}^2y_{11}^1y_{11}^0$ in two’s complement form with word length $p = 3$. Figure 6-1(b) presents a conceptual view of bit-level addition of partial-product bits generated from $x_{11}y_{11}$ and $x_{12}y_{21}$. The partial product bits within circles are added first, and the carry generated is forwarded to the next cycle as in a carry-save adder. A partial-sum bit is added to another partial sum and carry bit generated in the previous cycle, with proper alignment, to generate a new partial sum and carry bit. The intermediate product $x_{11}y_{11}$ is represented by the partial sum and carry; the carry bits are not propagated to form the complete word. Instead, the carries are forwarded to the next cycle in each case. After the last iteration, when all partial product bits have been added, a carry-propagate adder sums the remaining carries. Thus the carry chain is broken in all steps except the last. Since the carry-chain does not lie in the critical path determining the multiplier speed, our bit-level design can run faster than the corresponding word-level design where the carries are
propagated in each intermediate product. This scheme is extended similarly to multiply larger matrices. It should be noted that the concept is applicable only if the multiplier and multiplicand have the same word length.

Example 1: Figure 6-2 shows the layout and the execution of the PEs at each clock when one row of matrix $X ( [x_{11}^2 x_{11}^1 x_{11}^0 \ x_{12}^2 x_{12}^1 x_{12}^0 \ x_{13}^2 x_{13}^1 x_{13}^0 \ x_{14}^2 x_{14}^1 x_{14}^0 ] )$ is multiplied with one column of matrix $Y ( [y_{11}^2 y_{11}^1 y_{11}^0 \ y_{21}^2 y_{21}^1 y_{21}^0 \ y_{31}^2 y_{31}^1 y_{31}^0 \ y_{41}^2 y_{41}^1 y_{41}^0 ] )$, with word length $p = 3$ and $r = 4$. A processor array with $p \times r$ or $p \times (r+1)$ PEs, depending
on whether $r$ is odd or even respectively, is required to generate one word of the product matrix. There are three main types of PEs, cells A, B and C in the array. These cells are described in greater detail in Section 6.5. The small dot indicates a registered output.

![Diagram](image)

Figure 6-2. Processor array structure to generate one word $w_{11}$ of product matrix $W$ for $r = 4$ and $p = 3$.

It can be seen in Figure 6-2 that the carry-propagate chain exists only in the carry-propagate adder, which does not lie in the critical path that determines the execution speed. The data elements of $Y$ may be pre-stored into the registers of the PEs, for matrix-vector multiplication, prior to starting the execution. Each PE adds two to five bits and contains a few one-bit clocked registers to store one bit of $Y$, a partial-sum bit and one or two carry bits. The carries (higher-order) and partial sums
are pipelined between adjacent PEs, and the data elements of matrix $X$ are
broadcast to the various PEs. The carries (lower-order) are stored in the same PE
and added to the partial-product bit of higher weight that is input in the next cycle.

Table 6-1 shows the computation at each PE in different cycles. Each PE
$pe_{i,j}^t$ adds a few bits to generate, at time $t$, a partial-sum bit represented by
$pe_{i,j}^t$ and one or two carry bits, $c_{i,j}^t$ (lower-order carry) and $C_{i,j}^t$ (higher-order
carry). For example, the input bits to $pe_{2,1}$ in cycle 3 are: $\overline{x_{12}y_{21}}$, output from $pe_{1,1}$
in cycle 2, constant 1, and carry bit from $pe_{2,1}$ in cycle 2. In cycle 3, $pe_{2,1}$ adds
these four bits and generates one sum bit ($pe_{2,1}^3$) and two carry bits ($c_{2,1}^3$ and $C_{2,1}^3$).
The lower bits of the final product are shifted out of $pe_{3,3}$ into shift registers (not
shown). Consider the output of $pe_{3,3}$ at time 3, which is the sum of bits $pe_{2,3}^2$ and
$pe_{4,3}^2$. The partial-sum bit $pe_{3,3}^3$ is the least significant bit of the final product.

The outputs for the PEs with indices $pe_{4,x}^t$ and $pe_{5,x}^t$, $x=1,2,3$, can be
derived similarly as in Table 6-1(a). The values of the input $x$ vectors and the stored
$y$ bits differ from those shown in the table. We can obtain the outputs for the PEs
by making the following substitutions in Table 6-1(a): replace $x_{11}$ with $x_{14}$, $x_{12}$ with
$x_{13}$, $y_{11}$ with $y_{41}$ and $y_{21}$ with $y_{31}$. Also, replace $pe_{1,x}$ with $pe_{5,x}$, $pe_{2,x}$ with $pe_{4,x}$,
$c_{1,x}$ with $c_{5,x}$, $c_{2,x}$ with $c_{4,x}$ and $C_{2,x}$ with $C_{4,x}$, $x=1,2,3$.

The bits of the final product are given by the partial-sum bits $pe_{3,3}^t$ (in the
third column of Table 6-1(b)): $\{s_p, s_p, pe_{3,3}^5, pe_{3,3}^5, pe_{3,3}^1, pe_{3,3}^3, pe_{3,3}^3\}$. The lower bits of
the product are shifted out serially from the PE $pe_{3,3}$ (from times 3 to 6 in this case).
The partial sums and carries remaining in the PEs in the center row i.e. $pe_{3,1}$, $pe_{3,2}$
and \(pe_{3,3}\) are summed in the carry-propagate adder to generate higher bits of the final product (two bits in this case).

Table 6-1. Computation of PEs in Figure 6-2 at each cycle.

(a)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>(pe_{1,1})</th>
<th>(pe_{1,2})</th>
<th>(pe_{1,3})</th>
<th>(pe_{2,1})</th>
<th>(pe_{2,2})</th>
<th>(pe_{2,3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x_{11}^0 y_{11}^0) &amp; (x_{11}^0 y_{11}^0) &amp; (x_{11}^0 y_{11}^0)</td>
<td>(x_{12}^0 y_{21}^0 + pe_{1,3}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,2}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,3})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(x_{11}^0 y_{11}^0 + 1) &amp; (x_{11}^0 y_{11}^0) &amp; (x_{11}^0 y_{11}^0)</td>
<td>(x_{12}^0 y_{21}^0 + pe_{1,1}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,2}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,3})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(x_{11}^0 y_{11}^0 + c_{1,1}) &amp; (x_{11}^0 y_{11}^0) &amp; (x_{11}^0 y_{11}^0)</td>
<td>(x_{12}^0 y_{21}^0 + pe_{1,1}) + (c_{2,1}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,2}) + (c_{2,2}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,3}) + (c_{2,3})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(1 + c_{1,1}^3) &amp; (x_{12}^0 y_{21}^0 + pe_{1,1}) + (c_{2,1}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,2}) + (c_{2,2}) &amp; (x_{12}^0 y_{21}^0 + pe_{1,3}) + (c_{2,3})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(c_{1,3}^4) &amp; (1 + pe_{1,1}^4 + c_{2,1}^4 + C_{2,1}^4) &amp; (c_{2,2}^4) &amp; (c_{2,3}^4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>(pe_{1,1}^5 + c_{2,1}^5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>(pe_{3,1})</th>
<th>(pe_{3,2})</th>
<th>(pe_{3,3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(pe_{2,1}^2 + pe_{4,1}^2) &amp; (pe_{2,2}^2 + pe_{4,2}^2) &amp; (c_{3,3}^2, pe_{3,3}^3) = (pe_{2,3}^2 + pe_{4,3}^2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(pe_{2,1}^3 + pe_{4,1}^3 + c_{3,1}^3) &amp; (pe_{3,1}^3 + pe_{4,2}^3 + c_{3,2}^3) &amp; (C_{3,3}^2, C_{3,3}^2, pe_{3,3}^4) = (pe_{3,2}^3 + pe_{2,3}^3 + pe_{4,3}^3 + c_{3,3}^3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For example, the input bits to $pe_{2,1}$ in cycle 3 are: $\bar{x}_{1,2}y_{2,1}$, output from $pe_{1,1}$ in cycle 2, constant 1, and carry bit from $pe_{2,1}$ in cycle 2. In cycle 3, $pe_{2,1}$ adds these four bits and generates one sum bit ($pe_{2,1}^3$) and two carry bits ($c_{2,1}^3$ and $C_{2,1}^3$).

- The final product bits are: $\{s_{p1}, s_p, pe_{3,3}^6, pe_{3,3}^5, pe_{5,3}^4, pe_{5,3}^3\}$.

The design describes a semisystolic array, where a data item accessed from memory is broadcast to and used by a number of PEs concurrently, in order to take advantage of longlines present in field programmable gate arrays (FPGAs) [118]. However, the design could be easily modified to eliminate the data broadcast and pipeline the data in different directions.

### 6.3 Bit-level dependence structures

Algorithms considered in this chapter are represented by Fortran-like nested Do loops having the following form –
\[ \text{do (} j_1 = l_1, u_1; j_2 = l_2, u_2; \ldots; j_n = l_n, u_n) \]

\[ S_1(\overline{j}) \]

\[ S_2(\overline{j}) \]

\[ \ldots \]

\[ S_q(\overline{j}) \]

end

where column vector \( \overline{j} = [j_1, j_2, \ldots, j_n]^T \) is the index vector (also called the index point); \( S_1(\overline{j}), S_2(\overline{j}), \ldots, S_m(\overline{j}) \) are the \( m \) assignment statements in iteration \( \overline{j} \); the lower bound and upper bound of the \( i^{th} \) nested loop, \( 1 \leq i \leq n \), are denoted by \( l_i \) and \( u_i \), respectively. The iteration space or index set \( J \) of loop (1) is the set consisting of all index vectors of that loop, that is, \( J = \{ \overline{j} : l_i \leq j_i \leq u_i, i = 1, \ldots, n \} \). We assume perfectly nested loops. Algorithms with \( n \) nested Do loops are called \( n \)-dimensional algorithms.

In general nested Do loops, cross-iteration dependencies may exist. If iteration \( \overline{j} \) depends on iteration \( \overline{j}' \), then a pair can describe this dependence \( (\overline{j}, \overline{d}) \), where \( \overline{d} = \overline{j} - \overline{j}' \) is the vector difference of the index vectors of these two iterations. Vector \( \overline{d} \) is called a dependence vector and is said to be valid at index point \( \overline{j}' \).

Assuming that iteration \( \overline{j} \) depends on iteration \( \overline{j}' \), there are three types of dependences [119]. The first type is called flow dependence (or read-after-write dependence), where an input variable of the computation in \( \overline{j} \) is an output variable
of the computation in $j$. The second is called antidependence (or write-after-read dependence), where an output variable of the computation in iteration $j$ is an input variable of the computation in $j$. The third is called output dependence (or write-after-write dependence), where an output variable of the computation in iteration $j$ is an output of the computation in iteration $j$.

We assume that every variable in a nested Do loop is written only once during the entire algorithm; therefore, there is no output dependence. Consider a dependence vector $\mathbf{d}$. If for any two arbitrary index vectors $j_1, j_2 \in J$ such that $j_2 - j_1 = \mathbf{d}$ and that dependence vector $\mathbf{d}$ is valid at $j_2$, then this dependence vector is uniform. If all dependence vectors are uniform, then the algorithm is a uniform dependence algorithm. For uniform dependence algorithms all dependence vectors are valid at every index point; hence, it is not necessary to specify the points at which they are valid. When a dependence structure is not uniform, as the one to be presented later, it has to be specified where dependence is valid. A non-uniform dependence $\mathbf{d}$ is described by the pair $(j, \mathbf{d})$, where $\mathbf{d}$ is valid at index point $j$. In this chapter, an algorithm is characterized by a triplet $(J, D, E)$ where $J$ is the index set, $D$ is the dependence matrix containing all distinct dependence vectors as its columns, and $E$ contains all different computations in all iterations.

To get more insight on the dependence structures of expanded bit-level algorithms consider the following simple one-dimensional algorithm in loop (6.2).

\[
\begin{align*}
do (j = l, u) \\
x(j) &= x(j - h_1) \\
y(j) &= y(j - h_2) \\
z(j) &= z(j - h_3) + x(j) \cdot y(j)
\end{align*}
\]

(6.2)
Since this is one-dimensional, \( j, h_1, h_2 \) and \( h_3 \) are scalars instead of vectors. The index set and the dependence structure of this word-level algorithm are shown in Figure 6-3. Without loss of generality we assume that \( h_1 = h_2 = h_3 = 1 \).

![Figure 6-3. Index set and dependence structure of a one-dimensional algorithm.](image)

Figure 6-4 shows the dependence graph for the bit-level matrix multiplier design when \( r \) is even. In Figure 6-4, each index point in Figure 6-3 is replaced by a two-dimensional index set for the multiplication of two two’s complement words. Hence each index point in Figure 6-4 has three indices \( j, i_1 \) and \( i_2 \).

Each index point in the dependence graph represents a computation, and each line between two index points (with the exception of the broadcasted data) represents a flow dependence. (It may be easier to follow the subsequent discussion by drawing a rough analogy with the PEs in Example 1 and the index points in Figure 6-4: suppose that \( (j, i_2) \) represents the indices of the PEs, and index point \( (j, i_1, i_2)^T \) represents a computation of the corresponding PE). Consider the partial-sum bit \( z(j, i_1, i_2) \) in Figure 6-4. There are three flow dependences of \( z(j, i_1, i_2) \). The partial-sum bit \( z(j-1, i_1, i_2) \) is sent to index points \( (j, i_1, i_2)^T \) where \( j < e \) (\( e = \frac{r}{2} + 1 \) when \( r \) is even) and added to the partial-product bit and carry bit(s) there to generate the new partial-sum \( z(j, i_1, i_2) \). Hence, the dependence at these points can be described by the pair \( (q_1, d_j) \), where \( q_1 = (j, i_1, i_2)^T, \quad j < e \) and
\( \overline{d_1} = [1 \ 0 \ 0]^T \). Since this dependence is valid only at a subset of all index points it is not uniform. Similarly, the dependences of \( z(j, i_1, i_2) \) along points \( j > e \) and \( j = e \) can be obtained.

![Diagram of Bit-level dependence structure](image)

**Figure 6-4.** Bit-level dependence structure for \( r = 4, p = 3 \).

The lower-order carries \( c \), generated at index points \( (j, i_1, i_2)^T \), are sent to index points \( (j, i_1 + 1, i_2)^T \). Hence, the dependence at the points \( (j, i_1, i_2)^T \) is given by \( \overline{d_2} = [0 \ 1 \ 0]^T \) and is described by the pair \( ((j, i_1, i_2)^T, \overline{d_2}) \). The higher-order carries \( C \), generated at index points \( (j, i_1, i_2)^T \) where \( j = e \), are input into index points \( (j, i_1 + 1, i_2 + 1)^T \) where \( j = e \). Hence, the dependence at these points \( (e, i_1, i_2)^T \) is given by \( \overline{d_4} = [0 \ 1 \ 1]^T \), and the pair is \( ((e, i_1, i_2)^T, \overline{d_4}) \). The three control signal bits \( m, l \) and \( en \) (not shown in the dependence structure) are required to generate the partial-product bits according to the Baugh-Wooley scheme. The bits \( m, l \) and \( en \) signal the arrival of the most and least significant bits of each word of
and prevent incorrect values from being generated at the PEs during idle cycles respectively. The bits \( m, l \) and \( en \) are pipelined between rows of PEs, and are shown in Figure 6-5 as MSB, LSB and EN respectively.

The control signal bit \( m \) at index points \( (j, i_1, i_2)^T \) where \( j < e \) is sent to index points \( (j+1, i_1, i_2)^T \) where \( j < e \). The dependence vector is described by the pair \( (q_1, d_6) \) where \( q_1 = (j, i_1, i_2)^T, j < e \) and \( d_6 = [1 \ 0 \ 0]^T \). The control signal bits \( l \) and \( en \) pose dependence vector \( d_5 = [1 \ 0 \ 0]^T \) and are only valid at index points \( q = (j, i_1, i_2)^T, j < e, i_2 = p \). The signal bits \( m \) and \( l \) introduce an additional dependence \( d_7 = [0 \ 1 \ 0]^T \) at index points \( q \) as they generate two “1s” needed in the Baugh Wooley scheme for each intermediate product (see Figure 6-1(a)). The \( p \) bits of \( x(j) \) are broadcast along the \( i_2 \) axis and so they do not pose additional dependences.

The index set and bit-level dependence matrix obtained from the preceding discussion are presented next in (6.3) for \( j \leq e \). The structure is symmetric about \( j = e \), and computations occur in parallel in both halves. Therefore, we shall concentrate on the half with \( j \leq e \). Let \( J_1 \) and \( D_1 \) represent the index set and the dependence matrix when \( r \) is even.

Then, we have

\[
J_1 = \{ [j, i_1, i_2]^T : l \leq j \leq u, 1 \leq i_1 \leq p, 1 \leq i_2 \leq p, r = u - l + 1, i_1, i_2, j \in Z, r \text{ is even} \}
\]

\[
D_1 = [d_1, d_2, d_3, d_4, d_5, d_6, d_7] = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & -1 & 1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
j \leq e \quad j \leq e \quad j = e \quad j = e \quad q \quad j < e \quad q
\]
where, $e = \frac{r}{2} + 1$; and $\bar{q} = [j, i_1, i_2]^T$, $j < e$, $i_2 = p$. 

(6.3)

The above one-dimensional structures can be extended to the $n$-dimensional case. The $n$-dimensional index structure and dependence matrices are given in (6.4):

Let $\bar{q} = \begin{bmatrix} \bar{j} \\ \bar{i} \end{bmatrix}$ where, $\bar{j} = [j_1, \ldots, j_n]^T$; $\bar{i} = [i_1, i_2]^T$

$$J_1 = \{ \begin{bmatrix} \bar{j} \\ \bar{i} \end{bmatrix} : l_i \leq j_i \leq u_i, 1 \leq i_1 \leq p, 1 \leq i_2 \leq p, r_n = u_n - l_n + 1, i_1, i_2, j_i \in Z, i = 1, \ldots, n, r_n \text{ is even} \}$$

$$D_1 = \begin{bmatrix} d_1^T, d_2^T, d_3^T, d_4^T, d_5^T, d_6^T, d_7^T \end{bmatrix} = \begin{bmatrix} h_3 & 0 & 0 & 0 & h_3 & h_3 & 0 \\ 0 & \vec{\delta}_1 & \vec{\delta}_1 & 0 & 0 & \vec{\delta}_2 & \vec{\delta}_2 \\ \vec{\delta}_3 & \vec{\delta}_3 & \vec{\delta}_3 & \vec{\delta}_3 & \vec{\delta}_3 & \vec{\delta}_3 & \vec{\delta}_3 \\ j_n \leq e_n & j_n \leq e_n & j_n = e_n & j_n = e_n & \bar{q} & j_n < e_n & \bar{q} \end{bmatrix}$$

where, $e_n = \frac{r}{2} + 1$; $\bar{q} = [\bar{j}, i_1, i_2]^T$, $j_n < e_n, i_2 = p$; $\bar{\delta}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\bar{\delta}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and

$$\bar{\delta}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

(6.4)

For the index set and bit-level dependence matrix when $r$ is odd and the two’s complement bit-level matrix multiplication algorithm please refer to [120].

6.4 Design of Bit-Level Architectures

Based on the dependence structures presented earlier, we discuss in this section the design of bit-level architectures. Using a design method developed in [108,109], we show how to design a bit-level architecture for matrix multiplication. For the proofs and detailed description of the mapping model and optimization method, please see the references [108,109].

118
6.4.1 Summary of mapping model

Definition 1 (Linear algorithm transformation). A linear algorithm transformation maps an $n$-dimensional algorithm $(J, D, E)$ into a $(k-1)$-dimensional processor array according to the mapping:

$$\tau: J \rightarrow Z^k, \tau(j) = T\bar{j}, \forall j \in J$$

(6.5)

where, $T = \begin{bmatrix} S \\ \Pi \end{bmatrix}$ is the mapping matrix, $S \in Z^{(k-1)\times n}$ is the space mapping matrix and $\Pi \in Z^{1\times n}$ is the time mapping vector or linear schedule vector. The computation indexed by $\bar{j} \in J$ is executed at time $\Pi\bar{j}$ and at processor $S\bar{j}$. The mapping $\tau$ must satisfy the following four conditions:

1. $\Pi D > \bar{0}$. This preserves the partial ordering induced by the dependence vectors. If this condition is satisfied, then the computation indexed by $\bar{j} \in J$ is scheduled to execute only after the computations indexed by $\bar{j} - \bar{d} \in J, i = 1, \ldots, m$. The dependence relation is, therefore, respected.

2. $SD = PK$, where $P \in Z^{(k-1)\times r}$ is the matrix of interconnection primitives of the target machine that describes the connection links of processors in the processor array, and $K \in Z^{r\times m}$ is such that

$$\sum_{j=1}^{r} k_{ij} \leq \Pi d_i, \quad i = 1, \ldots, m$$

(6.6)

For an array in which each processor is connected to its four nearest eastern, southern, western and northern neighbours, it has four interconnection primitives

$[0,1]^T, [0,-1]^T, [1,0]^T, [-1,0]^T$ and matrix $P = \begin{bmatrix} 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 \end{bmatrix}$. 

119
(3) $\forall j_1, j_2 \in J$ if $j_1 \neq j_2$, then $\tau(j_1) \neq \tau(j_2)$ or $T_{j_1} \neq T_{j_2}$. This defines the condition to avoid computational conflicts. If the condition were not true, that is, $\tau(j_1) = \tau(j_2)$, then the computations indexed by $j_1$ and $j_2$ are mapped to the same processor at the same time, and a conflict occurs.

(4) The rank of $T$ is equal to $k$ ($\text{rank}(T) = k$). This guarantees that the algorithm is to be mapped into a $(k - 1)$-dimensional array and not a $k'$-dimensional array, where $k' < k - 1$.

In the following, some definitions and theorems in [108,109] that are needed for discussion in this chapter are stated. For the proofs and detailed description of the mapping model and optimization method, please see the references [108,109].

**Definition 2 (Constant Bounded Index Set).** An index set is constant bounded if it has the following form:

$$J = \left\{ [j_1, ..., j_n] : l_i \leq j_i \leq u_i; j_i, l_i, u_i \in \mathbb{Z}; i = 1, ..., n \right\}$$  \hspace{1cm} (6.7)

**Theorem 1.** If the computation at any index point takes one time unit, then the total execution time for algorithms with constant bounded index sets is

$$t = 1 + \sum_{i=1}^{n} |\pi_i| (u_i - l_i)$$  \hspace{1cm} (6.8)

where $\pi_i$ are the elements of time mapping vector $\Pi$. The total execution time $t$ is a monotonically increasing function of $|\pi_i|$, $i = 1, ..., n$.

**Definition 3 (Conflict Vector, Feasible and Non-Feasible Conflict Vectors, and Conflict-Free Mapping Matrix).** Given an algorithm $(J, D, E)$ and a mapping
matrix \( T \in Z^{k_{\times n}} \), an integral column vector \( \bar{\gamma} = [\gamma_1, ..., \gamma_n]^T \) is a conflict vector of the mapping matrix \( T \) if and only if \( T \bar{\gamma} = \bar{0} \) and \( \gcd(\gamma_1, ..., \gamma_n) = 1 \). If for any arbitrary index point \( j \in J, j + \bar{\gamma} \notin J \), then \( \bar{\gamma} \) is a feasible conflict vector. If there exists at least one index point \( j \in J \) such that \( j + \bar{\gamma} \in J \), then \( \bar{\gamma} \) is called a nonfeasible conflict vector. If all the conflict vectors are feasible, then this mapping matrix \( T \) is conflictfree.

**Theorem 2.** For algorithms with constant bounded index sets defined by Definition 2, a mapping matrix \( T \) is conflict-free if and only if for each of its conflict vectors \( \bar{\gamma} = [\gamma_1, ..., \gamma_i, ..., \gamma_n]^T \) there exists an entry \( \gamma_i \) such that \( |\gamma_i| > (u_i - l_i) \).

**Theorem 3 (Hermite normal form, pp.45 of [121]).** Let \( T \in Z^{k_{\times n}} \) and \( \text{rank} (T) = k \). Then there exists a unimodular\(^3\) matrix \( U \in Z^{n_{\times n}} \) such that \( TU = H = [L, 0] \) where \( L \in Z^{k_{\times k}} \) is a nonsingular and lower triangular matrix. Matrix \( H \) is called the Hermite normal form of \( T \).

**Theorem 4.** For a given mapping matrix \( T \), let \( H \) be its Hermite normal form and \( TU = H \), where \( U = [\bar{u}_1, ..., \bar{u}_n] \). Then, vector \( \bar{\gamma} \) is a conflict vector of mapping matrix \( T \), if and only if

\[
\bar{\gamma} = [\bar{u}_{k+1}, ..., \bar{u}_n] \begin{bmatrix} \beta_{k+1} \\ \vdots \\ \beta_n \end{bmatrix}
\]

\(^2\) \( \gcd(a_1, ..., a_n) \) denotes the greatest common divisor of integers \( a_1, ..., a_n \).

\(^3\) A matrix is unimodular if and only if it is integral and the absolute value of its determinant is one.
where \( \beta, i = k + 1, \ldots, n \), are arbitrary integers that are not all zeroes and are relatively prime.

### 6.4.2 Mapping algorithms to hardware

Consider the following matrix multiplication algorithm \( W = X \cdot Y - \)

\[
\begin{align*}
\text{Do} & \ (j_1 = 1, q; j_2 = 1, s; j_3 = 1, r) \\
& \quad z(j_1, j_2, j_3) = z(j_1, j_2, j_3 - 1) + x(j_1, j_3) \cdot y(j_3, j_2) \\
\text{End}
\end{align*}
\]

(6.9)

In this section we show how to map the above algorithm to a bit-level structure.

**Theorem 5.** The following mapping matrix

\[
T = \begin{bmatrix}
S \\
\Pi
\end{bmatrix} = \begin{bmatrix}
e & 0 & 1 & 0 & 0 \\
0 & p & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0
\end{bmatrix}
\]

where \( e = \frac{r}{2} + 1 \) for \( r \) even, and \( e = \frac{r + 1}{2} \) for \( r \) odd, is both feasible and time optimal.

**Proof:** (a) First we will show that \( T \) is feasible, i.e., it satisfies the conditions in Definition 1. The index set for the bit-level matrix multiplication in (6.9) is given by

\[
J = \left\{ [j_1, j_2, j_3, i_1, i_2]^T \in \mathbb{Z}^5; 1 \leq j_1 \leq q, 1 \leq j_2 \leq s, 1 \leq j_3 \leq e, \\
e = \frac{r}{2} + 1, 1 \leq i_1 \leq p, 1 \leq i_2 \leq p, r \text{ is even} \right\}
\]

(6.10)

and is constant bounded. The dependence structure for \( r \) even is obtained from (6.4) and is given by
\[
D = \begin{bmatrix}
z & c & z & C & en & l & m & m, l \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

(6.11)

\[
j_3 \leq e \quad j_3 \leq e \quad j_3 = e \quad j_3 = e \quad \overline{q} \quad j_3 < e \quad \overline{q}
\]

where \( \overline{q} = [j_1, j_2, j_3, i_1, i_2]^T \), \( j_3 < e, i_2 = p \). \( \Pi = [0 \ 0 \ 1 \ 1 \ 0] \) satisfies dependence relations for \( r \) even because \( \Pi D > 0 \). Using the dependence structure for \( r \) odd in [120], we can prove similarly that the dependence relation is satisfied for \( r \) odd.

Condition 2 in Definition 1 guarantees that the space mapping can be implemented in a systolic architecture with interconnection primitive matrix \( P \). Condition 2 does not have to be satisfied in our case because a new processor array is specially designed for the algorithm. It can be shown that \( T \) is conflict-free as follows.

Let,

\[
U = \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & -e \\
0 & 0 & 1 & 0 & e \\
0 & 1 & 0 & -p & 0
\end{bmatrix}, \quad H = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

Then \( U \) is unimodular, \( TU = H \), and \( H \) is the Hermite normal form of \( T \).

According to Theorem 4 all conflict vectors of \( T \) can be expressed as

\[
\overline{\gamma} = \begin{bmatrix}
0 & 1 \\
1 & 0 \\
0 & -e \\
0 & e \\
0 & -p & 0
\end{bmatrix} = \beta_1 \overline{u}_4 + \beta_2 \overline{u}_5
\]
where $\beta_4$ and $\beta_5$ are relatively prime and are not all zeroes. No matter what allowable values $\beta_4$ and $\beta_5$ take, either $|\gamma_4|=|\beta_4 e| \geq e$ or $|\gamma_5|=|\beta_4 p| \geq p$ because $\beta_4$ and $\beta_5$ cannot both be zero. Since the index set is constant bounded, according to Theorem 2, $\bar{\gamma}$ is feasible for any allowable values of $\beta_4$ and $\beta_5$, and $T$ is conflict-free. We can also verify that $\text{rank}(T)=3$. Therefore, according to Definition 1, $T$ is feasible.

(b) It can be shown that $\Pi_1=[0 \ 0 \ 1 \ 1 \ 0]$ is time optimal; i.e., there does not exist another feasible time mapping vector $\Pi_1=[\pi_1 \ \pi_2 \ \pi_3 \ \pi_4 \ \pi_5]$ that has a total shorter execution time.

Suppose that $\Pi_1$ exists. It must satisfy the partial ordering induced by dependence vectors, i.e. $\Pi_1 D > \bar{0}$, i.e. $[\pi_3 \ \pi_4 - \pi_5 \ \pi_4 + \pi_5 \ \pi_3 \ \pi_3 \ \pi_4]^T > \bar{0}$. The smallest values of $\pi_i$ that satisfy the above equation are $\pi_3 = \pi_4 = 1$, which gives $\Pi_1 = [0 \ 0 \ 1 \ 1 \ 0]$ which is the same as $\Pi$. Hence $\Pi$ is time optimal. We can prove similarly that $\Pi$ is time optimal when $r$ is odd.

The total execution time measures the number of cycles between the arrival of the first bit at any of the processors and the delivery of the last result bit to the environment, where it takes one time unit to complete a bit-operation on any processor. The total execution time required with the mapping matrix $T$ is calculated using the following equation (see [108,109] for details):

$$t = \max \{\Pi(\bar{q}_1 - \bar{q}_2 : q_1, q_2 \in J)\} + 1$$

Then, $t = [0 \ 0 \ 1 \ 1 \ 0]([s \ q \ \frac{r}{2} + 1 \ p \ p]^T - [1 \ 1 \ 1 \ 1 \ 1]^T) + 1$
\[ t = \left( \frac{r}{2} \right) + p. \]

An extra clock cycle is introduced by the “1” bit that is required to be added in the Baugh-Wooley method. Hence the total execution time

\[ t = \left( \frac{r}{2} \right) + p + 1. \]  \hspace{1cm} (6.12)

Equation (6.12) shows that the total execution time is \( O(r) \). This is compared to the execution time of the carry-save word-level matrix multiplier of \( O(gsp \log p) \) space and \( O(r \log p) \) execution time. Therefore, the bit-level matrix multiplier is \( O(\log p) \) faster than a word-level matrix multiplier.

The total number of processors required is

\( (r+1)pqs \)

The large, regular, highly pipelined structure of the bit-level array affords high throughput and makes it suitable for a VLSI implementation.

### 6.5 Hardware Implementation of Bit-Level Array

In this section, the hardware implementation of the various cells in the bit-level matrix multiplier array is described.

Mainly, there are three types of cells in the array – \( A, B, \) and \( C \). Each cell is a modified serial adder with added functionality to implement the Baugh-Wooley algorithm. A fourth type of cell – \( cell D \) is used for adding carries generated from cell \( A \) to the partial product with the correct weight. The small rectangle in each figure represents a one-bit clocked register.
The layout of the array to generate one element of the product matrix is shown in Figure 6-5 for the case when $r$ is even. The elements of the input vector $Y$ may be pre-stored in cells A and B in the array. With a few modifications in the center row the array can be used for the case when $r$ is odd. The final product bits are shifted out of the rightmost cell in the center row into shift registers (not shown). Only after the last cycle, when there are no more multiplicand-multiples to be added, the carries remaining in the center row are propagated using additional combinational logic (such as a carry-propagate adder that is connected to the PEs in the center row; this is not shown, in order to keep Figure 6-5 simple) instead of assimilating the carries using $p$ iterations. The carry chain is broken in every cycle but the last during the computation of a word of the product matrix.

Figure 6-5. Entire layout of the two’s complement bit-level array for matrix multiplication with even $r$. 
Figure 6-6 shows the schematic for cell A. There is one cell of type A in each row of the array, and its structure is more complex than the other cells. Different bits of a word ‘x’ namely $x^{p-1}x^{p-2}...x^0$ arrive at the cell at each clock. The most significant bit of word ‘y’ namely $y^{p-1}$ may be pre-stored in a register in the cell ($y_i$) before execution starts. The and gate ‘A4’ forms the partial product $x^{p-1}y^{p-1}$ when the bit $x^{p-1}$ arrives (the input at $m_i$ is a ‘1’ when $x^{p-1}$ arrives at the cell and the input at $l_i$ is a ‘1’ when $x^0$ arrives at the cell). For all other bits of $x$ ‘A4’ forms the partial product $x^iy^{p-1}$. The register One generates the two extra partial product bits ‘1’ (required in the Baugh-Wooley multiplication scheme) with the correct weight. The enable bit $e_i$ disables the cell from generating ‘false ones’ during idle cycles. Since no more than four bits are required to be added at any time, one sum and two carry bits are generated.

Most of the cells in the array are of type cell B, and it has a simple regular structure. The schematic for cell B is shown in Figure 6-7. The gate ‘A5’ generates the partial product $x^iy^j$, when $x^i$ is not the most significant bit; otherwise, it generates the partial-product $x^{p-1}y^j$. A partial-sum bit of the same weight, generated in a neighboring PE, is added to this partial-product bit.
The PEs in the center row of the array are of type cell C. Three partial sum bits of the same weight and one higher order carry bit, from neighboring PEs, are added in this cell. The schematic for cell C when \( r \) is even is shown in Figure 6-8.

Cell D is shown in Figure 6-9 and is used for handling the carries coming out of cell A and adding them to a partial sum of the same weight.
The bit-level approach requires one or two extra registers in each cell for storing the carry bit(s) as compared to the word-level approach. In addition, the bit-level design also requires an adder to sum the partial sums and carries in the center column at the end of the computation. The complexity of a cell in a bit-level design can be lowered by allowing partial carry chains of size $u$, where, $1 < u < p$; this reduces the number of registers for storing carries in the cell. It would be interesting to study the speed-area tradeoffs of this approach.

6.6 Summary

In this chapter, we presented a bit-level design for fixed-point two’s complement matrix multiplication that is faster than existing designs. Bit-level designs are $O(\log p)$ times faster than the fastest word-level design for fixed-point matrix multiplication. The dependence and mapping matrices and hardware structure for the design have been presented. It is proven that the design is time optimal and conflict-free. This work is significant because it can be used in digital signal and image processing applications. It is possible to convert an existing word-level design to bit-level with only small changes in hardware to gain speedup.
Chapter 7

Performance-cost tradeoffs in distributed arithmetic based discrete cosine transform

7.1 Background

The Discrete Cosine Transform (DCT) [122] is widely used in the implementation of codecs for video transmission. Several implementations of the DCT on Field Programmable Gate Arrays (FPGAs) have been described [123,124]. The FPGA makes it possible to develop high-performance DCT architectures with HDTV throughputs. However, in order to do so it is imperative to gain a deeper understanding of its features and the design rules to be followed.

The critical path dictates the fastest time at which an entire design can run. By varying the length of the critical path different tradeoffs in area, throughput, latency and power can be obtained. In this chapter, $d$-$r$ DA structures for DCT on Xilinx FPGAs are described, where $d$ is the digit size and $r$ is the length of the carry chain in the critical path, for $1 \leq r \leq n$, where $n$ is the width of the internal data paths in the design. For example, 1-16 represents bit-serial distributed arithmetic designs for DCT, with a 16-bit carry chain in each shift accumulator. Similarly, a 3-8 design implies a 3-bit serial distributed arithmetic DCT design, with two 8-bit carry chains in each 16-bit shift accumulator and adder. The designs are implemented on a Virtex-E chip (xcv300e-8pq240). The area, latency, throughput and power are determined for various designs with different $d$ and $r$. 

130
The performance trade-offs among a class of three different 8-point 1-D DCT architectures are presented in [69]. The architectures chosen for implementation are the Distributed Arithmetic (DA) architecture with digit sizes (1 and 3), digit-serial flow graph and systolic architecture. Their experiments show that the DA architecture is the best in terms of area, speed and latency. In the DA technique, instead of calculating the inner product with multipliers, the multiplication is performed using memories (ROMs) and shift accumulators [70, 71].

Design exploration for pipelined IDCT is reported in [124]. Since efficient pipelining is important to obtain good performance in the decoder, designs are pipelined in different ways to obtain a variety of cost/performance trade-offs.

In [10], new designs for DA-based multiplier accumulators with variable length carry chains were presented. Traditional lookup table (LUT) based DA architectures contain one or more carry propagation chains in the critical path. New designs were presented to reduce or eliminate the carry-propagate chain from the critical path in LUT-based DA architectures on FPGAs. Designs in which a w-bit carry chain, where w is the word length, is broken into smaller r-bit chains, \( 1 \leq r < w \), are described in the paper.

In this chapter, the DA DCT design space, with variable length carry chains in the critical path, is explored. The cost-performance tradeoffs of the new designs are discussed.

### 7.2 Variable Length Carry Chain Designs for DCT

The \( N \) th-order DCT matrix \( C \) is defined by

\[
    c_{k,l} = \frac{2}{N} \cos \left[ \frac{(2k - 1)(l - 1)\pi}{2N} \right] \quad \text{for} \quad k = 1, 2, \ldots, N, l = 2, \ldots, N
\]
and \( c_{k,l} = N^{-1/2} \) for \( l = 1 \). The \( N \times 1 \) DCT of each column of the input data matrix \( X \) order \( N \) is given by \( X'C \) [125]. Designs for 1-D DCT with variable length carry chains are described in this section.

![Figure 7-1. Traditional 1-D DA DCT.](image)

Figure 7-1 shows the traditional 1-bit serial DA architecture for 8-point DCT using \textit{word-level} components [69,125]. In word level components the carry chain in the components is as large as the internal data path width. For example, in Figure 7-1, the shift accumulators have a 16-bit carry chain.
Figure 7-2(a) and Figure 7-2(b) show two proposed designs for 16-bit shift accumulators with 4-bit carry chains. The design in Figure 7-2(b) has the advantage
that it can be built using standard optimized 5-bit registered adders from the library. Hence the design in Figure 7-2(b) has smaller routing delays and results in lower latency than the design in Figure 7-2(a). Therefore, this design was used in the implementation of the \( r \)-bit shift accumulators. The basic processing element (PE) is an adder, which adds three or four bits to generate a sum and one or two carry bits. The sum bit is registered, and the carry is propagated to the next PE except in the last PE of the component where it is registered. The two bits \( C_L \) and \( C_H \), shown in Figure 7-2(a), represent the lower order and higher order carry bits respectively generated from the addition of four input bits in the last PE. These are added to the incoming bits of correct weight in the next cycle. The partial sums and carries, remaining in the shift accumulators after the computation is over, are summed using a carry-propagate adder to form the final output. Therefore, the carry-propagate adder does not lie in the critical path. The shift accumulators can be used in place of the word-level shift accumulators, in Figure 7-1, to create a 1-4 DA DCT structure. Similarly, other components with carry chains of variable lengths can be designed.

Two other architectures, namely the 2-bit DA and 3-bit DA for the DCT are also implemented with varying carry chains. Both of these are similar to the 1-bit DA DCT design, except that 2 bits and 3 bits of the input matrix can be processed in parallel in the 2-bit DA and 3-bit DA architectures for DCT respectively [69]. The 2-bit DCT design differs from the bit-serial design in that the bit-serial adders are replaced by 2-bit serial adders and the ROMs are replicated twice. In the 3-bit DCT designs, the bit-serial adders are replaced by 3-bit serial adders and the ROMs are replicated thrice. Figure 7-3 and Figure 7-4 shows the shift accumulator components used in the 2-bit and 3-bit DA DCT designs respectively. The shift accumulator in Figure 7-3 shifts 2 bits at a time, whereas in Figure 7-4 it shifts three bits at a time.
Figure 7-3. Design of an 8-bit (or larger) shift accumulator, which shifts two bits at a time, using 5-bit components.

Figure 7-4. Design of an 8-bit (or larger) shift accumulator, which shifts three bits at a time, using 5-bit components.
Figure 7-5 shows the $r$-bit registered adder used in the 2-bit and 3-bit DA DCT architectures. This component has a simple structure. Note that the partial sums and carries are not assimilated at this stage, and are passed on to the next stage in the same form.

The $d$-$r$ DA DCT designs are implemented on an Virtex E chip, and compared with traditional designs. The implementation results are presented in the next section.

### 7.3 Implementation and Results

All of the various combinations of the $d$-$r$ DA DCT designs, described in the previous section, were implemented for $1 \leq d \leq 3$ and $r = 1, 2, 4, 8$ and word level. The designs were coded using Verilog HDL and implemented on a Virtex-E chip (xcv300e-8pq240) chip using Xilinx ISE 4.2i. The designs were verified through extensive simulations. Constraints were entered in the user constraints file (ucf). The resources required for each design are given in terms of the total number of Slices that are used. The overall path delay has two components: delay through the logic gates (logic delay) and the route delay. After the designs were placed and routed by the place-and-route tool, they were examined using the Timing Analyzer tool to obtain the latency.
Table 7-1 shows the resources and computation time required by the DA DCT designs for different carry chain lengths, for a word length of 16 bits. It can be noted that the design dct 1-8 with total delay 7.72ns and number of slices 327 provides a cost-delay product comparable to dct-word. The design dct 1-8 runs faster than dct-word, and has a higher throughput. The design dct 2-word has the smallest cost delay product among all word-level designs. It can be noted that the routing delays are large when the optimized word-level library components are not used. The reason is that the automated place-and route tool was used; however, manual placement may lower the routing delay for these designs.

Table 7-1. Values for logic and routing delays and number of slices for dct on xcv300e-8pq240

<table>
<thead>
<tr>
<th>Carry chain length (r)</th>
<th>Digit size (d)</th>
<th>Logic delay (ns)</th>
<th>Route delay (ns)</th>
<th>Total delay (ns)</th>
<th>Number of slices</th>
<th>Normalized cost-delay product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-bit 1</td>
<td>1</td>
<td>3.10</td>
<td>3.58</td>
<td>6.68</td>
<td>387</td>
<td>0.54</td>
</tr>
<tr>
<td>2-bit 1</td>
<td>1</td>
<td>2.96</td>
<td>5.15</td>
<td>8.11</td>
<td>375</td>
<td>0.64</td>
</tr>
<tr>
<td>4-bit 1</td>
<td>1</td>
<td>3.15</td>
<td>4.60</td>
<td>7.75</td>
<td>343</td>
<td>0.56</td>
</tr>
<tr>
<td>8-bit 1</td>
<td>1</td>
<td>3.35</td>
<td>4.37</td>
<td>7.72</td>
<td>327</td>
<td>0.53</td>
</tr>
<tr>
<td>word 1</td>
<td>1</td>
<td>3.56</td>
<td>4.21</td>
<td>7.77</td>
<td>319</td>
<td>0.52</td>
</tr>
<tr>
<td>1-bit 2</td>
<td>2</td>
<td>2.46</td>
<td>4.36</td>
<td>6.82</td>
<td>966</td>
<td>1.72</td>
</tr>
<tr>
<td>2-bit 2</td>
<td>2</td>
<td>3.27</td>
<td>5.08</td>
<td>8.35</td>
<td>608</td>
<td>1.36</td>
</tr>
<tr>
<td>4-bit 2</td>
<td>2</td>
<td>3.25</td>
<td>5.29</td>
<td>8.54</td>
<td>512</td>
<td>1.17</td>
</tr>
<tr>
<td>word 2</td>
<td>2</td>
<td>3.47</td>
<td>5.46</td>
<td>8.93</td>
<td>416</td>
<td>1</td>
</tr>
</tbody>
</table>

7.4 Summary

In this chapter, $d$-$r$ DA structures for DCT on Xilinx FPGAs are described, where $d$ is the digit size and $r$ is the length of the carry chain in the critical path, for $1 \leq r \leq n$, where $n$ is the width of the internal data paths in the design. The designs are implemented on a Virtex-E chip (xcv300e-8pq240). The area, latency, and throughput are determined for various designs with different $d$ and $r$. It is shown that
the designs provide a variety of cost/performance trade-offs. A particular design can be selected depending on the needs of the target application.
Chapter 8

Conclusion

8.1 Summary

In this dissertation, new designs for efficient storage, retrieval and processing of video are presented. To facilitate efficient storage and retrieval of video, the design of the SAM multimedia storage system is presented. For efficient processing of video, the design space, with variable length carry chains in the critical path, is explored for DA based DCT.

A new inter-node data layout scheme called constrained random striping (CRS) is proposed for SAM. The main feature of this scheme is that it uses a novel placement of the stripe units across nodes on a logical loop, which reduces the start-up latency of clients. The primary idea behind CRS is to partition the data into fragments, and place the fragments so that those needed earlier are placed on higher priority nodes that can transmit earlier in a round. The simulation results show that the CRS scheme provides a reduction of up to 40% in the start-up latency over other schemes.

A new intra-node placement scheme called Partial-Bundled is developed for VBR multi-resolution video. The advantage of this scheme is that it can provide higher throughput than other existing schemes, when the QoS parameter is jitter. Simulation results show that this scheme can support from 10% to 100% more clients than other schemes, depending on system parameters such as round time.
A new admission control scheme is also presented, which can provide QoS guarantees, so that the client jitter is bounded. In this scheme, deterministic guarantees are provided to minimum resolution layers, and statistical guarantees are provided to higher resolution layers. This allows more clients to be admitted than if a purely deterministic approach were used. It is different from previously proposed approaches that provide only statistical guarantees in that minimum or necessary layers are not dropped during overload rounds. Otherwise, it is possible that an important frame such as an I-frame may be dropped in an overload round, which may result in an unacceptably large glitch at the client display.

System support for graceful degradation in the presence of node failure is provided. Two heuristic algorithms are proposed to provide for graceful degradation. The heuristic algorithms attempt to maximize the rewards, related to average quality of a stream, while reducing the resolution of clients.

A new scheme is presented to support interactive operations such as fast-forward and rewind. We show that the statistical multiplexing of available resources can reduce the resources required to support interactive service on a parallel server. An analytical model is developed to predict the probability with which stream requirements can be met for a given node buffer level. It is proved that the probability of satisfying all stream requests with a buffer level of 0 in a round lies in the interval (0.5, 1). It is shown that the proposed scheme provides good QoS to the clients, with much smaller buffer levels than those required if statistical resource sharing is not used.

In SAM compressed video is used, and a second part of this dissertation looks at the design of video coders and decoders. Multiplier designs are implemented using serial and parallel distributed arithmetic look-up-table based structures (DALUT), in which an $n$-bit carry chain, where $n$ is the word length, is broken into smaller $r$-bit chains, $1 \leq r < n$. The results show that the proposed
designs can achieve speedup by a factor of at least 1.5 over traditional DA designs in some cases. A new multiplier design is also implemented using systolic arrays, which requires considerably more resources on FPGAs than DALUT-based multipliers.

The DALUT designs are used in the implementation of DA-based DCT. New designs are presented for $d$-$r$ DA structures for DCT on Xilinx FPGAs, where $d$ is the digit size and $r$ is the length of the carry chain in the critical path, for $1 \leq r \leq n$ where $n$ is the width of the internal data paths in the design. The area, latency and throughput are determined for various designs with different $d$ and $r$.

### 8.2 Possible Research Directions

In Chapter 3, an intra node placement scheme for VBR multi-resolution video was presented. For the simulation, traces of 2-layered temporally scalable MPEG-4 video were used. The disadvantage of using temporally scalable video is that the base layer (which comprises the I and P frames) has a high bit rate compared to the enhancement layers (which has only B frames). Since the base layer is provided deterministic guarantees, due to its high bit rate the number of clients that can be admitted to the system is significantly reduced. It would be interesting to explore the usage of other types of scalable video such as spatially scalable video, and video with more than two layers, with a base layer having a smaller bit rate. This can potentially increase the number of clients that are admitted to the system.

Caching is a simple method for reducing disk I/O bandwidth requirements [126, 127]. The basic idea behind caching is to store the data already read by the previous stream in a cache and feed the next stream for the same movie from the cache instead of from the disk. This avoids the need to initiate a new disk I/O stream. The commercial viability of the scheme depends on the trade-off between the cost of supporting a new disk I/O stream and the cost of the fast cache memory.
To implement caching in the parallel server, it is necessary to have a global view of the caches located in each node, in order to reuse data when a new stream wants to reuse parts of existing video data. Developing caching schemes for SAM is another possible avenue for further research.

In the data layout scheme presented in this dissertation it is assumed that data is accessed in blocks of constant size. However, there are different trade-offs when selecting the block size, such as node buffer size, throughput etc, depending on application characteristics. It may be more effective to support multiple block sizes for heterogeneous applications. One obvious advantage of using a multiple block size is that it can lower internal fragmentation and node buffer required when the stored data has widely different characteristics. The downside is that it complicates storage space management and can introduce external fragmentation. Using different block sizes also impacts the scheduling and delivery across the distributed nodes, and the admission control. It would be interesting to investigate the effect of using multiple block sizes.
APPENDIX

The IBM Ultrastar 18ES disk is modeled for the simulation. The main features of the model are described here. The IBM Ultrastar disk parameters are presented below:

IBM Ultrastar 18ES disk parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage space</td>
<td>18.2 GB</td>
</tr>
<tr>
<td>Number of heads</td>
<td>10</td>
</tr>
<tr>
<td>Number of cylinders</td>
<td>11474</td>
</tr>
<tr>
<td>Number of zones</td>
<td>55</td>
</tr>
<tr>
<td>Sector size</td>
<td>512 bytes</td>
</tr>
<tr>
<td>Rotation speed</td>
<td>7200 RPM</td>
</tr>
<tr>
<td>Latency (T_{lat})</td>
<td>8.33ms</td>
</tr>
<tr>
<td>Sectors per track (N_{scc})</td>
<td>390 to 247</td>
</tr>
<tr>
<td>Sustained data rate</td>
<td>12.7 to 20.2 MB/s</td>
</tr>
<tr>
<td>Maximum seek time</td>
<td>13.0 ms</td>
</tr>
<tr>
<td>Interface transfer rate</td>
<td>200 MB/s</td>
</tr>
<tr>
<td>Track skew factor (N_{tskew})</td>
<td>37 – 58 sectors</td>
</tr>
<tr>
<td>Cylinder skew factor (N_{cskew})</td>
<td>63 - 101 sectors</td>
</tr>
</tbody>
</table>

The I/O time for accessing data on a disk drive can be decomposed into several components associated with the physical operations required to process the I/O, such as seek time, transfer time, rotational delay, etc.

**Seek Time:** The seek time ($T_{seek}$) is the time elapsed while the disk head is moving from its current position to the cylinder that contains the required data. This time is a function of the distance that the head has to travel, which is measured in number of cylinders. The simulator stores the current disk head position. When a new request
needs to be processed, the simulator computes the cylinder where the requested data block is located, and calculates the difference between this and the current head position (called \( \text{dist} \)). It uses the following formulae to compute the seek time for the request:

\[
\begin{align*}
T_{\text{seek}} &= a + b \times \text{dist} \quad \text{dist} \leq 50 \\
T_{\text{seek}} &= c + d \times \sqrt{\text{dist}} \quad 50 < \text{dist} \leq 5000 \\
T_{\text{seek}} &= e + f \times \text{dist} \quad 5000 < \text{dist}
\end{align*}
\]

\( a = 1.4377, \quad b = 0.0239, \quad c = 2.2148, \quad d = 0.0866, \quad e = 4.9842, \quad f = 0.0007 \)

The seek time function of the disks is obtained through offline experiments [128]. The values of the parameters \( a, b, c, d, e \) and \( f \) are obtained via curve fitting.

**Rotational latency:** The rotational latency \( (T_{\text{rot}}) \) is the time elapsed after the head is positioned in the right track until the first byte of data is positioned under the disk head. Given that the disk starts at sector \( a \), seeks for time \( T_{\text{seek}} \), and must reach sector \( b \), \( T_{\text{rot}} \) can be calculated as follows:

\[
T_{\text{rot}} = \begin{cases} 
(b - a) \times T_{\text{lat}} / N_{\text{sec}} & b \geq a \\
(T_{\text{sec}} - (a - b)) \times T_{\text{lat}} / N_{\text{sec}} & a < b 
\end{cases}
\]

The above assumes that rotational skewing is present. In *cylinder skew* the start sectors of adjacent tracks are offset to minimize the likely wait time when switching tracks. Similarly, the start sector of tracks within the same cylinder are offset so that after reading from the first head/track in the cylinder, we can switch to the next one without losing our "pace". This is called *head skew*. If rotational skewing is not present then the performance is hit due to unnecessary platter rotations.
For simplicity, the rotational latency was modeled in our simulator by a uniformly distributed random variable in the range $[0, T_{rot}]$, where $T_{rot}$ is the time for a complete rotation of the disk platter.

**Track switch time:** The track switch $(T_{trkswitch})$ is the extra time necessary for the disk head to switch tracks when reading a data block that spans multiple tracks of the same cylinder. This time can be computed from $N_{tskew}$, which is the offset, in sectors, from the last sector of a track to the first sector of the following track on a cylinder. The track switch time for the $i$th zone can be calculated as:

\[
\frac{N_{tskew}^i T_{lat}}{N_{sec}^i}
\]

The track skew factor can be obtained using special SCSI commands to the disk drive. The total track switch time for reading a requested data block is determined by the number of track boundaries that are crossed by that data block.

**Cylinder switch time:** The cylinder switch time $(T_{cylswitch})$ is the extra time necessary for the disk head to switch cylinders when reading a data block that spans multiple cylinders. This time is computed in a manner similar to the track switch time, except that the cylinder skew factor is used instead of the track skew factor.

**Transfer time:** The disk transfer time is the time required to transfer data from the disk surface to the on-disk buffer or vice-versa. This time can be computed by dividing the amount of accessed data by the disk transfer rate. However, modern hard disks employ a technique called *zoned bit recording (ZBR)*, also sometimes called *multiple zone recording* or even just *zone recording*. With this technique, tracks are grouped into zones based on their distance from the center of the disk, and each zone is assigned a number of sectors per track. As you move from the innermost part of the disk to the outer edge, you move through different zones, each containing more sectors per track than the one before. This allows for more efficient
use of the larger tracks on the outside of the disk. However, the transfer rate is not constant over all regions of the disk due to disk zoning. But in each zone the disk transfer rate is practically constant and can be computed by dividing the track size by the disk revolution time. The zone layout information of a disk can be obtained using special SCSI commands. In our simulator, using the zone layout information obtained from [128], the disk zone on which a requested data block is stored is determined. The transfer rate of this zone is then used to compute the disk transfer time for a particular request.
BIBLIOGRAPHY


[106] Xilinx Inc. XC4000 XL Electrical characteristics, v1.7, October 1999


[118] Xilinx Inc., Virtex 2.5V Field Programmable Gate Array Data Book (v2.4), 2000.


