A Concurrent Video Compression System

by

Francis Honoré

S.B., Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1991

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1994

© Massachusetts Institute of Technology 1994. All rights reserved.

Author

Department of Electrical Engineering and Computer Science

January 21, 1994

Certified by

Stephen A. Ward
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by

Fredric R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Massachusetts Institute of Technology
APR 06 1994
LIBRARIES
A Concurrent Video Compression System

by

Francis Honoré

Submitted to the Department of Electrical Engineering and Computer Science on January 21, 1994, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

A system for evaluating and implementing video processing algorithms using a massively parallel computing environment is the subject of this thesis. This thesis establishes a framework for a parallel processing approach to a proposed industry standard for video compression. Not only will this thesis show that such an approach leads to great improvements in performance in processing speeds as well as degree of compression but it will also demonstrate the salient features of the NuMesh Project. The NuMesh system is a heterogeneous multicomputer platform with a three-dimensional high speed programmable interconnect.

Thesis Supervisor: Stephen A. Ward
Title: Professor of Electrical Engineering and Computer Science
Acknowledgements

I would like to thank Professor Steve Ward for providing a great research opportunity in the Computer Architecture Group and for his support and guidance. I would also like to thank all the members of the NuMesh team in working on the various components of the system, especially Chris Metcalf’s work on the simulator and assembler, John Nguyen’s NuMesh interface and John Pezaris, Milan Minsky and Russ Tessier work on the hardware. I would like to give special thanks to my family for their support and encouragement over the years.

This material is based upon work supported in part under a National Science Foundation Graduate Fellowship.
Contents

1 Introduction ........................................... 8
  1.1 Digital Video and Current Technology .................. 8
  1.2 Thesis Organization .................................. 10

2 The MPEG Compression Standard ......................... 11
  2.1 Overview of MPEG ................................... 11
  2.2 MPEG Compression Model ............................. 12
  2.3 Motion Compensation Stage ............................ 14
  2.4 Transformation Stage ................................ 15
  2.5 Quantization Stage .................................. 17
  2.6 Coding Stage ........................................ 17
  2.7 MPEG Summary ....................................... 18

3 The NuMesh Project .................................... 19
  3.1 Introduction ......................................... 19
  3.2 Two Dimensional Prototype ............................ 20
  3.3 Processing Elements ................................. 21
  3.4 Applications ......................................... 21
  3.5 NuMesh Simulator ..................................... 23
  3.6 Prototype Software Environment ....................... 23
  3.7 Summary .............................................. 23

4
CONTENTS

4 Concurrent Video Compression 25
   4.1 Ways of Benefitting from Concurrency .................. 25
   4.2 Tradeoffs in Achieving Concurrency .................. 26

5 Performance Analysis 30
   5.1 Introduction ........................................... 30
   5.2 Processing Time ........................................ 30
   5.3 Summary ................................................ 33

6 Conclusion 34
   6.1 MPEG encoder using NuMesh .......................... 34
   6.2 Applications and Improvements ....................... 34
   6.3 Future Work ........................................... 35
      6.3.1 Video Source Hardware .......................... 35
      6.3.2 MPEG Audio ....................................... 35
      6.3.3 Mass Storage .......................... 35

A MPEG Code Excerpts 36

B MPEG Examples 47
## List of Figures

2-1 Model of the MPEG encoder and decoder subsystems. ........................................ 13  
2-2 The frame relationships for the motion compensation model for MPEG. .. 14  
2-3 Two dimensional DCT coefficients arrangement. ................................. 16  
2-4 Default quantization matrix for DCT coefficients of the three frame types. . 17  
3-1 NuMesh Prototypye Modules. ............................................................. 20  
3-2 NuMesh Data Paths. ................................................................. 22  
3-3 A simple example of CFSM code. .................................................. 24  
4-1 One possible division of the MPEG encoder. .................................... 27  
4-2 Partitioning of the MPEG encoder tasks on a 2D mesh. ....................... 28  
B-1 A frame from test sequence *table-tennis*. Original in color. ............. 48  
B-2 A frame from test sequence *mobile*. Original in color. ...................... 48
## List of Tables

1.1 Some characteristics of the three television standards .......................... 9
1.2 Requirements of a 60-minute full-motion digital video sequence (12 bits/pixel). 10

5.1 Local memory as a function of the number of processors. ...................... 32

B.1 Some results of obtained from Stanford’s software MPEG encoder on various sources. ................................................................. 47
Chapter 1

Introduction

Computer manipulation of digitized video is increasingly achieving widespread use in diverse applications. New and advancing technologies in displays, processing power, and networks are helping to drive applications in such areas as medical records and treatment, scientific visualization, information retrieval, education, and entertainment. Important to these advancements is the establishment of standards for the exchange of visual and audio data as well as facilitating research in computer interpretation of this data. One such standard for compressing video has been proposed by the Motion Picture Experts Group (MPEG) and has been received well by industry [8, 6]. Use of this standard in the higher performance computing environments of the future for communicating information in video form will enhance human interaction with computers.

1.1 Digital Video and Current Technology

In order to handle video information in digital form, several stages are involved. Three major international television standards for transmitting and receiving video exist. These standards, NTSC, PAL and SECAM are outlined briefly in Table 1.1. The film industry has its own standard with a different aspect ratio and frame rate. These formats are all analog, meaning that the images are represented as a continuous range of levels of voltages. Video for display and processing on a computer must first be digitized. The digitization process involves representing a scene as a discrete set of numbers. This analog-to-digital
conversion takes a source such as a camera or TV receiver and generates binary values to represent them. This process is referred to as the **capture** stage.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>NTSC</th>
<th>PAL</th>
<th>SECAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field rate (Hz)</td>
<td>59.94</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Vertical lines</td>
<td>525</td>
<td>625</td>
<td>625</td>
</tr>
<tr>
<td>Luminance bandwidth (MHz)</td>
<td>4.2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Chrominance bandwidth (MHz)</td>
<td>1.3</td>
<td>1.3</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1.1: Some characteristics of the three television standards

In the capture stage, an analog-to-digital converter will typically divide a sequence of video frames into 24 bit picture elements (or pixels). Several integrated chips exist to perform this function. These bits representing pixels can then be treated as data by a computer. The process can be reversed to display this image sequence on a computer's display. The display stage recreates the sequence of frames through a digital-to-analog converter.

A large volume of data can be generated at the capture stage. Table 1.1 shows the amount of data generated for a 60 minute clip at various picture resolutions at 30 frames per second. These data rates for what will be referred to as raw video data put quite a strain on the storage and transmission capacity of today's workstation environments. A typical workstation in 1993 might have on the order of 500 MB to 1 GB of disk space and be connected to a 1 Mbit/s to 10 Mbit/s local area network. Even as storage densities continue to improve, users will always be straining the limits of their storage capacities as use of video images gain in popularity. Compressing these rates down will be key in allowing interactive retrieval of video and communicating via pictures through video mail and teleconferencing. Compression requires computation, however, with standards such as MPEG being quite computationally intensive for good compression while maintaining reasonable quality during decompression.

**High Definition TeleVision (HDTV)**, an emerging standard for high resolution digital television systems, is one area where very fast computation of these compression algorithms will be essential in delivering live broadcast programs. Today's fastest workstation class systems would require hundreds of hours of processing time to compress even a half hour
CHAPTER 1. INTRODUCTION

<table>
<thead>
<tr>
<th>Picture resolution</th>
<th>Bandwidth required</th>
<th>Storage needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>352x240</td>
<td>30.4 Mbits/s</td>
<td>13.7 GB</td>
</tr>
<tr>
<td>640x480 (NTSC)</td>
<td>110.6 Mbits/s</td>
<td>49.8 GB</td>
</tr>
<tr>
<td>1280x720 (HDTV)</td>
<td>331.8 Mbits/s</td>
<td>149.3 GB</td>
</tr>
</tbody>
</table>

Table 1.2: Requirements of a 60-minute full-motion digital video sequence (12 bits/pixel).

of video at these higher resolutions.

1.2 Thesis Organization

While MPEG is a powerful method for reducing the amount of data needed to convey a video image, it is computationally intensive to produce video compression retaining modest image quality. This thesis establishes a framework for a parallel processing approach to this proposed industry standard for video compression from MPEG.

This thesis is organized into the following main sections. Chapter 2 outlines the approach to video compression as presented in the MPEG standard. A description of the NuMesh architecture on which this work is based is given in Chapter 3. Chapter 4 details the approach taken in implementing this standard using the NuMesh architecture. Chapter 5 presents a performance analysis of the system showing speedups under various configurations and tradeoffs. Conclusions are summarized in the final chapter where follow on work is also discussed.
Chapter 2

The MPEG Compression Standard

2.1 Overview of MPEG

MPEG, which stands for Motion Picture Experts Group, is a subcommittee of the International Standards Organization and is a body setup to recommend a standard for motion picture compression. This standard, commonly known as MPEG I, was made an international standard in the summer of 1993 after existing in draft form since 1991[6]. This standard specifies the format and acceptable methods for compressing images and accompanying audio but is flexible enough to allow for various conforming implementations that achieve varying levels of compression and image quality. It is designed for systems able to handle up to about 1.5 Mbit/s. These implementations can vary in software as well as hardware and should generate data that is readable by any conforming MPEG player. Since this thesis only deals with the video portion of the standard, only that part of the standard will be described.

Each frame of an incoming image stream from a video capture stage is subdivided into 8 by 8 pixel blocks where pixels have been converted from an RGB (red, green, blue) triplet into a YCrCb triplet. In this color space, the Y value contains the luminance (degree of brightness) information and the Cr and Cb values contain chrominance information. The human eye is less sensitive to chrominance and more sensitive to luminance information.
thus the Cr and Cb values are subsampled by a factor of 2 along each axis. Therefore, in going from RGB to YCrCb, an 8 by 8 block contains 64 Y samples but only 16 each of Cr and Cb samples. These transformed blocks of pixels are fed into the stages of the MPEG encoder model and at the output emerges a compressed video bitstream.

### 2.2 MPEG Compression Model

The MPEG model, shown in Figure 2-1, consists of five stages: a motion compensation stage, a frequency transformation stage, a lossy quantization stage, and two lossless coding stages. The motion compensation stage attempts to use information in previous or future frames in an effort to reduce the number of bits required to reproduce a given frame. The frequency transform shifts the view of the video data to a domain that allows for a more compact representation. Quantization causes a controlled loss of information by attempting to cut out high frequency artifacts that the human eye cannot discern in moving pictures. The two coding stages further compress the data using run-length and Huffman coding techniques to efficiently code the output of the other stages in the fewest number of bits.

A typical encoded MPEG sequence consists of three frame types as depicted in Figure 2-2: frames that act as references for determining temporal based compression called **intraframes** (I), frames that are predicted by forward motion (P), and frames that are bidirectionally predicted (B). I frames skip the motion compensation stage. P frames contain a mixture of forward predicted and intracoded blocks according the criteria outlined below. B frames contain a mixture of forward, reverse, interpolated or intracoded blocks.

First an I frame is coded followed by a forward prediction frame from the first frame. Finally B frames are interpolated from these two frames. This prediction process repeats itself with a variable number of forward and bidirectional frames between successive intraframes. The relative number of the different frame types is left open to the encoder. This information which will be needed by the decoder is passed along in the header for the sequence. A balance must be struck between I, P, and B frame types. While B frames have the most potential to yield high compression, too infrequent I and P frames lead to poor matches as the scene is changing from one frame to the next. Ideally, the encoder would
Figure 2-1: Model of the MPEG encoder and decoder subsystems.
2.3 Motion Compensation Stage

The motion compensation stage is where a great deal of the compression gain is achieved in the MPEG model. This gain is achieved based on the observation that there is temporal locality in a video sequence, meaning that consecutive frames will have much in common with each other. Quite often in going from one frame to the next, an area in the image is a shifted version from other frames in time. MPEG is able to take advantage of these cases by calculating this shift as motion vectors. These motion vectors and any small amount of error is all that is needed to reconstruct this area of the image. For example, the video sequence might be a camera pan of a fixed scene or of a person or object moving against a constant background. Motion estimation is calculated on a special grouping called a macroblock,
which is composed of 4 Y blocks and one each of the subsampled Cr and Cb blocks. A search is performed comparing each macroblock where motion compensation is to be used to a reference frame from the past (for P frames) or future (for B frames). To simplify the search, only the luminance components of the macroblocks are used for comparison[8].

The determination of these motion vectors is the most computationally challenging task of the MPEG standard but yields high compression ratios without suffering loss of quality in the decompression of the image. A search area is designated depending on the frame type in an attempt to find a macroblock from this area that closely resembles the macroblock to be coded. The mean absolute difference (MAD) is computed on each pass through the search area to find the best match and is expressed by,

$$MAD = \frac{\sum_{i=0}^{31} \sum_{j=0}^{31} |R(i,j) - C(i,j)|}{32 \times 32}$$ (2.1)

where R is the target in the reference frame used for comparison to determine a match for C, the macroblock to be coded, where each 4 Y block region is 32 by 32 pixels in size. R is selected from a search area that can vary in size and is typically 8 to 16 pixels larger on each side than a macroblock. One approach is to start R at the same horizontal and vertical position as C, calculate its MAD, and then shift R by 1 or ½ pixels and recalculate until the search area is covered. The best match is the reference macroblock in the search area that produces the smallest MAD value. If the “best match” yields no better compression than encoding the original by itself, the match is rejected and the macroblock is coded as an I type instead. The test for determining whether better compression could be achieved looks at the variance in the error, the difference between the two macroblocks constituting the best match, and the variance in the original macroblock to be coded. Generally, the higher the variance the more the transform coefficients will be spread out leading to poorer compression.

2.4 Transformation Stage

Blocks of pixels are passed through a frequency transformation stage. This transformation concentrates the energy of a spatially slowly varying image into fewer coefficients. The
**CHAPTER 2. THE MPEG COMPRESSION STANDARD**

Figure 2-3: Two dimensional DCT coefficients arrangement.

The transform method chosen is the discrete cosine transform which performed on a block can be expressed by the following formula:

\[ F(u, v) = \frac{1}{4} C(u)C(v) \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j) \cos((2i+1)\frac{\pi}{16}) \cos((2j+1)\frac{\pi}{16}), \quad (2.2) \]

where \( F(u, v) \) are the frequency domain transform coefficients, \( f(i,j) \) are the pixel values, and

\[
C(a) = \begin{cases} 
\frac{1}{\sqrt{2}} & \text{if } a = 0 \\
1 & \text{otherwise}
\end{cases}.
\quad (2.3)

The two-dimensional DCT output is ordered with the mean value (DC coefficient) in the upper left corner and then progressively higher frequency coefficients are arranged in a zigzag pattern from upper left to lower right as depicted in Figure 2-3. The result of the DCT is a sparse 8 by 8 matrix. More zeros is beneficial for the run-length encoding stage. The largest benefit stems from the human visual system’s differing sensitivities to frequency ranges which is taken advantage of in the quantization stage[15].
CHAPTER 2. THE MPEG COMPRESSION STANDARD

2.5 Quantization Stage

Quantization of the DCT coefficients is the source of lossiness in MPEG compression. The coefficients are quantized to increase the number of zero value terms in places where the coefficients are close to zero. Additionally, since the human eye is more sensitive to slowly varying intensities, more weight can be given to the DC and low frequency terms and the high frequency terms can be more coarsely quantized. A uniform quantizer is used for MPEG, with a different stepsize per DCT coefficient position. The MPEG committee recommendation for stepsizes is shown in Figure 2-4. P and B frames use equal default step sizes of 16 since the visual weights of motion compensated blocks tend to be more evenly distributed. The degree of quantization can be varied to achieve different levels of compression while trading off image quality in the process. In fact, a different constant scale factor can be used for each of the three frame types which are given as examples in the standard. Quantizing I frames less than B frames is usually beneficial since I frames act as references. A highly quantized set of DCT coefficients lead to greater compression gains but poorer image quality during decompression.

2.6 Coding Stage

Two lossless methods are used to further compress the image representation once the DCT coefficients have been quantized and scaled properly. The first method uses a run-length
coding technique to compact the large numbers of zeros present in the AC coefficients. The second method uses the Huffman entropy coder[5]. Huffman codes are arranged by frequency of use with symbols that are most repeated in the source given the shortest code. A codebook is constructed to best take advantage of redundancy in the data stream.

2.7 MPEG Summary

These stages combine to reduce to a minimum the amount of temporally and spatially related data needed to recreate the image. Note that the image will not be an exact duplicate of the original by nature of the DCT and quantization effects but the goal is to be reasonable close so that slight defects are not noticeable to the observer. The MPEG system specifies the means for degrading image reproduction to achieve better data compression for transmission over limited bandwidth media. Achieving good performance is a computationally challenging task which makes the MPEG system a suitable target for a high performance computing environment such as the NuMesh.
Chapter 3

The NuMesh Project

3.1 Introduction

The NuMesh Project, initiated by the Computer Architecture Group at MIT, aims at establishing a flexible yet fast and powerful system for communication between processors in a parallel processing system[19]. The goal of the NuMesh initiative is to standardize a scalable, modular communications substrate to support communication between processors in a three-dimensional topology. Each communication module interfaces to a processor and its six nearest neighbors to form a three-dimensional mesh of processors. These modules are programmable and control the flow of information through the mesh. These modules each have multiple ports and plug together in any configuration and size that the user requires. A novel topology under development connects together four neighbors in three-dimensional space resembling the lattice structure of a diamond.

The key to these units being modular and scalable is resolving such issues as power distribution and establishing the interconnectivity protocol. NuMesh will provide a standard platform for developers of parallel processing systems that allow the designer to not have to resolve the nontrivial problems of designing a new communication scheme and instead be able to provide a new hardware product that can be added to an existing user's system, similar to developing a card to plug into a standardized backplane bus. The user would have greater flexibility to customize the computing environment to fit her needs. For example, users can create a system to match their current computing needs and then selecting from
Figure 3-1: NuMesh Prototypye Modules.

a range of interoperable processing and IO products easily upgrade their system by simply plugging into place the desired NuMesh modules as if building with Lego blocks. Also this connectivity flexibility allows for reconfiguration of the network to better suit a particular application's communication and work distribution requirements.

Communication between nodes is handled by a programmable traffic routing engine. Its strength lies in its ability to route data at runtime without having to read it based on communications requirements determined at compile time, a technique termed static routing.

3.2 Two Dimensional Prototype

In order to begin to be able to test out the feasibility of such a system a two dimensional prototype using off-the-shelf TTL and CMOS parts has been built [10, 3]. The most recent prototype consists of five interface ports: four to connect to four neighbors and one to connect to a processor element. Figure 3-1 shows a view of this two-dimensional prototype. Each prototype module is composed of a communication finite state machine (CFSM) that sequences communication between nodes and the local processing element.

Each NuMesh node has two 32-bit buses that span between the north and south ports
and the east and west ports. These two independent buses allow for two simultaneous transactions to take place. A registered transceiver permits transactions between buses. This setup allows for the most efficient communication paths to be straight through the node to the other side and imposes a penalty for turning corners. With “manhattan” routing of information on a two-dimensional mesh, communication between two nodes would require at most turning one corner.

3.3 Processing Elements

Communications with a node’s attached processor board occurs through two sets of FIFO memory. The FIFOs operate in two directions. One direction goes from the north-south bus towards the processor and is called the Up FIFO. The other direction transfers data from the processor to the east-west bus via the Down FIFO.

One processing element designed for this initial prototype is based on the Texas Instrument TMS320C30 which is a digital signal processor. This processor board is described in detail in [1]. A processing node based on a SPARC microprocessor has also been built. This prototype has been very successful in showing that the goals of the NuMesh project are realistic and its future promising. Benchmark testing with twelve nodes and digital signal processing boards showed that there is a linear speed up in performance as each node is added to the mesh for a class of applications.

Along with processing nodes, a frame buffer node capable of capturing two 512x512 frames of data and a SCSI compatible hard disk drive allow for significant storage capacity within a NuMesh system.

3.4 Applications

Each node runs at 40MHz and the system achieves supercomputer performance with only a few modules for certain applications. The prototype system includes a Macintosh interface that allows the hardware to be controlled and run from a NuBus slot connection that makes testing and debugging much easier. Through this interface several test applications have been developed and run such as the traveling salesman problem and a graphical spectral
Figure 3-2: NuMesh Data Paths.
CHAPTER 3. THE NUMESH PROJECT

analysis program[17, for details on the interface]. An SBUS interface has also been built to allow a Sun workstation to act as host.

The NuMesh prototype is at a stage where diverse applications need to be developed to explore its strengths and weaknesses. One of the goals of this thesis will be to explore these areas and offer an opportunity to explore improvements in the communications substrate.

3.5 NuMesh Simulator

Designing and debugging code for NuMesh can more easily be accomplished from a UNIX workstation using the NuMesh simulator[11]. The simulator accepts C programming language code[14]. The simulator then steps through both processor and communication code and gathers statistics. This environment is extremely useful since runtime debug capabilities are limited to a small character string display on the processors and status light-emitting diodes on the CFSMs.

3.6 Prototype Software Environment

A graphical user interface exists for communicating with the prototype mesh from a connected host workstation. This interface explores the mesh to determine its connectivity as well as to identify the processor types. It handles the bootstrapping of the nodes and prepares them to receive program code.

3.7 Summary

The NuMesh environment provides the flexibility to be adapted to most any task. It provides a heterogeneous processing capability and is scalable by one processor at a time. Another big strength of NuMesh is its ability to do static routing for low overhead communication for precompilable patterns. These features all proved to be extremely useful in the development of the MPEG compression system.
CHAPTER 3. THE NUMESH PROJECT

; Code to send all data arriving from the east to the processor
; and all data from processor back to east

start
()

(case ifulls
  (east (drivee loadxew)) ; Check incoming port flags
   ; If east present then take
   (else (goto nextcheck))) ; otherwise don’t
  (drivexns loadup) ; Send to processor interface
  (drivexns loadup) ; takes 2 cycles

nextcheck
()

(case ccmisc ; Check FIFO full/empty flags
  (empty (goto start)) ; If empty try east again
   (else ()))
  (drivedown loadxew) ; Send data to crossover transceiver
  (drivedown loadxew)
  (drivexns loadxns) ; Turn data around
  (drivexew loade) ; Send to east
()
(goto start)

Figure 3-3: A simple example of CFSM code.
Chapter 4

Concurrent Video Compression

This chapter outlines the methodology of the concurrent video compression system developed for the NuMesh. As described previously, the MPEG standard is comprised of many components that can be subdivided into individual tasks. If performed on a conventional one processor system, these tasks would necessarily be serialized even though there might not be a dependency between them. Identifying tasks that can proceed at the same time is the first step towards concurrent processing on a multiprocessor system. Since a digitized image can be subdivided into smaller components, the processing time can be overlapped by duplicating the encoding engines that are assigned to the task. The encoding engines are processors programmed to execute the encoding phase of MPEG. The work to be done is distributed in some fashion among these encoding engines.

4.1 Ways of Benefitting from Concurrency

The general approach to achieving speed up in implementing the encoding of an MPEG video stream can be summarized as follows:

- Pipelining: divide the tasks to be done into discrete stages that the data stream flows through such that a different part of the computation is done at each stage. This technique is used in RISC microprocessors to improve throughput. In this way, each stage can be computed on a different processor.
CHAPTER 4. CONCURRENT VIDEO COMPRESSION

- Parallelization: duplicate these pipelined stages so that different parts of the image will be worked on by different processors.

- Spatial locality: perform as few operations that require access to all the data in an image as possible and partition the data to minimize communication between physically distant nodes.

- Precompiled routing: determine communication patterns at compile time and generate efficient code for the programmable communication hardware to minimize time spent by the data moving through the system at runtime[12].

These methods work well together to improve performance. In fact spatial locality makes scheduling communication easier since the burden on the network is lessened. Partitioning the tasks on a 2D machine is fairly straightforward as can be seen by overlaying a 2-dimensional mesh topology over the task graph in Figure 4-1 as depicted in Figure 4-2.

4.2 Tradeoffs in Achieving Concurrency

While there are clear benefits to splitting up MPEG tasks and executing them concurrently, multiprocessor systems have costs associated with trying to achieve concurrent execution. The most critical cost is in interprocessor communication. The network that allows processors to communicate with one another can become a critical resource and slow down processors if they must wait to receive data to continue a computation. Looking at the data requirements of the various stages of MPEG will show whether this issue is important for performance.

Starting with the DCT, only 64 adjacent pixels are needed to compute each DCT. Only the motion estimation portion needs access to blocks to neighboring blocks from a previous or later frame. The compression system has a limited search area in this implementation to ± 15 pixels. To satisfy this search area would require a 46×46 pixel area for each prediction direction. For B frame types, each macroblock needs 18 macroblocks of forward and reverse history for the best match comparisons. Since these regions will overlap for neighboring macroblocks, the cost of saving these regions in local memory can be amortized over several macroblocks.
Figure 4-1: One possible division of the MPEG encoder.
Figure 4-2: Partitioning of the MPEG encoder tasks on a 2D mesh.
CHAPTER 4. CONCURRENT VIDEO COMPRESSION

One issue that gets raised is how this history should be maintained and distributed. One possible ordering would be to do the I frame blocks that would be needed by any B frame blocks in each processor. This way would lead to quite a bit of redundant computations potentially as several overlapping blocks would be recalculated by each processor. The same could be done for the P frame types once the I frames have been calculated.

Another possibility would be for each block type to be computed only once and then copies sent to neighboring processors as needed. Since the granularity of the computations is at the macroblock level, only the peripheral macroblocks contained at each node need to be sent to neighbors. This method is preferred since it is much easier to duplicate blocks than to recompute their DCT value. Each processor passes along the corresponding outside edge blocks to its neighbor after computing the DCT for each of these blocks. Processors at most 2 hops would need to be reached with this strategy using a 2-dimensional NuMesh topology to reach diagonal nodes that would need one block each.

In summary, NuMesh’s current topology is well suited to data distribution requirements as described above. The critical performance issue is maximizing the efficiency of each processor’s code. The performance of various parts of the MPEG implementation is explored in the next chapter.
Chapter 5

Performance Analysis

5.1 Introduction

This chapter covers the analysis of the results for the previous chapter. Projections are made based on the preliminary results of implementing the MPEG standard using NuMesh. The questions answered here include: How much processing resources do various sections of the algorithm require? How does the performance change as the number of processors in the system increase? Does the topology of the network have any effect on performance?

Other issues include the partitioning of the program among processors and the amount of memory and network bandwidth required. In essence, the sources of potential bottlenecks are explored.

5.2 Processing Time

Processor requirements for performing an 8x8 DCT are approximately 2kOPs (1 OP = 1 machine level instruction) broken down as follows: 896 adds, 1024 multiplies, plus some startup overhead. To perform the quantization required is another 1kOPs and for the encoding add 1kOPs more. Therefore, for one YUV (4:1:1) frame (1980 blocks) to be compressed as an I frame would require 11 MOPs. For a 30 frame per second video source, this conversion translates into a rate of 320MOPs per second. Assuming 40 MOPs per second processors and hidden cost communication, eight processors would be required to sustain a
realtime processing rate. Hidden cost communication means time spent transmitting data over the network between processors is overlapped with computation and so has a negligible effect on the overall processing time.

However, I frames are not the most compute intensive frame types. The P and B frame types are significantly more costly to compute since they require comparisons and best match searches for determining motion based compression. To calculate a P frame using a full search algorithm for motion estimation in a \( \pm 8 \) pixel range would take approximately 75 MOPs. To convert every incoming frame into a P frame would require 1700 MOPs per second of processing which translates 42 processors making the same assumptions as above. B frames work out to be roughly twice this requirements since both a forward and reverse comparison need to be made.

Taking a look at a more realistic scenario, where the frame type mix is something like IBBPBBPBPPBB... with 2.5 I frames, 7.5 P frames, and 20 B frames per every 30 frames, the number of MOPs consumed respectively is: 26.67, 425, and 2267. Summing these MOPs yields a requirement of 2718 MOPs per second to keep pace with a 30 frame-per-second video stream. This rate can be achieved with roughly 68 processors running at 40 MOPs each. The fastest of today’s processors operate in the 100-200 MOP/s range which even at the high end of that range is a factor of 13 too slow to be handled in real time by one processor\(^1\).

The above computations all made one key assumption that will now be examined. This assumption deals with the cost to performance for handling the communication required for a large system of processors. As discussed previously, the MPEG standard does not require access to all data at any one time, thus making it well suited for a distributed system. However, an important issue is how video data from an external source is delivered to these processors as needed. Since data will not need to be passed along from one processor to a processor very far away, the overhead for communication and the contention for network resources will be much lower than a program that does not exhibit this spatial locality. In general, if the ratio of computation to communication is large and the number of hops for

\(^1\)Although an OP is not necessarily the best metric for determining processor performance, it is used here to simplify the discussion.
CHAPTER 5. PERFORMANCE ANALYSIS

communication is small then the cost of communication will not have a huge impact on performance. The performance will be largely a function of how long it takes to compute the stages for each processor.

The preceding performance calculations assume that each processor does the entire algorithm on equally divided portions of the frame. Under this assumption all frame data required by each processor must be present in its local memory. The local memory requirements under these conditions scales downwards as the number of processors increases since a smaller portion of the image is handled by each processor as Table 5.2 shows for several configurations\(^2\). In the NuMesh prototype the incoming video can be delivered to the processors from two possible sources: the workstation host via the SBUS interface, or a SCSI disk drive node.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>10</th>
<th>15</th>
<th>30</th>
<th>66</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacroBlocks/processor</td>
<td>33</td>
<td>22</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Local Memory (in kB)</td>
<td>66</td>
<td>44</td>
<td>22</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5.1: Local memory as a function of the number of processors.

To examine the effect of communication overhead using NuMesh to implement MPEG, a 100 node system, affording ample processing capabilities, has several topological characteristics that reduce this overhead. Assuming a contiguous square arrangement of the nodes forming the mesh, the longest minimal path between any two nodes requires 18 hops or node crossings to go diagonally from one corner to the opposite one. Therefore, if it takes 5 cycles per flit (32 bits) to cross one node, a message containing 2500 flits will take 225,000 communication cycles to traverse this longest distance. However, these communication cycles can be hidden by the millions of processor cycles so as not to affect the throughput of the system. These communication costs will introduce extra latency in generating an output stream but really its throughput, the ability to maintain a continuous output stream, that matters if this output is to be fed directly into a decoding stage via some transmission media. Because the communication patterns are very simple, deterministic, and recurring,

\(^2\)In computing these values instruction memory is not counted which would add a constant amount per processor to the total memory requirements regardless of the number of processors.
a programmable communication substrate as found in NuMesh can be taken advantage of. Another point on topology is that a square arrangement of the nodes is clearly a better arrangement than a long string of nodes in order to be most efficient in distributing the data and collecting results.

5.3 Summary

By calculations taken from analyzing the code and taking into account processor performance, a NuMesh system on the order of 70 TI-C30 DSP processors is able to create an MPEG compressed video stream in real-time. Over 30% of time is spent on computing motion based compression with the interpolated (B) frame types taking almost twice as long as the other two to compute. In terms of supporting communication between processors, systems able to handle simple recurring patterns are well-suited to this task. NuMesh is able to support such patterns easily through its programmable communications substrate.
Chapter 6

Conclusion

6.1 MPEG encoder using NuMesh

This thesis work has established that MPEG video compression is a task suited to parallel implementation based on benchmarking of the NuMesh prototype system. NuMesh's programmable communication architecture has proven to be very valuable for the flexibility and low overhead data transfers required in implementing MPEG. This result is very encouraging for future development efforts of NuMesh.

6.2 Applications and Improvements

Being able to greatly reduce digital video data requirements will continue to spur many useful applications in scientific research, multimedia education and entertainment, global telecommunications, and many other areas. Continued improvements in high performance systems will be key in this effort. In the case of the MPEG standard, requiring everyone to have a 70-processor parallel system is clearly infeasible. Initially, a handful of such systems might be employed to generate these compressed video streams for broadcast. Another standard, Px64, is in place for one-to-one types of communication where both ends need to compress and decompress a video stream.
6.3 Future Work

While this thesis has focused on the MPEG I standard, the general approach is extensible to improvements on this standard. An MPEG II standard is already in progress targeted at higher bandwidths. Furthermore, research carried out using this architecture could lead to recommendations for improvements to better suit parallel environments. The NuMesh system is undergoing a revision in its communication engine design which will enhance any NuMesh based application. Along with these alterations will come a large scale implementation of NuMesh. Several additions to the NuMesh system, outlined below, which are on the horizon, will greatly improve the MPEG system described in this thesis.

6.3.1 Video Source Hardware

Currently, the input/output devices available for the NuMesh system are very limited. Work is under way to develop a video hardware addon to allow live video input and output directly to NuMesh.

6.3.2 MPEG Audio

The audio portion of the MPEG standard was not a part of this thesis. While the processing requirements for the audio component are only a fraction of that required for doing video, a way must be found to incorporate this into the system or if handled separately then a way to synchronize and merge the two streams must be devised.

6.3.3 Mass Storage

The storage capacity for the NuMesh system will be greatly expanded with the addition of more hard disk drives. These array of drives will be useful as a place to deposit MPEG streams for retransmission as a later time.
Appendix A

MPEG Code Excerpts

The code for a NuMesh MPEG implementation can be categorized into three groups: DSP code, host code, and CFSM code. The bulk of the code is DSP code which runs on the processors and is written in C. The host code, also in C, runs on the Sun workstation connected to the mesh and controls access to it. All data into and out of the mesh flows through this interface. The raw video stream is injected into the mesh from the host and the compressed result is collected and written to a file. The CFSM code written in a LISP-like syntax controls the timing of communication within the NuMesh system at runtime.

`mpegmain.c`: configuration for either processor or simulator execution. Entry point into the main stages of MPEG. A pointer to a struct containing the various parameters including memory locations of blocks of data is passed around to all the major procedures.

```c
/* NuMesh Processor code (targeted to C30 but portable) */

struct Mpegparams {
    int nodeNumber;
    FrameSlice *currFrame;
    FrameSlice *backRefRawFrame;
    FrameSlice *forwRefRawFrame;
    FrameSlice *backRefDCT;
    FrameSlice *forwRefDCT;
    int frameType; /* Intra=0, Predicted=1, Bidirectional=2 */
    char *framePattern;
    int startSliceX;
    int startSliceY;
    int endSliceX;
    int endSliceY;
    int motionSearchRange;
    int currFrameNumber;
    int totalFrames;
    int *fwdMotion, *backMotion;
    int *lastFwdMotion, *lastBackMotion;
    int
```
APPENDIX A. MPEG CODE EXCERPTS

};
#ifdef NSIM
cpu_code()
#else
main()
#endif
{
    struct Mpegparams *params; /* Keep all processor specific params here */
    MacroBlock *mblock;
    register int currX, currY, maxSliceX, maxSliceY;
#ifdef _TMS320C30
dsp_init();
#endif
#ifdef _SPARC
sparc_init();
#endif
/* First thing we do is to send dummy word to cfsm to tell it we are up and running. */
write_mesh(0xa5a5a5a5);
init_mpeg_params(params); /* Configures settings sent by host */
currX = params->currX;
currY = params->currY;
maxSliceX = params->maxSliceX;
maxSliceY = params->maxSliceY;
check_data_type();
for(;currX<=maxSliceX;currX++) {
    for (;currY<=maxSliceY;currY++) {
        get_Y_macroblock(params);
        get_Umacroblock(params);
        get_V_macroblock(params);
        if ((params->frameType == "iFrame") || (fwdBlock == NULL))
            mpeg_iframe(params);
        if ((params->frameType == "pFrame") || (prevBlock == NULL))
            mpeg_pframe(params);
        if (params->frameType == "bFrame")
            mpeg_bframe(params);
    }
    if (params->doEncode)
        huffman_encode(mblock);
    else
        put_block(mblock);
    next_frame_type(params);
}
void mpeg_iframe(params) /* I frame routine */
struct *Mpegparams params;
{
    if (params->doDCT)
        mpeg_dct(mblock);
    if (params->doQuant)
        mpeg_Iquantize(block);
}
void mpeg_pframe(params) /* P frame routine */
struct *Mpegparams params;
{
    if (params->doFwdEstimation)
        fwd_motion_estimation(mblock,params);
    if (params->doCompensation)
        motion_compensation(mblock);
    if (params->doDCT)
        mpeg_dct(block);
    if (params->doQuant)
        mpeg_Pquantize(block);
void mpeg_bframe(params) /* B frame routine */
struct *Mpegparams params;
{
    if (params->doFwdEstimation)
        fwd_motion_estimation(mblock, params);
    if (params->doBackEstimation)
        back_motion_estimation(mblock, params);
    best_motion_vector(params);
    if (params->doCompensation)
        motion_compensation(mblock, params);
    if (params->doQuant)
        mpeg_Bquantize(block);
}

streamio.c: Host file handling functions.

//******************************************************************************
* Contains: stream_frame_segment(fileptr, paramsptr)                  *
* determines how big a chunk of data to inject                          *
* into the mesh before going on to the read phase                        *
* collect_mesh_output(fileptr, paramsptr)                               *
* collects samples from mesh and unpacketizes as                        *
* required and stuffs in a file for continued processing                *
//******************************************************************************
void stream_frame_segment(infile, params)
    File *infile[];
    struct mpegParams *params;
{
    int count;
    Datum pixel;
    printf("Blasting stream of video bits into the mesh pipeline.\n");
    /* NOTE nsm may want us to tell which cfsm we're writing to at this 
     * stage. Check to see if this can be specified elsewhere. */
    for (count=CHUNKSIZE(params);count;count--)
    {
        pixel = read(infile[CURRFRAME(params)]);
        if (pixel != EOF)
            PutMesh(IONODE, pixel);
        else
            error("Unexpected end of file while reading input file: %s\n",
                infile[CURRFRAME(params)]);
    }
    if (CURRCHUNK(params) == ENDCHUNK(params))
    {
        CURR_FRAME(params)++;
        CURR_CHUNK(params) = 0;
    }
    /* Framesize must be an integer multiple of selected chunksize. */
}

void collect_mesh_output(fileptr, params)
    File *fileptr;
    struct mpegParams *params;
{
    int count, numMacroBlock, currProcNum, MBlockCount;
    static long totalFlits=0;
    static int numProcs;
    int MBlocksReceived=0;
    Datum flit;
    static mpegFrame *frame;
    static int *procArray[]; /* table of processor->Mblock association */
    numProcs = NUM_PROCS(param);
frame = (int *)malloc(MAX_MB_OUT(params)*NUM_MBLOCKS(params));
procArray = (int *)malloc(numProcs*NUM_MBLOCKS(params));
while(MBlocksReceived < NUM_MBLOCKS(params)) {
    flit = GetMesh(IONODE);
    numMacroBlock = MACROBLOCK(flit);
    currProcNum = CURR_PROC(flit);
    currMBptr = procArray[currProcNum][0];
    procArray[currProcNum][currMBptr] = numMacroBlock;
    numFlits = GetMesh(IONODE);
    procArray[currProcNum][currMBptr+1] = numFlits;
    frame[numMacroBlock][0] = numFlits;
    totalFlits += numFlits;
    for(count=1; count <= numFlits; count++)
        frame[numMacroBlock][count] = GetMesh(IONODE);
    MBlocksReceived ++;
}
for(MBlockCount = 0; MBlockCount < NUM_MBLOCKS(param); MBlockCount++) {
    for (count=0; count < frame[MBlockCount][0]; count++)
        write_file(outfile, "%d", frame[MBlockCount][count]);
}

dspdct.c: Code for DCT only.

/**************************************************************************
 * NuMesh processor node code for doing 8x8 DCT computation
 * Takes chunks of 64 words into memory performs DCT then transfers
 * back to FIFOs.
 **************************************************************************/
#define BLOCKSIZE 8*8 /* Each frame block is 8 by 8 pixels */
#define DCT_SOURCE HOST
#define DCTSINK HOST
char nodeNames[NUMNODES+1][80];
int *Data;
InitNode()
{
    #if NODE<=NUMNODES
        register struct Complex *p;
        register kount;
        #ifdef _TMS320C30
            asm(" OR 0800H,ST ; Enable TMS320C30 cache");
            Data = (int *) 0x809800;
        #else
            char *malloc();
            Data = (int *) malloc(sizeof(int)*NUMMACROBLOCKS);
        #endif
        for (kount=1; kount<=NUMNODES; kount++)
        {
            strcpy(nodeNames[kount], "Node");
            nodeNames[kount][4]=#kount;
            nodeNames[kount][5]=0;
        }
        NodeID(nodeNames[NODE]); /* specify node name */
    #if NODE==1
        PutMesh((long) NUMNODES, HOST); /* Tell host how many nodes */
    #else
        PutMesh((long) NODE, HOST); /* Check in with host, signal done */
    #endif
    strcpy(DispBuffer, " READY ");
}
MainLoop()
{
    int *Data;
register counter;
Data = malloc (BLOCKSIZE*sizeof(int));
display ("DCT reading");
NumBlocks = GetMesh(DCT_SOURCE);
while (!NumBlocks) {
  for(counter=0;counter<BLOCKSIZE;counter++)
    Data[counter] = GetMesh(DCT_SOURCE);   /* Read 1 block of values */
dsp_fwd_dct(Data);                       /* Call DCT routine */
display ("Computing DCT");
  for(counter=0;counter<BLOCKSIZE;counter++)
    PutMesh(OutData[counter], DCT_SINK);  /* Write DCT to mesh */
}

fwddct.asm: Assembly output from code adapted from the Independent JPEG Group's public domain distribution by Thomas Lane.

*****************************************************************************
* TMS320C30 C COMPILER  Version 4.30
*****************************************************************************
; ac30 -DSHORTxSHORT_32 jfwddct jfwddct.if
; opt30 -m -s -D2 jfwddct.if jfwddct.opt
; cg30 -m -n jfwddct.opt jfwddct.asm jfwddct.tmp
.version 30
FP .set AR3
.globl _memchr
.globl _memcmp
.globl _memcpy
.globl _memmove
.globl _memset
.globl _strcat
.globl _strchr
.globl _strcmp
.globl _strncpy
.globl _strcoll
.globl _strcsnp
.globl _strerror
.globl _strlen
.globl _strlcn
.globl _strncat
.globl _strncpy
.globl _strpbrk
.globl _strrchr
.globl _strspn
.globl _strstr
.globl _strtok
.globl _strxfrm
.globl _jpeg_compress
.globl _j_c_defaults
.globl _j_monochrome_default
.globl _j_set_quality
.globl _j_add_quant_table
.globl _j_quality_scaling
.globl _jpeg_decompress
.globl _j_d_defaults
.globl _j_fwd_dct
.globl _j_rev_dct
.globl _j_round_up
.globl _j_copy_sample_rows
.globl _j_copy_block_row
.globl _jzero_far
.globl _jselectpipeline
.globl _jselect Huffman
APPENDIX A. MPEG CODE EXCERPTS

.globl _jselcarithmetic
.globl _jselexpand
.globl _jseldownsampling
.globl _jselmcu
.globl _jselccolor
.globl _jselrgif
.globl _jselrppm
.globl _jselrle
.globl _jselrtarga
.globl _jselwjfif
.globl _jseldpipeline
.globl _jseldhuffman
.globl _jseldarithmetic
.globl _jseldmcu
.globl _jselbsmooth
.globl _jselupsampling
.globl _jseldcolor
.globl _jseliquantize
.globl _jsel2quantize
.globl _jselrjif
.globl _jselgif
.globl _jselwpmm
.globl _jselwrle
.globl _jselwtarga
.globl _jselerror
.globl _jselwrle
.globl _jselmemmgr
.globl _j_fwd_dct

* FUNCTION DEF : _j_fwd_dct

**--------------------------------------------------------

.j_fwd_dct:
PUSH FPLDI SP,FP
ADDI 4,SP
PUSH R4
PUSH R5
PUSHF R6
PUSHF R7
PUSH AR4

* R2 assigned to variable 'tmp4'
* R3 assigned to variable 'tmp6'
* R4 assigned to temp var 'C$12'
* R4 assigned to variable 'tmp10'
* R4 assigned to temp var 'C$24'
* R4 assigned to variable 'z1'
* R5 assigned to temp var 'C$14'
* R5 assigned to temp var 'C$17'
* R5 assigned to temp var 'C$9'
* R5 assigned to variable 'z3'
* R5 assigned to temp var 'C$4'
* R5 assigned to variable 'tmp12'
* R5 assigned to temp var 'C$15'
* R6 assigned to variable 'tmp5'
* R7 assigned to temp var 'C$13'
* R7 assigned to temp var 'C$10'
* R7 assigned to variable 'tmp13'
* R7 assigned to temp var 'C$22'
* R7 assigned to temp var 'C$25'
* R7 assigned to variable 'z4'
* AR2 assigned to variable 'data'
* AR4 assigned to variable 'dataptr'
* IR1 assigned to temp var 'C$8'
* IR1 assigned to temp var 'C$26'
* IR1 assigned to variable 'tmp11'
* IR1 assigned to variable 'z5'
* BK assigned to temp var 'C$11'
APPENDIX A. MPEG CODE EXCERPTS

* BK assigned to temp var 'C$23'
* BK assigned to temp var 'C$21'
* BK assigned to variable 'z2'
* BK assigned to temp var 'C$16'
* BK assigned to temp var 'C$3'
* RC assigned to temp var 'L$1'
* RC assigned to temp var 'L$2'
* --- 'C$20' shares AUTO storage with 'C$5'
* --- 'C$19' shares AUTO storage with 'C$6'
* --- 'C$18' shares AUTO storage with 'C$7'
*
LDI **FP(2),AR2  
*** 15 ------------------------ dataptr = data;
LDI AR2,AR4  
*** -------------------------- L$1 = 7;
LDI 7,RC  
*** -------------------------- g2:
*** --------------------------g12:
RPTB L8  
*** 18 ---------------------- C$24 = dataptr[7];
LDI *AR4(7),R4  
*** 18 ---------------------- C$23 = *dataptr;
LDI *AR4,BK  
*** 18 ---------------------- tmp7 = C$23-C$24;
SUBI R4,BK,R0  
STI R0,**FP(1)  
*** 20 ---------------------- C$18 = dataptr[6];
LDI *AR4(6),R1  
STI R1,**FP(4)  
*** 20 ---------------------- C$17 = dataptr[1];
LDI *AR4(1),R5  
*** 20 ---------------------- tmp6 = C$17-C$18;
SUBI R1,R5,R3  
*** 22 ---------------------- C$20 = dataptr[5];
LDI *AR4(5),R1  
STI R1,**FP(2)  
*** 22 ---------------------- C$19 = dataptr[2];
LDI *AR4(2),R1  
STI R1,**FP(3)  
*** 22 ---------------------- tmp5 = C$19-C$20;
SUBI **FP(2),R1  
LDI R1,R6  
*** 24 ---------------------- C$26 = dataptr[4];
LDI *AR4(4),IR1  
*** 24 ---------------------- C$25 = dataptr[3];
LDI *AR4(3),R7  
*** 24 ---------------------- tmp4 = C$25-C$26;
SUBI IR1,R7,R2  
*** 30 ---------------------- C$22 = C$25+C$26;
ADDI IR1,R7  
*** 30 ---------------------- C$21 = C$23+C$24;
ADDI R4,BK  
*** 30 ---------------------- tmp10 = C$21+C$22;
ADDI R7,BK,R4  
*** 31 ---------------------- tmp13 = C$21-C$22;
SUBI R7,BK,R7  
*** 32 ---------------------- C$16 = C$19+C$20;
LDI **FP(3),BK  
ADDI **FP(2),BK  
*** 32 ---------------------- C$15 = C$17+C$18;
ADDI **FP(4),R5  
*** 32 ---------------------- tmp11 = C$15+C$16;
ADDI BK,R5,IR1  
*** 33 ---------------------- tmp12 = C$15-C$16;
SUBI BK,R5
**APPENDIX A. MPEG CODE EXCERPTS**

*** 35 -----------------------
ADDI IR1,R4,R1
LSH 2,R1
STI R1,*AR4
*** 36 -----------------------
SUBI IR1,R4,R1
LSH 2,R1
STI R1,*+AR4(4)
*** 38 -----------------------
ADDI R7,R5,R4
MPYI 4433,R4
*** 39 -----------------------
LDI 6270,R1
MPYI R7,R1
ADDI R4,R1
ADDI 1024,R1
ASH -11,R1
STI R1,*+AR4(2)
*** 41 -----------------------
LDI 15137,R1
MPYI R5,R1
SUBI R1,R4,R1
ADDI 1024,R1
ASH -11,R1
STI R1,*+AR4(6)
*** 49 -----------------------
ADDI R0,R2,R4
*** 50 -----------------------
ADDI R3,R6,BK
*** 51 -----------------------
ADDI R3,R2,R5
*** 52 -----------------------
ADDI R0,R6,R7
*** 53 -----------------------
ADDI R0,R6,IR1
ADDI R2,IR1
ADDI R3,IR1
MPYI 9633,IR1
*** 59 -----------------------
NEGI R4
MPYI 7373,R4
*** 60 -----------------------
NEGI BK
MPYI 20995,BK
*** 61 -----------------------
NEGI R5
MPYI 16069,R5
*** 62 -----------------------
NEGI R7
MPYI 3196,R7
*** 64 -----------------------
ADDI IR1,R5
*** 65 -----------------------
ADDI IR1,R7
*** 67 -----------------------
MPYI 2446,R2
ADDI R4,R2,R1
ADDI R5,R1
ADDI 1024,R1
ASH -11,R1
STI R1,*+AR4(7)
*** 68 -----------------------
MPYI 16819,R6
ADDI BK,R6,R1
ADDI R7,R1
ADDI 1024,R1
ASH -11,R1
STI R1,*+AR4(5)
*** 69 -----------------------
MPYI 25172,R2
ADDI R4,R2,R1
ADDI R5,R1
ADDI 1024,R1
ASH -11,R1
STI R1,*+AR4(5)

```c
*dataptr = tmp10+tmp11<<2;
```

```c
```

```c
z1 = (tmp12+tmp13)*4433;
```

```c
dataptr[2] = tmp13*6270+z1+1024>>11;
```

```c
dataptr[6] = z1-tmp12*15137+1024>>11;
```

```c
z1 = tmp4+tmp7;
```

```c
z2 = tmp5+tmp6;
```

```c
z3 = tmp4+tmp6;
```

```c
z4 = tmp5+tmp7;
```

```c
z5 = (tmp5+tmp7+tmp4+tmp6)*9633;
```

```c
z1 = -(z1*7373);
```

```c
z2 = -(z2*20995);
```

```c
z3 = -(z3*16069);
```

```c
z4 = -(z4*3196);
```

```c
dataptr[7] = (tmp4 *= 2446)+z1+z3+1024>>11;
```

```c
dataptr[5] = (tmp5 *= 16819)+z2+z4+1024>>11;
```

```c
dataptr[3] = (tmp6 *= 25172)+z2+z3+1024>>11;
```
APPENDIX A. MPEG CODE EXCERPTS

MPYI 25172,R3
ADDI BK,R3,R1
ADDI R5,R1
ADDI 1024,R1
ASH -11,R1
STI R1,**AR4(3)

MPYI 12299,R0
STI R0,**FP(1)
ADDI R4,R0,R1
ADDI R7,R1
ADDI 1024,R1
ASH -11,R1
STI R1,**AR4(1)

*** 70 -----------------------
dataptr[1] = (tmp7 *= 12299)+z1+z4+1024>>11;

*** 72 -----------------------
dataptr += 8;

LB: ADDI 8,AR4
*** 16 -----------------------
if ( --L$1 >= 0 ) goto g8;

*** 79 -----------------------
dataptr = data;

LDI AR2,AR4
*** -----------------------
L$2 = 7;

LDI 7,RC
*** -----------------------
g5:

RPTB L7
*** 82 -----------------------
C$12 = dataptr[56];

LDI **AR4(56),R4
*** 82 -----------------------
C$11 = *dataptr;

LDI *AR4,BK
*** 82 -----------------------
tmp7 = C$11-C$12;

SUBI R4,BK,R0
STI R0,**FP(1)
*** 84 -----------------------
C$6 = dataptr[48];

LDI **AR4(48),R1
STI R1,**FP(3)
*** 84 -----------------------
C$5 = dataptr[8];

LDI **AR4(8),R1
STI R1,**FP(2)
*** 84 -----------------------
tmp6 = C$5-C$6;

SUBI **FP(3),R1
LDI R1,R3
*** 86 -----------------------
C$8 = dataptr[40];

LDI **AR4(40),IR1
*** 86 -----------------------
C$7 = dataptr[16];

LDI **AR4(16),R1
STI R1,**FP(4)
*** 86 -----------------------
tmp5 = C$7-C$8;

SUBI IR1,R1,R6
*** 88 -----------------------
C$14 = dataptr[32];

LDI **AR4(32),R5
*** 88 -----------------------
C$13 = dataptr[24];

LDI **AR4(24),R7
*** 88 -----------------------
tmp4 = C$13-C$14;

SUBI R5,R7,R2
*** 94 -----------------------
C$10 = C$13+C$14;

ADDI R5,R7
*** 94 -----------------------
C$9 = C$11+C$12;

ADDI R4,BK,R5
*** 94 -----------------------
tmp10 = C$9+C$10;

ADDI R7,R5,R4
*** 96 -----------------------
tmp13 = C$9-C$10;

SUBI R7,R5,R7
*** 96 -----------------------
C$4 = C$7+C$8;

ADDI IR1,R1,R5
*** 96 -----------------------
C$3 = C$5+C$6;

LDI **FP(2),BK
ADDI **FP(3),BK
APPENDIX A. MPEG CODE EXCERPTS

*** 96 -----------------------
ADDI R5, BK, IR1
*** 97 -----------------------
SUBI R5, BK, R5
*** 99 -----------------------  * dataptr = tmp10+tmp11+16>>5;
ADDI IR1, R4, R1
ADDI 16, R1
ASH -5, R1
STI R1, *AR4
*** 100 -----------------------
SUBI IR1, R4, R1
ADDI 16, R1
ASH -5, R1
STI R1, *AR4(32)
*** 102 -----------------------
z1 = (tmp12+tmp13)*4433;
ADDI R7, R5, R4
MPYI 4433, R4
*** 103 -----------------------
* dataptr[16] = tmp13*6270+z1+131072>>18;
LDI 6270, R1
MPYI R7, R1
ADDI R4, R1
ADDI QCONST+0, R1
ASH -18, R1
STI R1, *AR4(16)
*** 105 -----------------------
* dataptr[48] = z1-tmp12*15137+131072>>18;
LDI 15137, R1
MPYI R5, R1
SUBI R1, R4, R1
ADDI QCONST+0, R1
ASH -18, R1
STI R1, *AR4(48)
*** 113 -----------------------
z1 = tmp4+tmp7;
ADDI R0, R2, R4
*** 114 -----------------------
z2 = tmp5+tmp6;
ADDI R3, R6, BK
*** 115 -----------------------
z3 = tmp4+tmp6;
ADDI R3, R2, R5
*** 116 -----------------------
z4 = tmp5+tmp7;
ADDI R0, R6, R7
*** 117 -----------------------
z5 = (tmp5+tmp7+tmp4+tmp6)*9633;
ADDI R0, R6, IR1
ADDI R2, IR1
ADDI R3, IR1
MPYI 9633, IR1
*** 123 -----------------------
z1 = -(z1*7373);
NEGI R4
MPYI 7373, R4
*** 124 -----------------------
z2 = -(z2*20995);
NEGI BK
MPYI 20995, BK
*** 125 -----------------------
z3 = -(z3*16069);
NEGI R5
MPYI 16069, R5
*** 126 -----------------------
z4 = -(z4*3196);
NEGI R7
MPYI 3196, R7
*** 128 -----------------------
z3 += z5;
ADDI IR1, R5
*** 129 -----------------------
z4 += z5;
ADDI IR1, R7
*** 131 -----------------------
* dataptr[56] = (tmp4 *= 2446)+z1+z3+131072>>18;
MPYI 2446, R2
ADDI R4, R2, R1
ADDI R5, R1
ADDI QCONST+0, R1
ASH -18, R1
STI R1, *AR4(56)
*** 133 -----------------------
* dataptr[40] = (tmp5 *= 16819)+z2+z4+131072>>18;
MPYI 16819, R6
APPENDIX A. MPEG CODE EXCERPTS

ADDI BK,R6,R1
ADDI R7,R1
ADDI @CONST+0,R1
ASH -18,R1
STI R1,**AR4(40)

*** 135 -----------------------

MPYI 25172,R3
ADDI BK,R3,R1
ADDI R5,RI
ADDI @CONST+0,R1
ASH -18,R1
STI R1,**AR4(24)

*** 137 -----------------------

dataptr[24] = (tmp6 * 25172) + z2 + z3 + 131072 >> 18;

MPYI 12299,R0
STI R0,**FP(1)
ADDI R4,R0,R1
ADDI R7,R1
ADDI @CONST+0,R1
ASH -18,R1
STI R1,**AR4(8)

*** 140 -----------------------

MPYI 25172,R3
ADDI BK,R3,R1
ADDI R5,RI
ADDI @CONST+0,R1
ASH -18,R1
STI R1,**AR4(24)

*** 137 -----------------------

dataptr[8] = (tmp7 * 12299) + z1 + z4 + 131072 >> 18;

MPYI 12299,R0
STI R0,**FP(1)
ADDI R4,R0,R1
ADDI R7,R1
ADDI @CONST+0,R1
ASH -18,R1
STI R1,**AR4(8)

*** 140 -----------------------

++dataptr;

L7: ADDI 1,AR4

*** 80 -----------------------

if ( --L$2 >= 0 ) goto g7;

*** -----------------------

return;

EPIO1:
LDI **FP(1),R1
LDI *FP,FP
POPP AR4
POPP AR7
POPP R6
BD R1
POPP R5
POPP R4
SUBI 6,SP

*** B R1; BRANCH OCCURS

**********************************************************************

* DEFINE CONSTANTS
**********************************************************************

.bss CONST,1
.sect "cinit"
.word 1,CONST
.word 131072 ; 0
.end
Appendix B

MPEG Examples

To provide a better feel for what typical compression results could be achieved, a study was done using a public domain software encoder. Table B shows results of compressing various image sequences.

Table B.1: Some results obtained from Stanford’s software MPEG encoder on various sources.

<table>
<thead>
<tr>
<th>Video sequence</th>
<th>Resolution</th>
<th>Number of frames</th>
<th>MPEG size (in kilobytes)</th>
<th>Compression factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: bicycle</td>
<td>352x240</td>
<td>148</td>
<td>719</td>
<td>26</td>
</tr>
<tr>
<td>2: canyon</td>
<td>144x112</td>
<td>1758</td>
<td>1744</td>
<td>24</td>
</tr>
<tr>
<td>3: coast</td>
<td>240x176</td>
<td>43</td>
<td>60.2</td>
<td>45</td>
</tr>
<tr>
<td>4: mjackson</td>
<td>144x112</td>
<td>557</td>
<td>725</td>
<td>18.6</td>
</tr>
<tr>
<td>5: mobile</td>
<td>352x240</td>
<td>148</td>
<td>719</td>
<td>26</td>
</tr>
</tbody>
</table>

Figures B-1 and B-2 is a sampling from the sequences used to study the MPEG standard.
Figure B-1: A frame from test sequence table-tennis. Original in color.

Figure B-2: A frame from test sequence mobile. Original in color.
Bibliography


