Parallel MPEG Encoder

Quinn Mahoney
⟨qmahoney@mit.edu⟩

May 16, 2005

Abstract

Here I present an implementation of a parallel MPEG-2 encoder. This encoder is unique in that it employs the principles of graph partitioning to distribute the workload among processors and to improve upon the encoded video quality. I show that the pre-processor provides significant quality and performance gains with a relatively small amount of overhead.

Contents

List of Figures

1 Introduction

The MPEG-2 standard is incredibly popular for encoding consumer video and as such, it is advantageous to examine the encoding algorithm in a parallel context. The goals of this project are first to produce a parallelized MPEG encoder with a performance increase proportional to the number of processors. In order to achieve this, the standard algorithm is modified to avoid bottlenecks in disk access and inter-process communication.

The second goal of this project is to use graph partitioning to improve both the load balancing among processors and to create high quality video files. I will show that a video sequence that is properly partitioned for encoding will produce a better output stream with a lower file size. For this purpose, I use the ParMETIS partitioning library. The MPEG encoder is based on the commonly-available MSSG implementation.

This particular implementation differs from other parallel encoders primarily in the use of a graph partitioner as a pre-processor. The Berkeley Multimedia Research Center has released a parallel MPEG encoder which is similar in that it parallelizes the video stream in the temporal direction.[1] However, the BMRC encoder requires three auxiliary processes: the master server, a decode server, and a combine server. My implementation eliminates the need for these separate servers and uses a pre-processor to minimize communication between servers.

The Cornell parallel MPEG encoder is written in their own Resolution Independent Video Language and is designed to run on heterogeneous and unreliable hardware.[2] Thus, it has totally different architectural and performance requirements. It is similar, though, in that it uses a "scene detector" to optimize performance prior to encoding.

2 The MPEG-2 Standard

MPEG-2 is a common video encoding standard employed most notably in the Digital Versatile Disc (DVD), but it is also used for Internet video in both streaming and non-streaming applications. MPEG compresses raw video into a much smaller format by exploiting similarities between frames (*temporal redundancy*), similarities within a frame (*spatial redundancy*), and removing detail that is undetectable to the human eye.

The quality of an MPEG video and the size of the file are determined by the particular MPEG encoder and its run-time options. A good MPEG encoder may create very high-quality files, but it may also take more time to encode the file. Most encoders are adjustable such that the user can compress the video to a target bit rate.

2.1 Discrete Cosine Transform

The discrete cosine transform is performed on a single frame in order to exploit any spatial redundancy that exists in the picture. For instance, in Figure 1, we see that each picture has a huge area of snow around the skier. The storage required for this frame decreases dramatically if we account for this feature.

[1] Gong, "The Parallel Berkeley Encoder," 1995

[2] Dawson, "Optimal Parallel MPEG Encoding," 1995

The particular DCT algorithm can vary between encoders. For my purposes, I use the code from the MSSG serial implementation, which is based on the fast Chen-Wang 1984 algorithm.[3] This algorithm scans the frame in a "zigzag" pattern and produces the DCT coefficients for each macroblock.

The DCT coefficients are 'quantized,' meaning that their lower order bits are dropped, under the assumption that this information would be invisible to a person. The resulting coefficients are Huffman coded using fixed tables. Interestingly, this process is very similar to the common JPEG compression for still pictures.

2.2 Motion Vector Prediction

A scene within a movie consists of multiple shots, each taken from a viewpoint that is either stationary or slowly moving. The result is that within each shot, the picture changes only slightly between frames. An MPEG encoder takes advantage of this fact by predicting the content of a frame based on those that come before and after. This process, called *motion vector prediction*, is the second of the two compression techniques employed for MPEG video.

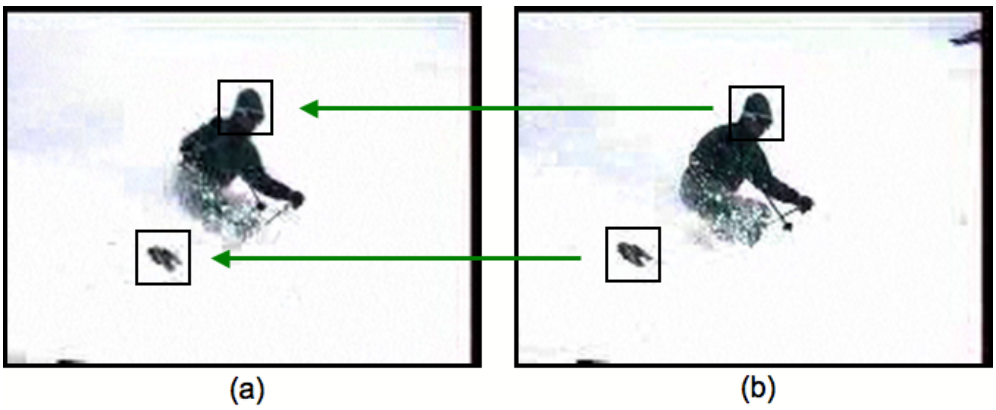

Figure 1: Temporal Redundancy

Examine Figure 1, which shows two adjacent frames of a video sequence. As you can see, the skier's head remains at the top center of the screen, but the tip of the skis have moved slightly to the left. The encoder searches for this type of similarity and uses this to form a prediction about frame (b) based on frame (a).

MPEG is often described as a "block-based coding scheme".[4] Each frame is divided into *macroblocks*, which are 16×16 blocks of pixels (See Figure 2). According to the standard, the frame dimensions do not have to be a multiple of 16, but in practice they usually are.

[3]MSSG, "MPEG-2 Video Encoder," 1994

[4]Fogg, "Questions that Should Be Frequently Asked About MPEG," 1996

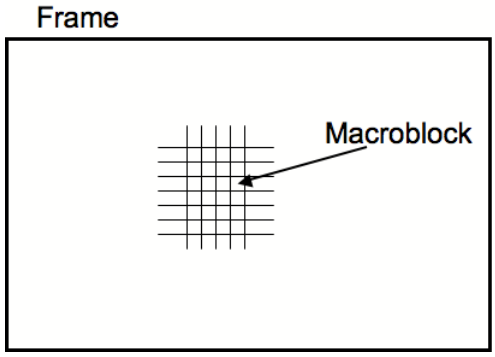

Figure 2: Macroblocks

The motion vector prediction can be performed for a given frame (called F in Figure 3), relative to a reference frame (called R). For every macroblock in F, the encoding algorithm searches for a similar block of pixels in R. When a good match is found, the matching pixels are subtracted from the macroblock in F. The encoded macroblock then consists of the displacement vector and the difference between the two blocks. This new format requires much less space than the original raw pixels.

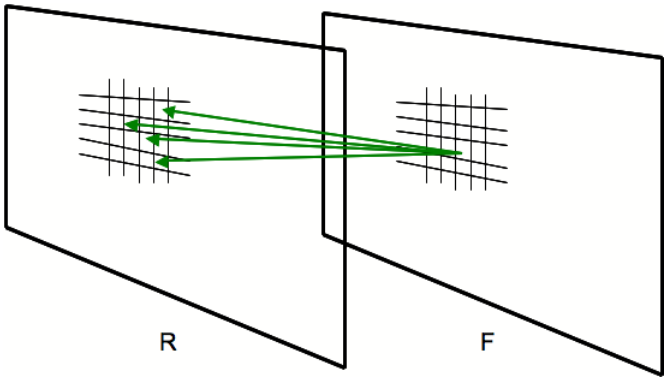

Figure 3: Candidate Motion Vectors

2.3 Sequence Structure

Of course, motion vector prediction only works for frames that are similar to each other. So it only makes sense to use motion vectors for frames that are nearby. The MPEG standard specifies three types of frames, as demonstrated in Figure 4. The first type is the I type, for "Intra-coded". The I frame does not reference any other frames, but it is compressed using the DCT described in section 2.1.

The P type frame (for "Predicted") contains references to the most recent I frame only. Within a predicted frame, each macroblock might be coded with a motion vector, or it might be coded "Intra," without a reference. Consequently, a P type frame may also be coded Intra such that it does not reference the I frame at all.

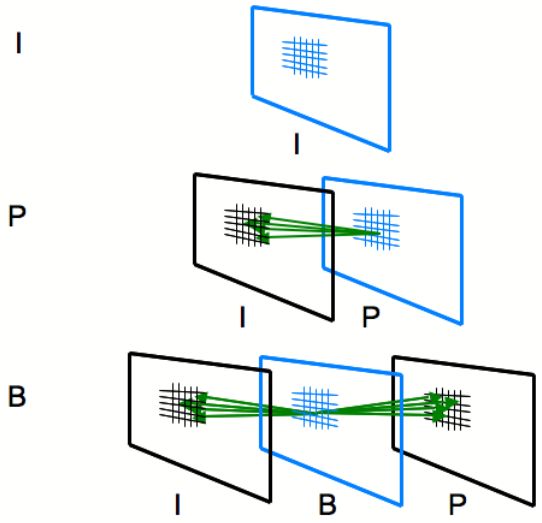

Figure 4: Frame Types

Finally, the *B* type frame (for "Bi-directional," though it is also called "Interpolated") can contain references to the most recent I frame in the past and the next P frame in the future. For instance, the prediction for a single block might be determined as a weighted sum of both a forward and a backward reference. It possible for a frame to be based on a frame in the future because the encoder always stores the P frame *before* the B frames. So, a sequence of frames such as: $I_0\ B_1\ B_2\ B_3\ P_4$ would actually be stored as: $I_0\ P_4\ B_1\ B_2\ B_3$. A *group of pictures* is defined as an I frame and all of the following frames until the next I frame.

The specific sequence of I, P, and B frames is completely up to the encoder, but it has a significant impact on the quality and file size of the encoded video. Techniques for determining such a sequence will be discussed further in section 3.2.

2.4 The Encoding Loop

Figure 5 is a flow-chart of the main loop of the MPEG encoder. The dotted arrows represent a disk read or write. By far, the most time-intensive part of the computation is the motion estimation. The reason is that for every macroblock in a predicted frame, the encoder searches a larger area of the reference frame for a close match.

Typically, the search dimensions are between ± 15 and ± 32 pixels.[5] This means that it must search between 961 and 4,225 different positions for a good match. Of course, the real algorithms are a bit smarter than that, but this at least demonstrates that the running time for the motion estimation increases quadratically with the size of the search window.

The second most time-intensive computation is the DCT. The running time of this step is proportional to the area of the frame, so it is also a quadratic function of the dimensions. Of course, the motion vector search skips all I frames, but I frames make up a small fraction of the frame sequence.

[5]Fogg, "Questions that Should Be Frequently Asked About MPEG," 1996

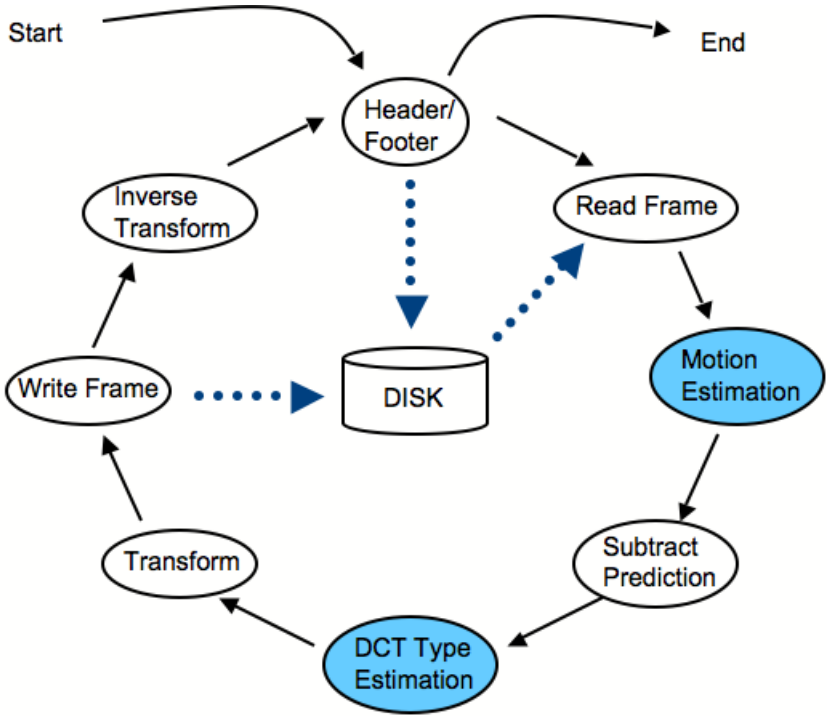

Figure 5: The Encoding Loop

The complete transform is computed prior to writing the encoded frame to disk. Finally, for I and P frames, the encoded version is *decoded* so that they can serve as reference for other frames. It is important that motion predictions refer to the decoded frame rather than the original frame because in the end, the decoder does not have access to the original version.

In section 3.1 I will discuss how we can achieve better performance through parallel processing.

3 Implementation

The parallel MPEG encoder implementation is based on the serial implementation written by the MSSG.[6] Specifically, the motion vector estimation and the DCT remain unchanged. But the read and write methods are updated to minimize conflicts between processors. However, the most significant change is the addition of a pre-processor (described in sections 3.2 and 3.3) which divides the workload evenly among the processors and also produces higher-quality MPEG video than the MSSG version. The encoder is implemented in C/MPI.

[6]MSSG, "MPEG-2 Video Encoder," 1994

3.1 Parallel Computation

As noted above, the most computationally intensive parts of the encoding process are the motion estimation and the discrete cosine transform. The DCT of a frame is independent of any other frames and the motion estimation is dependent only on the closest I and P frames. Therefore the encoding process can be distributed among processors by splitting the movie at the I frames.

The MSSG encoder reads the video stream on a frame-by-frame basis. But in a parallel context, this scheme makes little sense. This would cause processes to block frequently on disk access, and so would undermine the benefits of parallel computing. The solution to this is to read a group of pictures at a time. All motion vector references would be contained within the group, so no further disk access would be required until that group is fully encoded.

3.2 Partitioning

Recalling the definitions of I, P, and B, frames in section 2.3, it is obvious that the different types require different amounts of storage and produce various quality video. For instance, if a movie were made completely of I frames, then each frame would be as detailed as a JPEG picture. However, such a movie would require much more space to store.

Most encoders use a fixed pattern of frame types in order to encode the video. For example, one common pattern would have an I frame once every 15 frames (1/2 a second) and would insert a P frame once every three, like this:

$$I_0\ B_1\ B_2\ P_3\ B_4\ B_5\ P_6\ B_7\ B_8\ P_9\ B_{10}\ B_{11}\ P_{12}\ B_{13}\ B_{14}\ I_{15}\ B_{16}\ B_{17}\ P_{18}...$$

Yet this method is far from optimal. The natural place for an I frame is at a cut or a scene transition, where ever the video is changing the most. A fixed sequence would not account for this transition, and so could harm the quality of the video or increase the file size. For quality, it is desirable to have variable sequences of frames, such as:

$$I_0\ B_1\ P_2\ B_3\ B_4\ B_5\ B_6\ B_7\ I_8\ P_9\ P_{10}\ B_{11}\ B_{12}\ B_{13}\ I_{14}\ B_{15}\ B_{16}\ B_{17}\ P_{18}...$$

This is where graph partitioning becomes useful. Assume that we could perform some quantitative analysis of the relation between a frame and every other frame in the sequence. These relations could be represented as a weighted graph, where each node is a frame and each edge weight denotes the similarity between the frames.

In order to perform this graph partitioning, the encoder requires a pre-processor. The pre-processor constructs this weighted graph with a single pass of the entire video stream and produces a partitioning of the stream that can be used to determine the proper frame sequence.

3.3 The Pre-processor

The pre-processor is designed to create a "good" partitioning of the video stream from a single pass of the data. A good partitioning would split the video stream at scene changes

and cuts and avoid splitting it between very similar frames. However, since this would involve extra overhead, it is preferable to make the pre-processor run in parallel as well.

The first step is to generate a weighted graph, in the form of a sparse matrix, where each node represents a frame in the sequence and each edge represents a similarity between two frames. The actual edge weights are calculated by subtracting the pixels of one frame from another. A more accurate method would perform all the motion vector searches, but this would be incredibly slow and redundant.

Most frames will be similar to those around it, and no more. The result is a fairly linear graph, such that the matrix representation would only have elements close to the diagonal. This step can be easily distributed among multiple processors. The weighted graph is stored in the distributed compressed storage format (CSR) described in the ParMETIS manual.[7]

Once the weighted graph is constructed, the pre-processor uses the ParMETIS graph partitioning library. ParMETIS is designed to compute high quality partitionings for very large graphs, especially when the input graph is already well-distributed among processors. In this case, the frames would first be distributed evenly among the processors, so we can expect the ParMETIS algorithm to run quickly.

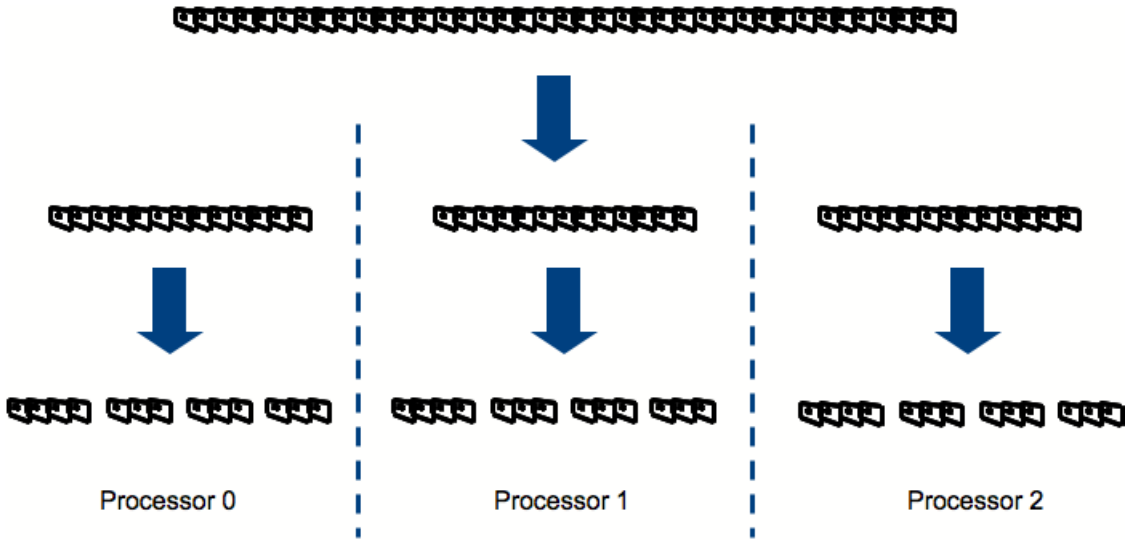

Figure 6: A p-way partitioning followed by a k-way partitioning

As of ParMETIS version 2.0, the graph partitioning algorithm can produce a k-way partitioning on p processors (as opposed to just a p-way partition). The pre-processor will run the partitioner twice (see Figure 6). First, it will create a p-way partition to distribute the data among the processors. Second, it will create a k-way partition to divide the video into groups for encoding. For the purposes of this implementation, we will choose k such that the average group will consist of about 0.5s of video, which is common for commercial encoders.

In this manner, we will achieve a high quality partitioning for better load balancing and high quality encoding.

[7]Karypis et al., "ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library," 2003

4 Results

Unfortunately, I was not able to finish the ParMETIS version of the MPEG-2 encoder. The preprocessor completes the weighted graph and creates the compressed storage format, but the ParMETIS function crashes consistently.

However, I did create a parallel version *without* graph partitioning, so that I could use it as a basis for comparison. The non-partitioned encoder distributes the groups in parallel, but uses a fixed sequence for determining I, P, and B frames.

The parallel encoder was tested against the normal encoder by ensuring that when the output is decoded, it exactly matches the input files.

Two input files were used: a short, 100-frame sequence, and a long, 1516-frame sequence. The long sequence performed close to ideal, whereas the encoding time for the short video was dominated by overhead. (See Figures 7 and 8.)

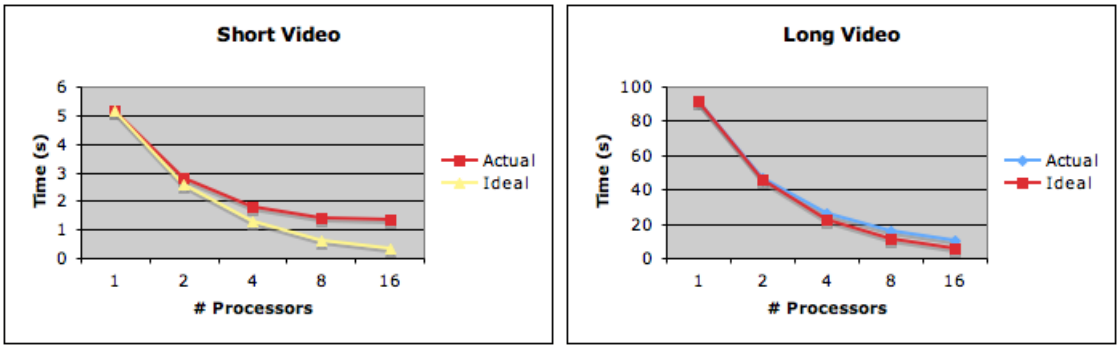

Figure 7: Encoding Time

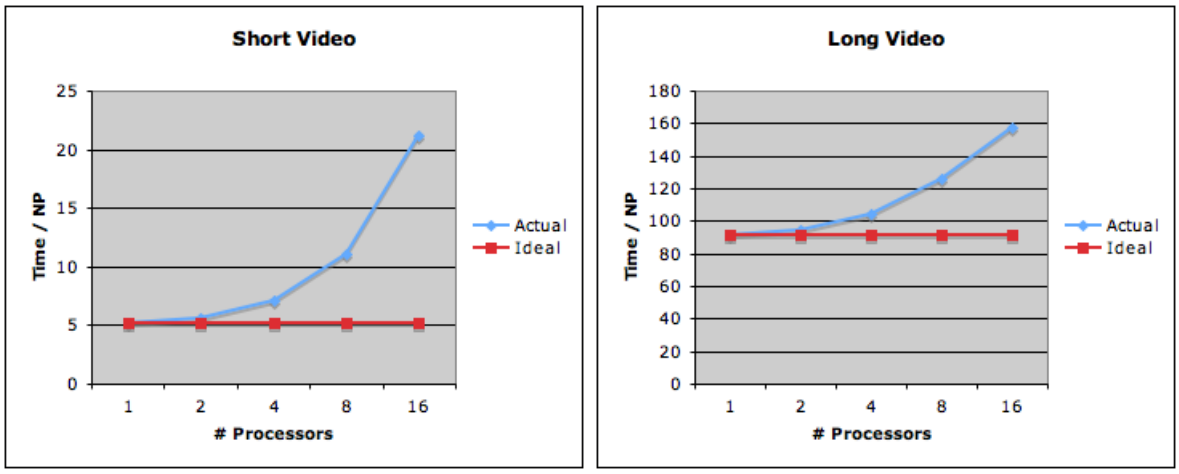

Figure 8: Encoding Time per Processor

5 Future Work

Though the pre-processor has its advantages, it also requires the entire raw video to be available before encoding. This precludes it from being used in any streaming applications. (Note that the encoded file can still be sent over the Internet and viewed in a streaming fashion.) One such application would be a television recorder. In that case, it is preferable to compress the video as it is captured and saved to disk.

This implementation could be modified to support streaming encoding, though with possible loss of quality. The encoder would have a FIFO buffer of raw footage to which it would constantly add frames and from which it would constantly remove frames. Each time data is added or subtracted, the weighted graph could be repartitioned quickly, as this is a feature of the ParMETIS library. In this way one might create a streaming, or single-pass encoder with the benefits of a pre-processor.

Additionally, the encoder would benefit from an enhanced file server. Currently, frames are read and written over NFS to the frontend server. The file access is performed only at certain intervals in order to avoid bottlenecks, but it could be more efficient with a dedicated file server.

6 References

Dawson, Lee, and Moore. 1995. *Optimal Parallel MPEG Encoding.* Cornell University; available from http://www.cs.cornell.edu/Info/Projects/zeno/Projects/OPME/opme.report.html.

Fogg, Chad. 1996. *MPEG-2 FAQ: Questions that Should Be Frequently Asked About MPEG.* Available from http://bmrc.berkeley.edu/research/mpeg/faq/mpeg2-v38/index.html.

Gong, Kevin et al. 1995. *The Parallel Berkeley Encoder.* University of California, Berkeley; available from http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_encode.html.

Karypis et al. 2003. *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library.* University of Minnesota; available from http://www-users.cs.umn.edu/ karypis/metis/parmetis/index.html.

MSSG. 1993. *MPEG-2 Video Encoder.* Available from http://www.mpeg.org/MPEG/MSSG/.