A Real-Time MPEG Software Decoder Using a Portable Message-Passing Library

Man Kam Kwong, P. T. Peter Tang, and Biquan Lin*

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439-4844
Email: kwong, tang, blin@mcs.anl.gov

Abstract

We present a real-time MPEG (Motion Pictures Expert Group) software decoder that uses message-passing libraries such as MPL, `p4`, and MPI. The parallel MPEG decoder currently runs on the IBM SP system but can be easily ported to other parallel machines. This paper discusses our parallel MPEG decoding algorithm as well as the parallel programming environment under which it uses. Several technical issues are discussed, including balancing of decoding speed, memory limitation, I/O capacities, and optimization of MPEG decoding components. This project shows that a real-time portable software MPEG decoder is feasible in a general-purpose parallel machine.

Keywords: Image processing, high-performance computing, video compression, real-time system, message-passing library.

1 Introduction

Video compression is a crucial technique in coping with large amounts of digitized video data. MPEG is an industrial standard of video and associated audio compression for digital media storage and transmission. An MPEG video system consists of an encoder and a decoder: the encoder compresses a sequence of images (video) into a bitstream and the decoder decompresses the bitstream and displays the decompressed video. Since a video sequence has to be displayed in real-time, an MPEG decoder is required to perform over a billion operations per second. Usually, special hardware with signal processing chips is needed to implement an MPEG decoder. This paper explores the possibility of using portable parallel software environment to implement such a video decoder.

Although a hardware-based MPEG system can encode and decode video sequences in real-time and the cost for the hardware will decrease dramatically in the coming years, a software-based approach presents several advantages: First, it provides a simulation environment for

*This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

designing the hardware. In fact, a software simulation must be performed before designing any hardware-based MPEG system, since it involves complex compression algorithms. Second, a software-based approach provides flexibility to accommodate growing varieties of algorithms and specific applications. Third, a software-based approach enables the use of a single general-purpose multiprocessor computer which, for many visual communication and image processing tasks, is more economical than buying separate special hardware pieces. Our investigation of a parallel software-based implementation of MPEG system was motivated by these consideration.

Recently, several real-time software decoders have been implemented. Rowe et al. [7] developed a portable MPEG-1 video decoder that can play small-sized (160×120) video in real-time. They used a SPARC 1+ to read the bitstream and a SPARC 10 to decode and display the video. Some frames may be dropped to accommodate network load and decoding speed. Taylor [8] implemented an MPEG-1 encoder and decoder that works in real-time using some special DSP processors embedded in parallel hardware. The drawback of this implementation is that it cannot be ported to a general-purpose parallel machine without such DSP processors. Ghafoor et al. [1] studied speedup with different numbers of processors on several parallel machines including the nCUBE2 and Intel's Paragon. But they did not incorporate such parallel decoding processes with real-time and continuous video display.

Our parallel MPEG-1 parallel decoder has the following features. First it is implemented in a general-purpose parallel machine (IBM SP) and can be easily ported to other machines, since it uses a message passing library such as MPL, `p4` and MPI. Second, it can decode and display video smoothly in real-time by means of a HIPPI (HIgh Performance Parallel Interface) frame buffer. Third, the parallel MPEG decoder requires only 16 processors, which are now available on many commercial parallel machines.

The remainder of this paper is organized as follows. Section 2 discusses our parallel MPEG-1 decoding algorithm. Section 3 describes our implementation environment, including the system configuration and message-passing libraries used. Section 4 discusses several technical issues faced in implementing the decoder. Section 5 presents our testing results. Finally, Section 6 summarizes the project and points out some future research and implementation topics.

2 Parallelization of the MPEG Decoder

MPEG is a video coding standard established by the Motion Pictures Expert Group of the International Standards Organization. Version 1 of MPEG (or MPEG-1) is primarily designed for digital storage such as CD-ROM at transmission speeds up to 1.5 Mbits/second. MPEG-2 is designed as a generic standard to support a variety of applications including high-definition TV, digital cable TV, and video-on-demand. Both MPEG-1 and MPEG-2 use discrete cosine transform coding, motion estimation, and Hoffman coding techniques to compress video data. This paper is mainly concerned with MPEG-1.

The syntax of an MPEG bitstream is organized into several layers: video sequence layer, group of pictures (GOP) layer, picture layer, slice layer, macroblock layer, and block layer. An upper layer encapsulates a lower layer, and each layer conveys information for some specific functions. For example, the video sequence layer contains information for an entire video sequence such as video size, bit rate, and default quantization matrices; the picture layer contains

information such as picture coding type and temporal reference for non-intra coded pictures; the macroblock layer deals with motion estimation and compensation; and the block layer contains information on DCT coefficients.

There are three types of MPEG picture frames: intra-coded (I) frame, predictive-coded (P) frame, and bidirectionally predictive-coded (B) frame. An I-frame is coded by using information only from itself. A P-frame is coded by using motion compensation from a past I-frame or P-frame. A B-frame is coded by using motion compensation from a past and/or future I-frame or P-frame. The group of pictures (GOP) layer is intended to assist random access to the sequence. A GOP contains at least one I-frame, and it may contains some P-frames and B-frames. In the bitstream, the first frame in a GOP must be an I-frame, and the reference frames (an I-frame or a P-frame) by a P-frame or a B-frame are coded ahead so the the bitstream can be decoded and displayed on-the-fly. But in display order, the first displayed frame in a GOP needs not be an I-frame; it may use an I-frame or a P-frame in the preceding GOP. In general, a GOP is a relatively independent unit and can be decoded in parallel if we add the sequence header and the previous GOP information. Our parallel algorithm is based on this observation.

Figure 1 is the diagram of the parallel MPEG decoder. The parallel MPEG decoder consists of a distributor, a number of decoders, and a collector. The distributor cuts a sequential MPEG bitstream into segments. Each segment contains sequence header, the preceding GOP (which may be referred to by the current GOP), the current GOP, and the sequence end code. The distributor also dispatches the cut segments to decoders in turn. Each decoder receives and decodes segments, dithers the decoded frames into the ARGB format (the display format for HIPPI), and sends frames to the collector. The number of decoders is scalable to accommodate different CPU speeds. In our system, 14 to 18 SP nodes (each roughly equivalent to a RS/6000 model 370 workstation) are sufficient to achieve real-time decoding (30 frames/second). The collector collects decoded frames in order and sends them to a HIPPI frame buffer for real-time display.

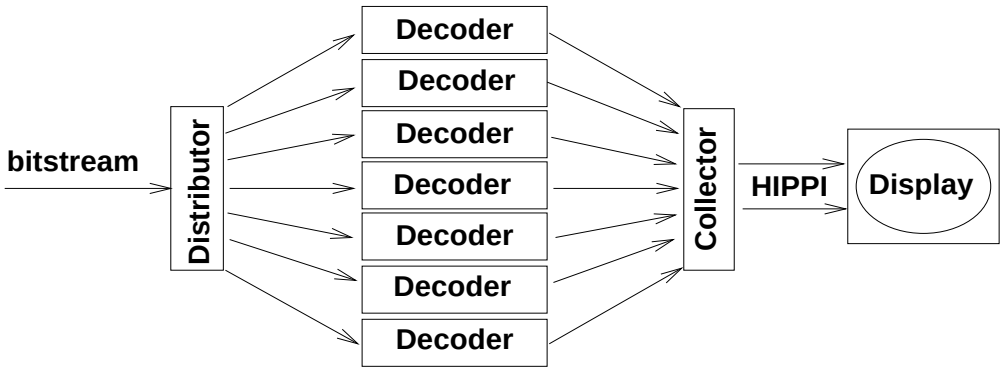

Figure 1. The Basic Model of Parallel MPEG Decoder

3 System Environment and Parallel Programming Libraries

The parallel MPEG decoder was developed on IBM SP system using message passing parallel libraries. In this section, We describe system environment and parallel software tools.

SP. The SP is an IBM POWERparallel system that can provide high-performance CPU and I/O power with scalability and flexibility on a UNIX operating system. The current SP2 system can be scaled from 2 to 512 nodes, each node is essentially an RS/6000 model 370. The nodes are connected by internal high-performance switch. In the Mathematics and Computer Science Division of Argonne National Laboratory, 128 nodes are currently installed; each node is equipped with 128 MBytes of memory and 125 MFlops. The peak performance for switching between nodes is 35 MBytes/sec bandwidth and 63 μsec latency. In our parallel MPEG decoding system, only 16 to 20 nodes are required to achieve real-time performance.

MPL. MPL is IBM's message-passing library for the high-performance switch. It is easy to parallelize a standard C program by calling a few message-passing functions in the MPL library. In our implementation of the MPEG decoder, fewer than 10 MPL functions are used. A list of MPL message-passing functions can be found in [3].

p4. `p4` is one of the most popular message-passing systems that can run on a wide variety of parallel systems and workstations. One of the impediments to widespread use of parallel computers is lack of standard software tools; users have to use specific software tools provided by vendors. `p4` is an early effort to build a "common language" for these machines. Currently, it has been installed in most major parallel machines and workstations We implemented the parallel MPEG decoder using *p4* library; and the performance is almost the same as that using MPL library.

MPI. MPI (Message Passing Interface) is a standard for message-passing system established by a broadly based parallel computing group including vendors, library developers, and users. MPI was completed in the spring of 1994 and is now awaiting public comments. An excellent book on MPI for newcomers as well as for experienced parallel researchers and programmers is [2]. One version of our parallel MPEG decoder was implemented with the MPI message-passing system.

HIPPI. HIPPI (HIgh Performance Parallel Interface) is, as its name says, a high-performance I/O interface. At Argonne, a HIPPI frame buffer developed by Input Output Systems Corporation is connected by a HIPPI channel to the IBM SP2 system. The image can be displayed from the HIPPI frame buffer at high resolution (1280×1024) or low resolution (640×512). TCP/IP and IPI-3 protocols are currently used for the connection. The peak transmission performance is 40 MBytes/sec. Our parallel MPEG system delivers 30 frames/sec. at low resolution.

4 Implementation Issues for the Parallel MPEG Decoder

In this section, we discuss several technical issues in our implementation of parallel MPEG decoder. These issues must be taken into account when porting the parallel MPEG decoder into other machines.

Parallel Models. Figure 1 is a simple parallel MPEG decoding model. We also studied several more complicated parallel models to accommodate different CPU speeds, memory capacities, and transport protocols. Here we give some examples:

Token Model. Asynchronic message passing between nodes makes tasks more independent of each other. For example, in `p4`, the *p4_send*() function will return without waiting until an acknowledgment is received, so that the calling process can continue work on other calculations such as decoding. If this function is used, some decoders may keep sending decoded frames to the collector where they must be wait in the buffer. This procedure will cause overflow if the buffer size is small. A scheduling algorithm is needed to overcome this drawback. A simple scheduling policy is to pass a token among each decoding node and to allow only the node holding the token to send the frames. Once it finishes sending, it releases the token to the next decoding process. This model is called a token model.

Scalable Model. Another way to overcome the memory limitation of the collector is to build a hierarchical buffering for the collector. For example, we can add a first-layer buffering processor for every three decoders and a second-layer buffering processor for every first-layer buffering processors and so on. This model enables decoding processes to be scaled to any number. The disadvantage of this model is that it introduces many overhead.

Parallel I/O Model. Display speed and stability can be dramatically improved if we can let the collector's output (sending to the HIPPI frame buffer) in parallel with its input (receiving from decoding nodes). At the current stage, the time for displaying one frame is bounded by the sum of the time for receiving it from a decoding nodes and the time for sending it to the frame buffer. Moreover, an instable transmission rate between a decoding node to the collecting node will affect the display rate. This effect will be removed if a parallel I/O mechanism is implemented. A synchronization scheme is currently used to reduce the instability of transmitting frames from decoding nodes to the collecting nodes.

Load Balance. Load balance is an important issue in parallel computing. Several strategies are used in the parallel MPEG decoder. Since the decoding speeds for I-frames, P-frames and B-frames are different and a future reference frame will be delayed to display in MPEG codings, the decoding rate will vary significantly if we sent a frame as soon as it is decoded. Instead, we send frames when all frames in this GOP are decoded. Therefore, the decoding loads among decoders are almost balanced assuming each GOP requires the same decoding time. We also must balance the CPU speed and transmission capacities to achieve real-time performance. For example, if a routine that transforms a YUV format to ARGB format is put in the decoder, the transmitted data from decoding nodes to the collecting nodes will be reduced by 2.67 times. But by doing so, the collector must transform the format. This process is feasible only if the collector has a very high CPU speed.

Reducing Overhead. In our prototype implementation, one GOP with its preceding GOP is sent to each decoder. This process causes one GOP overhead for each transmission from distributor to decoder. The overhead can be reduced by transmitting several consecutive GOPs with one preceding GOP. But this modification will increase latency. The overhead can also be reduced by restricting bitstream in encoding process. If every GOP is started with an I-frame in the display order, one no longer needs to add a preceding GOP when distributing segments to decoders.

Local Optimization. Numerous coding optimizations were used in implementing our parallel MPEG decoder. These optimizations included use of local copies of variables to avoid memory references; as many register variables as possible; bit operations instead of arithmetic operations, and in-line expansions instead of function calls. Also, a fast dithering algorithm from YUV format to HIPPI's ARGB format is used.

5 Experiment Results

We tested our parallel MPEG decoder for two standard video sequences: "flower garden" (Figure 2) and "tennis" (Figure 3). The testing result are summarized in Table 1. Note that the time is an approximation based on a segment containing GOPs with six frames. The testing was conducted in the system environment described in Section 3.

Figure 2. Flower Garden Image

Figure 3. Tennis Image

Table 1. Key Statistics of Parallel MPEG Decoder

Total Number of Processors	16
Overall Speed	30 frames /sec.
Latency	about 10 sec.
Image Size	352x240
Number of GOPs	26
Number of Frames	150
Bit-rate from Disk to Distributor	3.16 MB/sec.
Bit-rate from Distributor to Decoder	17 MB/sec.
Time from Decoder to Collector	0.0112 sec./frame
Time from Collector to HIPPI	0.0167 sec./frame
Time for Dithering a Frame	0.135 sec.
Time for Decoding a Segment (Fig. 1)	2.48 sec.
Time for Decoding a Segment (Fig. 2)	1.95 sec.

6 Conclusions

In this paper, we developed a real-time software MPEG decoder using portable parallel processing tools. Compared with a hardware-based approach, the software-based approach provides a better environment for exploring video compression algorithms. In addition, the software approach enables flexibility and portability in applications. A future research topic is to investigate parallel video data distribution and management algorithms and parallel MPEG encoding schemes by using portable message passing libraries.

7 Acknowledgments

We thank our colleagues E. Lusk and W. Gropp for many discussions on using the `p4` and MPI message-passing systems at their early stages, T. Pierce for his help for efficiently using the SP2 I/O subsystem, and S. Bradshaw for allowing us to use and modify his HIPPI display program.

References

[1] Arif Ghafoor, J. Yang, and S. Baqai, "Coarse-grained Parallel Algorithm and Implementation for MPEG-1 Decoder," *Proceedings of the Workshop on Wavelets and Large-Scale Image Processing*, Argonne National Laboratory, 1994.

[2] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1994.

[3] IBM, *High-Performance Parallel Interface User's Guide and Programmer's Reference Manual*, AIX version 3.2, May 1993.

[4] IBM, *IBM AIX Parallel Environment Parallel Programming Subroutine Reference Release 2.0*, June 1994.

[5] ISO/IEC Committee Draft 11172-2, *Coding of Moving Pictures and Associated Audio for Digital Storage Media at upto 1.5 Mbits/s*, ISO/IEC JTC1/SC29 WG11, Nov. 1991.

[6] R. Butler and E. Lusk, *User's Guide to the p4 Parallel Programming System*, Technical Report ANL-92/17, Argonne National Laboratory, Oct. 1992.

[7] L. A. Rowe, K. D. Patel, B. C. Smith and K. Liu, "MPEG Video in Software: Representation, Transmission, and Playback," *SPIE Proc. of High-Speed Networking and Multimedia Computing*, pp. 134–144, Feb. 1994.

[8] H. H. Taylor, D. Chin, and A. W. Jessup, "An MPEG Encoder Implementation on the Princeton Engine Video Supercomputer," *IEEE Proc. of Data Compression Conference*, pp. 420–429, 1993.