A REAL-TIME SOFTWARE DECODER FOR SCALABLE VIDEO ON MULTI-PROCESSORS

Wai-tian Tan *Avideh Zakhor**

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
e-mail: dtan@eecs.berkeley.edu, avz@eecs.berkeley.edu

ABSTRACT

Increasing heterogeneity of future networks is likely to result in transmission links with widely different available bandwidths. Scalable video compression has been shown to be an effective tool for video communication across these heterogeneous networks. For instance, the scalable algorithm described in [2, 3] can generate one embedded bit stream with tens of available bit rates ranging from tens of kilo bits to several mega bits per second. A multi-threaded software decoder for scalable video described in [2] is developed. The software can be configured for systems with varying physical memory sizes and different number of processors. Experiments were performed on a Sparc-20 workstation with 4 processors. Comparisons with results obtained from using 1 processor suggest that a speed up factor of 2 to 2.5 times can be achieved with all CPU's approximately 75 % loaded. As expected, the decoding speed scales with compressed video bit rate and picture size.

1 INTRODUCTION

With ever more video information stored in digital format, real-time video decoding has become an indispensable capability for workstations. The simplest application would be playing video archives from either a CD-ROM or over the network. Another example would be video conferencing. Currently, most real-time decoding mechanisms resort to expensive special purpose hardwares that become outdated easily. A software implementation is clearly superior not only in being cheaper, but also in being easily portable to virtually any machine with performance scaling automatically with processing power.

Real-time MPEG decoding is already happening [1]. However, MPEG lacks the multi-rate, multi-resolution capability as often desired for transmission over heterogeneous networks, and therefore needs to resort to transcoders for video delivery through low bandwidth links. Scalable video bit streams as described in Section 2, on the other hand, can gracefully degrade to lower bit-rates by simply dropping selected layers [2, 3]. For transmission over networks whose loads are bursty, having multi-rate property means we could simply decode at a lower rate during periods of high load and resume at higher rate during periods of low network load, without introducing jerkiness on the video being decoded.

Recently a scalable video compression algorithm has been proposed [2] that not only operates at a wide range of bit rates, from tens of kilo bits to several megabits per second, but also provides a fine granularity of available bit rates. In addition, these properties are achieved without loss of compression efficiency as compared to standard algorithms such as MPEG2. However unlike MPEG, the scalable algorithm in [2] requires symmetric computation power in encoding and decoding, making its decoding complexity quite high. For instance, an implementation of [2] on a Sparc-20 with 1 processor could decode only about 6.1 frames per second with color at quarter SIF resolution, i.e. 320×240. The use of multiple processors, which is arguably the most economical way to boost processing power, is one way to achieve real-time software decoding of scalable video bit streams.

For general purpose computers, concurrency is better exploited using multiple threads within the same process rather than the more conventional

*This work has been supported by SUN Microsystems, Philips, California State Program MICRO and Office of Naval Research

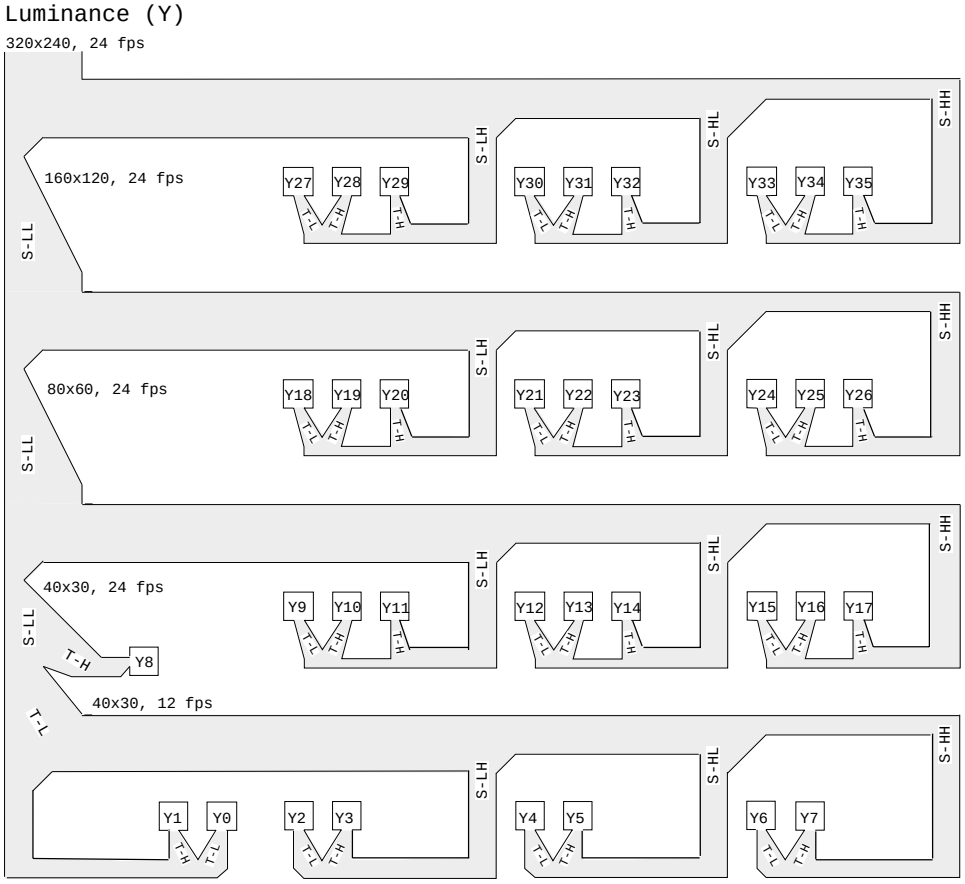

Figure 1: *Spatio-temporal subband structure. The letters S and T stand for "Spatial" and "Temporal" decompositions. L and H stands for "Low" and "High" frequencies.*

method of using multiple processes. The main motivation is that threads share the same address space and therefore have no need for expensive inter-process communications. Threads also require fewer kernel resources during execution and, depending on the particular implementation, do not necessarily involve a system call for switching [4]. A multi-threaded architecture can also provide better CPU utilization by overlapping CPU-intensive video decoding with I/O operations that are often blocking. An operation is called blocking when the program issuing it cannot continue execution until the operation is finished.

In this paper, we will describe the architecture of a real-time software decoder for the scalable video compression algorithm in [2], using multi-threading techniques, on a Sparc-20 with 4 processors. We will describe the resources needed for our implementation in section 4 and compare the results obtained on 4 processors to that obtained using only 1 processor in section 5.

2 THE VIDEO BIT STREAM

Video source at 24 frames per second is digitized into SIF size YUV video frames. The video frames are then encoded using the method described in [2]. Three-dimensional subband analysis is first performed on the video frames to generate a set of spatio-temporal subbands. Depending on how many levels of spatial decomposition one employs, these subbands can be of different sizes and visual importance. Each subband coefficient is then progressively quantized into layers where successive layers are a refinement of the previous layers. The redundancy in every layer is well exploited using conditional arithmetic coding [5] where the value of a subband coefficient in any given layer is coded conditioned upon the currently available values of its neighbors. Since

arithmetic decoding is a sequential process, it is important to divide larger subbands into smaller *subband blocks* and then encode these blocks independently of each other so as to limit propagation of transmission errors and to facilitate parallelism in computation.

Fig. 1 shows the subband hierarchy for the luminance component used in this paper. It is obtained by applying 2 levels of temporal decomposition and 4 levels of spatial decomposition. The chroma components are similar except with only 3 levels of spatial decomposition. All subbands of high temporal frequencies are coded using layered intra-frame PCM and zero coding techniques in [3] while the low temporal, low spatial frequency subband Y0 is coded using layered DPCM methods described in [3]. Coefficients in the low temporal, high spatial frequency subbands are coded using inter-frame techniques with reference to the corresponding subband coefficient in the previous frame in the same subband. For example, the subband Y9 in Fig. 1 is coded with reference to previously coded frame in subband Y9 while the subbands Y0 and Y10 are not.

In general, employing L levels of temporal hierarchical decomposition requires the subbands of 2^L frames to be decoded together. The experiments described in this paper assume $L = 2$ to achieve low latency. Thus, we work with *decoder buffers* which are large enough to hold the subband coefficients of 4 frames.

By simply discarding some quantization layers, we can get a coarser representation of the subband coefficients and operate at a lower data rate. Monochrome decoding is achieved by discarding the chrominance components. Decoding at smaller resolution is achieved by ignoring appropriate high spatial frequency subbands.

3 DECODER ARCHITECTURE

The decoder, being the inverse of encoder, needs to get the compressed video stream from either network or local storage, arithmetically decode the received stream to reconstruct the subband coefficients which are then used to synthesize back the video frames to be displayed. Each of these tasks defines a functional unit which is implemented using one or more threads, all running concurrently.

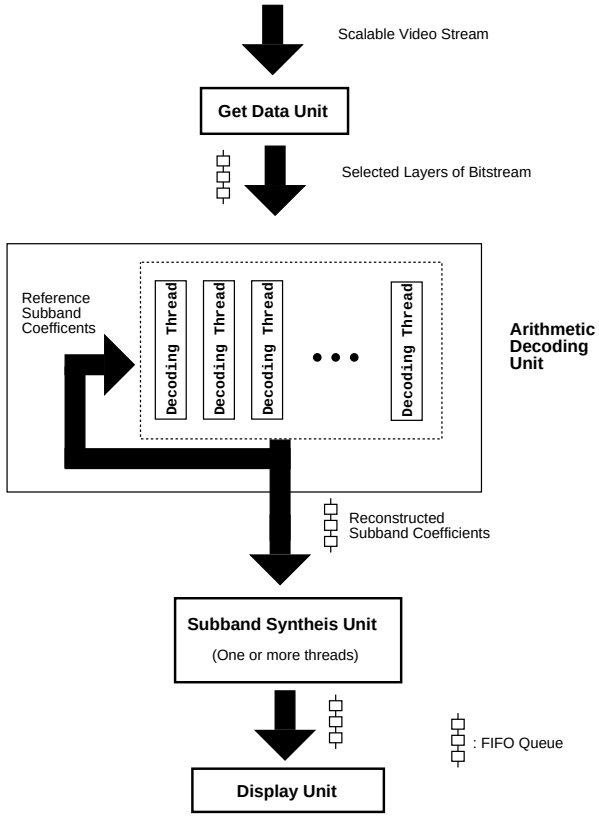

Figure 2: *Architecture of Scalable Video Decoder Software*

3.1 Get Data Unit

The Get Data Unit in Fig. 2 continuously reads the video bit stream, discarding layers that are not used for the bit rate the system is currently decoding.

The straight-forward implementation of issuing a read request and waiting for the data is undesirable for traditional single-threaded implementations. This is because it may take a significant time before the requested piece of data arrives, as in the case of reading from a remote device or from a network connection, idling precious computing power. The fore-mentioned blocking I/O however, has the advantage of being able to retrieve data almost immediately as it arrives. This feature is less important in a typical reading from disk applications where we have random access to the data but proves to be essential if one is getting the bit stream from a network. This is because many operating systems have small I/O buffers that can only hold a fraction of a second's worth of video. This defines a small time frame

within which one must either retrieve data from the buffer or face loss due to buffer overflow.

To avoid CPU idling, single-threaded programs often used non-blocking I/O to provide better utilization of computing resources. The price is increasing complexity and failure to retrieve network data as they arrive. For instance, asynchronous I/O as provided in some UNIX implemetations is an attractive option at the price of an interrupt per read to notify program of the availability of data. To summarize, single-threaded implementations either suffer from idling in the form of blocking I/O or overheads of non-blocking I/O.

For our multi-threading implementation, the Get Data Unit is a thread itself and reads data into a circular buffer using blocking operations. Before data arrives, only the unit is blocked and other threads can continue to execute. The Get Data Unit is implemented with a higher priority than other threads so that it can retrieve data from network as soon as possible. Thus, the implementation has the advantage of not idling the CPU as well as being able to retrieve data promptly.

3.2 Arithmetic Decoding Unit

The Arithmetic Decoding Unit creates and operates on decoder buffers. Compressed video is passed, not copied, from the Get Data Unit and is dispatched to a specified number of threads for decoding. All the subband blocks are divided among the decoding threads with the constraint that no two threads can work on the same subband block. Every decoding thread knows which subband block it is decoding and writes the result directly back to the corresponding buffer. The decoded subband coefficients are passed to the Subband Synthesis Unit.

A master-slave scheme is used in which one thread is in charge of dispatching subband blocks to other threads for decoding. Because the cost of dispatching might exceed the cost of decoding for smaller subbands blocks, only bigger subband blocks are dispatched,

Arithmetic decoding will not start unless the Arithmetic Decoding Unit has received enough compressed video data equivalent to a whole decoder buffer. The unit also makes sure that all the data in the decoder buffer is processed before proceeding to the next by having threads that finish early to wait. This not only enables the current decoder buffer to be processed faster, but also puts smaller memory requirements on the unit. Concurrency is derived from the fact that subband blocks can be decoded independently of each other, rather than the ability to process subband blocks in parallel.

Four threads are used for arithmetic decoding in the experiments that we carried out. Using more threads will incur higher cost for synchronization and is justifiable only if one has more CPU.

3.3 Subband Synthesis Unit

The Subband Synthesis Unit reconstructs the video frames and converts it from YUV format to RGB format for the Display Unit to display. The reconstruction of YUV frames involves mainly filtering operations and as such, can easily admit a parallel implementation. More than 1 thread is used for subband synthesis only when the physical time for arithmetic decoding of a frame using many threads is shorter than that of subband synthesis using 1 thread. This is because using more threads adds more overheads. If the decoder is busy doing arithmetic decoding most of the time, i.e. in the high data rate, low resolution scenario, performance is better with only 1 thread for subband synthesis. Depending on the relative computational requirements for subband synthesis and arithmetic decoding, more threads are used for subband synthesis, particularly in the low data rate, high resolution case.

3.4 Display Unit

Outputting video to the display device can take considerable time if we cannot directly write to the device buffer, as is often the case. In an X windows implementation, it involves transmission of tens of uncompressed video frames per second if shared memory is unavailable or not used. For instance, using a plain X11 routine to display an 8-bit SIF size frame takes tens of milli-seconds on a Sparc 20. The implementation of the Display Unit as a separate thread is necessary in this case to avoid spending tens of milli-seconds waiting for one frame to be put to the display. With the shared memory extension of X windows, we have to wait for acknowledgement from the X server before putting up another frame. With the coding scheme of [2, 3], the time it takes to decode one frame varies considerably and it is sometimes

possible that a frame is ready for display before we receive acknowledgement for the previous frame. In that case, one has to wait and idle. The implementation of the Display Unit as a separate thread allows the rest of the decoder to proceed while it waits for completion of the display.

3.5 Connecting Units

No two units can operate on the same piece of data at any one time to guarantee data integrity. Units are connected using queues of short lengths, typically 2 or 3, that act as both input and output buffers. While requiring more memory, this design provides better performance because it allows for statistical variations in the finishing times of the units. Data processed by one unit is passed to the next unit using these queues. When a unit cannot get from a queue or pass data to a queue, it blocks until the operation is possible. Note that we have a linear structure of units and as such, there will always be at least one unit running. However, for faster and smoother decoding, we use sufficient threads in the most computationally expensive unit to make sure it is never blocked. Fig. 2 may appear to suggest the use of pipelining, but since computing resources are dynamically allocated to the units that need them and not tied to a particular unit, there is no bottleneck if one unit performs slower than the others unless the number of threads in all the unblocked units is less than the number of processors.

In our current implementation, computational load cannot be evenly distributed among all arithmetic decoding threads because subband blocks of the same size may have different number of layers. Also, the thread implementation that is available on our machine is non-preemptive in the sense that a thread that seizes a processing unit will run into completion before another thread can use the same processing unit, unless the executing thread explicitly relinquishes control. Thus not all threads are truly executed concurrently but are serviced one by one.

4 RESOURCE REQUIREMENTS

In this section, we will estimate the extra memory that is needed to run our decoder on multiprocessor machines. For the architecture described in section 3, the Arithmetic Decoding Unit always keeps a decoder buffer to work on. For arithmetic decoding to run concurrently with other units, we need at least another decoder buffer. Thus, the absolute minimum number of decoder buffers needed is two. For decoding at SIF resolution and full color, and employing 2 levels of temporal hierarchical decomposition, one decoder buffer requires:

$$\begin{aligned} &\text{display size} \times \text{bytes per pixel} \\ &\times \text{number of frames per decoder buffer} \\ &= (240 \times 320) \times 2 \times 4 \\ &= 614.4 \text{ k bytes} \end{aligned}$$

for luminance and $614.4/2 = 307.2$ k bytes for both chroma components. This is because each chrominance component is sub-sampled by 2 both in the vertical and horizontal directions. Thus we require 921.6 k bytes per extra decoder buffer. The number of bytes per pixel is 2 because the subband coefficients are stored as 16 bit short integers while decoding.

Because the low temporal, high spatial frequency subbands are inter-frame coded, it is necessary to store the corresponding reference frames while decoding. For the reference subbands, we need at most $921.6/4 = 230.4$ k bytes which is big enough to hold all low temporal frequency subbands.

For quarter SIF resolution, the memory required per buffer is one quarter that of SIF resolution. For monochrome decoding, the buffer sizes are 2/3 that of color decoding.

In practice, the best configuration depends on the actual number of processor and physical memory a system has.

5 RESULTS

Before describing the results, it is important to point out that even though arithmetic coding provides good compression ratio, it is computationally expensive. When decoding at 3 M bits/s, most of the computation in our implementation resides in the layered conditional arithmetic decoding, accounting for about 50 to 70 % of CPU time depending on resolution and whether we include color. Actual synthesis of the frames from subband coefficients takes less than half of what arithmetic decoding takes at 3 M bits/s.

Tables 1 - 4 compare the decoding rates our decoder performs as compared to an earlier uniprocessor implementation, running on the same Sparc 20 station. The uni-processor version has

Data Rates	3 M bits/s	1 M bits/s	500 k bits/s	300 k bits/s
Frame Rates with 4 Processors	5.0	8.0	10.5	13.0
Frame Rates with 1 Processor	1.9	3.5	5.2	6.5

Table 1: Decoding Rates at SIF Resolution with Color

Data Rates	3 M bits/s	1 M bits/s	500 k bits/s	300 k bits/s
Frame Rates with 4 Processors	6.8	10.5	11.5	15.0
Frame Rates with 1 Processor	2.7	4.3	6.5	8.5

Table 2: Decoding Rates at SIF Resolution, Monochrome

Data Rates	3 M bits/s	1 M bits/s	500 k bits/s	300 k bits/s
Frame Rates with 4 Processors	8.0	13.8	18.0	21.2
Frame Rates with 1 Processor	3.3	6.1	8.8	10.0

Table 3: Decoding Rates at Quarter SIF Resolution with Color

Data Rates	3 M bits/s	1 M bits/s	500 k bits/s	300 k bits/s
Frame Rates with 4 Processors	13.5	19.5	24+	24+
Frame Rates with 1 Processor	6.0	9.0	12.0	14.5

Table 4: Decoding Rates at Quarter SIF Resolution, Monochrome

one processor fully loaded while decoding whereas the multi-processor version has all 4 CPU about 75-80% loaded on the average. More arithmetic decoding threads are used for high than low bit rate decoding to reduce overhead.

Note that decoding rates are higher than that obtained with a single processor by 2 times in the monochrome, low bit rate regime, to 2.5 times in the color, high bit-rate regime. The inclusion of color enhances concurrency by having simultaneous decoding of chrominance and luminance components. At higher bit rates, more computation is required per subband block and thus a lower percentage overhead for decoding in parallel.

When stepping down from SIF resolution to quarter SIF resolution, decoding rates are increased by 60 % at 3 M bits/s and 63 % at 300 k bits/s. One reason is that it takes 4 times more operations to synthesize one video frame from its subband coefficients at SIF resolution and this cost becomes more important at lower bit rates. Another reason is that it takes less computation to decode more quantization layers of fewer subbands than fewer quantization layers of more subbands. This has to do with the fact that for a fixed total number of bits, it takes longer to decode more shorter arithmetic codewords than fewer, longer ones.

For quarter SIF resolution at 1 M bit/s, with color, we can currently decode at 13.8 frames/second which is half the rate that is needed for real-time decoding.

REFERENCES

[1] L.Rowe, K.Patel, B.Smith, and K.Liu, "MPEG Video in Software: Representation, Transmission, and Playback," *IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology*, San Jose, February 1994.

[2] D.Taubman and A.Zakhor, "Highly Scalable, Low-delay Video Compression," *Proceedings ICIP*, 1994, Vol 1, pp 740-744.

[3] D.Taubman and A.Zakhor, "Multirate 3-D Subband Coding of Video," *IEEE Trans. Image Proc.*, September 1994.

[4] M.Powell, S.Kleiman, S.Barton, D.Shah, D.Stein, and M.Weeks, "SunOS Multi-threading Architecture," *Proc USENIX Winter Conference*, 1991.

[5] G.Langdon and J.Rissanen, "Compression of Black-White Images with Arithmetic Coding," *IEEE Trans. Commun.*, vol. COM-29, pp 858-867, June 1981.