EFFICIENT BUFFERING CONTROL FOR A SOFTWARE-ONLY, HIGH-LEVEL, HIGH-PROFILE, MPEG-2 DECODER

Ju Wang, Jonathan C.L. Liu, Yishu He

Department of Computer, Information Science and Engineering
University of Florida, Gainesville, FL
{jwang, jcliu, yhe}@cise.ufl.edu

ABSTRACT

A high-quality MPEG-2 software decoder should support a good scalability performance for a wide range of video format, especially for the high-resolution MPEG-2 video (e.g., HDTV). However, it is found that the existing parallel decoder suffers significant performance degradation when decoding high-level MPEG-2 video with the full system configuration, due to inefficient management of memory space in the decoder. We propose an efficient buffer management mechanism such that the memory requirement can be reduced by 50%. This is approached by two steps, first we use a ST scheme to minimize the transmission buffer in a slave node by allowing dynamic sharing between frames in one group of picture (GOP). Then we further reduce the buffer space by a dynamic buffer allocation according to image type. The revised parallel decode showed a satisfactory scale-up performance when decoding the high-resolution video formats.

1. INTRODUCTION

High-quality MPEG-2 software decoder needs to support good scalability performance across different video resolutions. For example, video resolutions should be supported from medium resolutions (e.g., 704*480) to large resolutions (e.g., 1404*960) with guaranteed display quality. Recent development of high performance microprocessor and software technology has made it possible for real-time MPEG-2 decoding. Especially, multimedia instructions were introduced in many processor architectures, and were used to accelerate MPEG-2 decoding process [3]. With the Pentium III 700MHz CPU, the main level MPEG-2 video (DVD quality) can be decoded at nearly jitter-free quality. Some commercial software DVD decoders can operate on a lower CPU clock rate with hardware multimedia features provided by the video card [5].

However these approaches have not support high-quality scalable MPEG-2 video formats that are becoming more and more popular. Thus a generic pure software solution is still desirable in many cases. Pure software MPEG-2 encoding/decoding requires large amount of computation power and are usually parallelized [1][2]. We have been researching a generic, portable, pure-software MPEG-2 codec for the last few years. However, software-only MPEG-2 decoding is very computation intensive, especially for high-level video format with multiple layers. To achieve high performance software MPEG-2 decoding, we had designed a parallel MPEG-2 decoder that can run on both cluster and multi-processor environments [4]. The results were very promising with 30-fps playback achieved with 4 Pentium 400MHz desktop computers, and a 72-fps HDTV frame rate achieved in a SUN SMP environment. However, challenge arises when attempts are made to support (1024*1024) and (1404*960) high-level MPEG-2 video. We have observed a severe performance degradation (e.g., dropping from 18 or 20 fps to 2.5 fps) when more than 10 slave nodes are used.

By analyzing the runtime system resource utilization, we found that the system memory is quickly exhausted when increasing the number of slave nodes. When decoding the video file with high spatial resolution, the increase of memory usage eventually becomes a system bottleneck. To address the challenge and obtain high scalable decoding for high resolution video, we proposed and implemented two revised memory management approaches to reduce the buffer requirement. The first is Minimum Transmission Buffer in Slave Node (ST scheme). In our original design, the slave nodes allocate the buffer for the whole GOP. When the number of nodes grows, a lot of memory is needed. To reduce the memory requirement, we reduce the transmission buffer size in the slave nodes to three frames. In the second approach, we proposed a dynamic buffer scheme which is adaptive according to the current frame type. For the I-frame, the system will only request one frame buffer. When P- or B- image are to be decoded, 2 frames will be allocated. The effective number of frames per buffer is only 85% of the 3 frame buffer. Furthermore, dynamic buffer allocation can be applied inside the decoding of each frame. The experimental results show that the buffer space is significantly reduced, and we observed a well-scaled decoding performance for the high resolution MPEG-2 video. A satisfactory performance of 26-fps is observed when 14 slave nodes are equipped.

The organization of the paper is as follows: Section 2 discusses our preliminary observation of the decoding performance for high-resolution video. The nature of the problem is identified by analyzing the original buffer scheme and run time system statistics (CPU usage, memory occupation). In Section 3, we present the two improved buffer schemes and report the experimental results. Finally, Section 4 gives the conclusion of this paper.

2. PROBLEM NATURE AND ANALYSIS

We have long suspected that the scalability issue for high-profile high-level MPEG-2 video could be a challenging issue, but it was not until recently that we found this practical issue does exist. The problem is observed during an experiment to decode a serial of MPEG-2 videos in different spatial resolutions, from 352*240 to 1404*960. Using the same hardware configuration as in the SUN SMP environment in [4], the expected decoding performance is
shown in Table 1.

<table>
<thead>
<tr>
<th>Video Spatial Resolution</th>
<th>2 node</th>
<th>4 node</th>
<th>8 node</th>
<th>16 node</th>
</tr>
</thead>
<tbody>
<tr>
<td>352*240</td>
<td>25 fps</td>
<td>55 fps</td>
<td>120 fps</td>
<td>260 fps</td>
</tr>
<tr>
<td>704*480</td>
<td>10 fps</td>
<td>25 fps</td>
<td>48 fps</td>
<td>92 fps</td>
</tr>
<tr>
<td>1024*1024</td>
<td>3 fps</td>
<td>8 fps</td>
<td>20 fps</td>
<td>34 fps</td>
</tr>
<tr>
<td>1404*960</td>
<td>3 fps</td>
<td>6 fps</td>
<td>14 fps</td>
<td>25 fps</td>
</tr>
</tbody>
</table>

Table 1: Expected Parallel Decoding Performance

The achieved frame rates for the low- and main-level MPEG-2 video are very close to our prediction. For the small resolution video we observed a linear increasing of frame rate. The maximum frame rate is achieved when 14 nodes are deployed, providing 220 fps for tennis, 231 fps for calendar, and 213 fps for flower. The performance for the three video titles show little difference in terms of frame rate. Each of them increases close to linearly when more slave nodes are used. The highest frame rate achieved is 70 fps (with 14 nodes). However, the scalability performance for the high resolution MPEG-2 videos are not satisfactory. In Figure 1, the decoding rates for (1404*960) MPEG-2 files are illustrated. Starting with 2 fps at single node configuration, a linear increase can be observed. The highest decompression rate is 20 fps for "flower" at 9 slave nodes, and 22 fps for "tennis" and "calendar" at 11 nodes. With 10 slave nodes, the decoding performance of "flower" suddenly dropped to 2.5 fps, and continued deteriorating with a small rebound at 11 slave nodes. For "tennis" and "calendar", a similar performance degradation is observed at 12 slave nodes, right after the peak performance.

![Figure 1: Decoding Frame Rate for 1404 x 960](image1)

Similar performance degradation is observed for the (1024*1024) format. The system can do well for up to 10 slave nodes, where a peak of 23 fps can be achieved with 10 nodes. However, decoding rate reduces significantly after 11 nodes. The frame rate dramatically drops to only 2 fps, which is even worse than a single slave node configuration. Further increasing of the slave node seems having negative effects in decoding performance. We observe no frame rate improvement after 11 nodes.

2.1 Memory Usage Analysis

In order to identify the system bottleneck which cause the degradation of decoding performance for high-resolution video, we record the utilization of system resources during the decoding process. The evidence from system runtime statistics can be collected from the CPU time distribution and the number of page faults to support this unique observation\(^1\). Figure 3 illustrates our measured number of page faults versus the number of slave nodes and the CPU statistic. For the sake of space-saving, we only present the results for "tennis".

For the 352x240 video, the number of page faults virtually remains unchanged, and is kept at a low level (1010 page faults/frame). Increasing the video resolution to 704x480 is reflected by a rise of the page fault number, a four fold jump is observed. Nevertheless, the 704x480 case still has a flat curve for the increasing slave node, indicating the system is running steadily. For the 1024x1024 video, the number of page faults increases considerably. It is noticed that the page faults significant increase at 10 to 12 slave nodes, reaching 3500 page faults per frame at 12 slave nodes as peak. Compared to the decoding performance in Figure 1.b, the period of high page faults coincides with the collapse of the decoding rate. This indicates that the excessive page faults had driven the system into an outrage state. The page faults behavior of the 1404x960 video shows the same pattern as in the 1024x1024 case.

![Figure 2: Page Fault VS Number of Slave Node](image2)

The excessive increasing of page faults is also indicated by the CPU usage. With one slave node, 90% of the system time is idle, 8% of the CPU time is used in the user space, and the remaining 2% for other system maintenance. When increasing slave nodes, the user space time increases proportionally, and the system idle time decreases. After 8 slave nodes, however, both system idle time and user space time drop significantly, while the system overhead shows a major increase. About 90% of the CPU time is used by the operating system, while user space only occupies 5% of CPU time. Recalling that the page faults number increases suddenly at 9 slave nodes (see Figure 3), we conclude that the system spend most of its CPU time swapping page in/out, thus the observed drop of decoding performance.

![Figure 3: CPU usage](image3)

It is worthy to point out that the decoding performance can be significantly affected by cache hit ratio. Nevertheless, the MPEG-2 decoding is a very data-intensive process where data locality is limited and very difficult to be exploited. Thus our parallel decoder

\(^1\)The CPU utilization could be obtained by system call `time()` in UNIX system, and the page fault is recorded by a utility process `truss` spawned by the decoding processes.
is not optimized in term of cache performance. Realizing that page fault is directly related to the shortage of system memory, we believe the buffer management of the parallel decode should be investigated. A not-optimized buffer scheme will devastate the competition between user processes (e.g., our communication and decompression software) and system processes (e.g., demand-paging mechanisms by OS). Because the shortage of the overall memory, the system process will generate a significant number of page faults, which in turn slowing down the decompression speed due to the lack of CPU.

The memory requirement for the master node and slave node can be expressed:

\[ \begin{align*}
M_m &= m_c + m_{streambuffer} + m_{outbuffer} + m_{inbuffer} \\
M_s &= m_c + m_{compressedbuffer} + m_{transmissionbuffer}
\end{align*} \]

Here \( m_c \) is the size of executable code for the master, about 500 KB. \( m_{streambuffer} \) is the streaming buffer to receive the compressed video packet from the video server, we currently fixed it to be 1 MB. \( m_{outbuffer} \) and \( m_{inbuffer} \) are dedicated for information exchange in the parallel decoding. \( m_{outbuffer} \) equals one GOP of MPEG-2 compressed frames, and \( m_{inbuffer} \) needs to accommodate two GOP of decompressed frames (one GOP for displaying and another for incoming traffic).

For the test stream tennis40 (1404*960), the corresponding memory requirement in master/slave sides are: \( M_m = 42 \) (MB) and \( M_s = 30.8 \) (MB). The accumulative buffering space will grow quickly when using a large-scale slave node configuration, which causes unsatisfactory scalability performance when the number of slave nodes is large. For instance, let \( N \) be the number of slave nodes, the total memory requirement becomes

\[ M_t = M_m + N \times M_s \]

Using the parameters of our testing MPEG-2 video, the total memory used can be estimated from above equation. For the video Tennis40, we need about 73 MB, 104 MB, 165MB, 319 MB and 381.5 MB respectively when the number of slave node are 1, 2, 4, 9 and 11. In the next section, we will discuss several techniques to reduce the buffer space.

### 3. EFFICIENT BUFFERING SCHEMES

It is noticed that the slave node allocated a GOP length of frame buffer originally, which could be further optimized to the minimal buffer space. However, due to the decoding dependency inside the MPEG-2 video structure, we are not able to use only one frame buffer. To decode a B-frame, we need two reference frames and one decoding working frame, resulting a total of 3 frames. With a careful redesign of the master-slave communication protocol, using a 3-frame transmission buffer in the slave side is possible, which we called the ST scheme. When the picture size is 1024*1024 and GOP=15, we can save about 12 MB buffer space per slave node, about an 80% reduction in the slave side.

The minimum required frame buffer can be further decreased from 3-frames to 2 frames. For the I- or P- frames, we need one buffer for the prediction picture, and another buffer for the working frame. The two buffers change their role after decoding a I- or P-frame, so that the most recently decoded I- or P-frame is used as the prediction frame for the next P-frame. For the B-type frame, since the decoded B-frame will not be used as reference frame, we can directly send decoded blocks into the network channel to the master without storing them. The above discussion assumes the reference frame of P-frame is always the last decoded P-frame, and the reference frame for B-frame are the last two P-frame. Nevertheless, this approach also works if the B- and P-frames always refer to the I-frame with the proper setting of the reference buffer.

With this scheme, the expected memory requirement becomes

\[ M_t' = M_m + N \times (m_c + 1.5 \times 3 \times m_{frame} + m_{inbuffer}) \]

Here the buffer space of the master node remains the same. The memory required increases at a slow slope, where each slave node will introduce only about 6 MB additional space for the tennis40 video. The tennis60 finds an even smaller memory requirement. To find out the number of maximum slave nodes before system memory runs out for the 1404*960 single layered MPEG-2 video, we have \((47 + (N-1) \times 6) < 300 \) MB (use 300 MB as a system threshold). This will give \( N=43 \) slave nodes and more than 60 fps.

### 3.1. Further Optimization in the Slave Nodes

In order to handle high profile MPEG-2 video stream with high resolution and multiple extension layer, we propose a dynamic buffer management scheme to further reduce the buffer space. It is observed that the decoding procedure in a slave node might not use three frames all the time. More specifically, the I-frame need only use one frame buffer, while P-frame can be decoded with two buffer. Only B-frame needs the whole three frame buffers. Thus the total amount of needed buffer can vary during the lifetime of the slave node. By allocate frame buffer dynamically according to the frame type, It can be expected that the total buffer can be significantly reduced for high quality video where I- and P-frame represent a considerable portion of the frames. Let the ratio of I, P, B frames in a GOP structure be \( a:b:c \), the effective buffer space for one layer is expressed by

\[ M = (1 + a + 2 + b + 3 * c)/(a + b + c) \]

In a typical GOP structure of “IBPBBPBBPBBPBBP”, we have \( a:b:c = 1:4:10 \), this will result in an effective buffer number of 39/15=2.6, which is about 85% of the 3 frames buffer scheme. The effective buffer space is a function of the GOP structure. When the percentage of I frame increases, the effective buffer space will decrease.

The concept of dynamic buffer allocation can be applied inside the decoding of each frame. Since the decompression of each frame is based on a serial decompression of macroblocks, the overall buffer space could be reduced by dynamically allocating buffer for macroblocks. For example, when decoding the first macro-block, we only need allocate a 16*16 block space. The other macroblocks buffer will be assigned when it is needed for decoding. The total buffer will grow as more macroblocks are decoded, and will reach the maximum full buffer size after the decoding is finished. Then the slave will keep the full buffer size until the frame is able to be discarded. After the decoded frame is sent back to the master node, the decoding buffer can be released and a new buffer growing process will be started for the next frame. With this dynamic memory allocation, we can expect an additional buffer reduction of 0.5 frame for the current frame. Notice that this scheme can not reduce the amount of buffer for the reference frame, which should be in system through the whole process. The effective buffer requirement become \( M =\)
\[(1 - 0.5) \cdot a + (2 - 0.5) \cdot b + (3 - 0.5) \cdot c)/(a + b + c).\] Using the same GOP structure as the above, the effective frame number of the buffer in slave node is 2.1, which is 60% of the 3-frame buffer scheme.

The dynamic allocation of buffer in the slave node is an application level memory management scheme, which is closely embedded in the decoding process. The current implementation rely on some system-provided routines (e.g., malloc and free). We speculate that a customized buffer management routine (direct access of the system memory) should be able to further increase the decoding performance, which should be discussed in our future research. The tradeoff here is the additional CPU cost introduced for the dynamic memory management. For each macro-block, the additional cost includes at least two system calls (for memory allocation/deallocation) and some other miscellaneous operation. It has been showed that the cost associated with dynamic memory allocation is significant for the database server and Web-server, where thousands of processes may co-exist to processing user requests. In our case, the number of slave nodes/processes is usually below 20 and it is expected that memory management activity is far less frequent, thus the overhead introduced should be limited. This is confirmed by our experimental results by comparing the performance of the decoding with/without dynamic memory allocation. With dynamic buffer allocation enabled, the overall decoding time is increased less than 7% than the static memory allocation case.

3.2. Implementation and Experiment Result

We implemented the ST memory allocation scheme and the dynamic frame buffer allocation. Our results show these two improvement along can bring significant memory reduction, and the phenomena of paging panic is eliminated. Using the 1404*960 video as reference, the total buffer size is 53.5 MB with one slave node, which is 27% less than the original one. For 4-slave-node case, the ST scheme use 85 MB instead of the original 165 MB, which is almost 50% memory saving. The system requires the maximum of 200MB when all 14 nodes are utilized, which is smaller than the system panic point where 300 MB is consumed as mentioned in section 2. Furthermore, the number of page faults for each individual node is significantly reduced. For the 1404*960 case, the number of page faults shrinks from 1500 to 1200, at one slave node configuration. For 1024x1024 video, the page faults are now 943, 25% less than before. For all of the video streams, the number of page faults almost remains unchanged when increasing the number of the slave nodes.

Figure 4.a show the scalable decoding frame rate for 1404*960 video of our revised ST scheme. We observed a close-to-linear increase of the frame rate. For one slave node, we have 1.7 fps for tennis, 1.86 fps for flower, and 1.6 fps for the "mobli" video. For two slave nodes, the performance is nearly doubled for each case, with 3.5 fps for tennis, 3.7 fps for flower, and 3.2 fps for mobli. For the other node configuration, the achieved frame rate increase proportionally, and there are slight difference between the three video titles. The peak decode rates are obtained at 14-slave nodes, where 20 fps is observed. The decoding performance for 1024*1024 video files shows a similar behavior. Thus our revised buffering scheme has successfully solved the memory shortage problem, and works well for high quality video up to MP@HL video.

Figure 4.b shows the overall CPU time distribution of slave nodes when decoding high-resolution video formats with the revised buffer scheme. The user space component represents the computation time for the MPEG-2 decoding procedure, the kernel space time is for the system level overhead, including time spent in the network layer, system call, and other costs. It is observed that the user space time increases linearly when the number of slave nodes increase, accompanied by a corresponding decrease in the system idle time. Meanwhile the operating system level cost is maintained in a low level (between 5% to 10% of total CPU time). For the large scale experiments (more than 11 slave nodes deployed), the abnormality cross-over of the user space time and system overhead observed in the original decoding experiments no longer exists. This further proves the effectiveness of the ST scheme in solving the memory shortage.

4. CONCLUSION

Due to the limited memory to support a scalable performance for high-level high-profile MPEG-2 video resolutions, new buffering controls and mechanisms need to be created within our software-only parallel MPEG-2 decoder. We thus propose an ST buffering scheme with a dynamic allocation algorithm to significantly reduce the memory demands within this parallel decoding software. The results are very promising with excellent scalability performance achieved in both down-scaling and up-scaling capability.

5. REFERENCES