# Experiences in Tuning a Software MPEG Player[1]

*Paul C. H. Lee*

Institute of Information Science
Academia Sinica, Nan-Kang, Taipei, ROC

**Abstract**

*An MPEG audio/video player in software is implemented on a commercial operating system and is detailed evaluated. The primary purpose of this work is to explore new system technologies in QoS management for advanced multimedia applications. Here, we address the performance analysis of this software-MPEG-II player and report the experiences in performance tuning and QoS enhancements via empirical experiments.*

## 1. INTRODUCTION

Due to the popularity in multimedia and the advances in digital technologies, the demands on multimedia standards are important and urgent to the world. To unify the specification in motion videos, the MPEG standard[1] is defined to meet the versatile requests.

MPEG decoders can be implemented in software or in hardware. Hardware decoders have higher processing performance than decoders implemented in software. The added costs in extra hardware, system maintenance and system upgrades, however, are the primary drawbacks. Software decoders, on the other hand, have advantages in providing versatile and adaptive functions. For example, when a system is overloaded, to dynamically adjust the quality of services (QoS) in playbacks is not easy using hardware MPEG decoders[2]. It is also costly for hardware decoders to support functions like partitioning the screen into several parts for different streams. This function can be seen in current TV products for playing two channel TV programs simultaneously.

The software decoders, however, consume a lot of computing power, which always request high-end processors in real-time decoding the audio/video streams. This is a serious drawback to implement decoders in software. However, we believe that it will not be a problem in the future. Due to the facts that higher speed processors are created, cheap and powerful enough processors can be expected in

[1] This work is sponsored in part by National Science Council, Taiwan, R.O.C., as granted in NSC 88-2213-E-001-016.

the near future[3]. Particularly, new created processors are endowed with multimedia extensions[4]. These enhancements also help the multimedia computing.

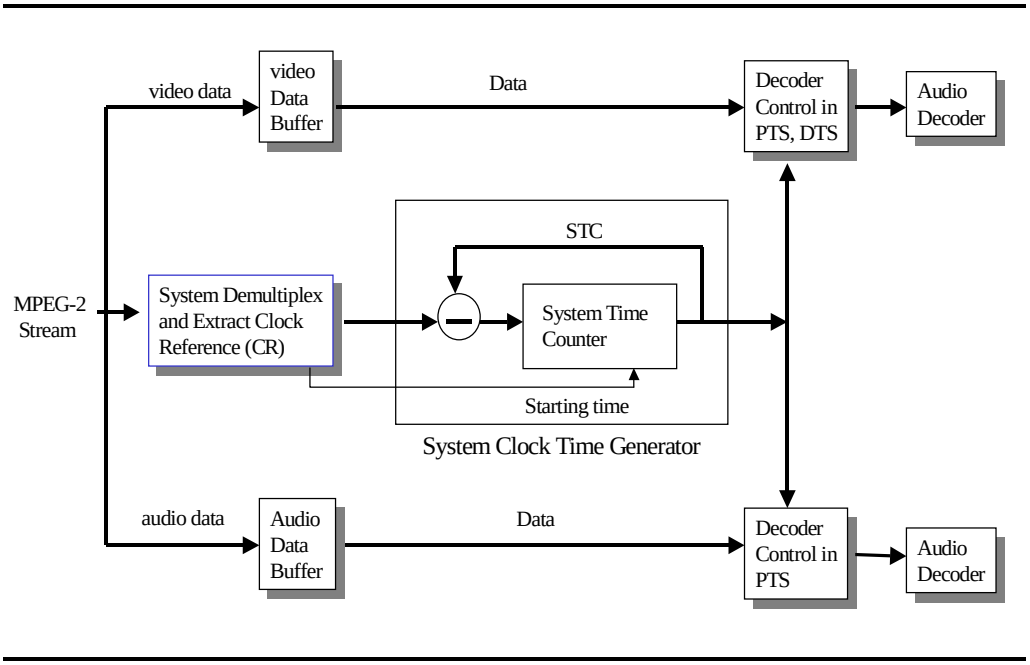
In this report, we present results and experiences in implementing an MPEG audio/video decoder in software. The video part is achieved by enhancing a shareware[5]. The audio part and the MPEG-II Systems[6] are implemented and integrated with the video decoder. This player is built on commercial operating systems[7]. The rest of this report is organized as follows. Section 2 briefly describes the fundamentals for MPEG standards. Section 3 is about the performance analysis and lessons. An adaptive QoS framework for supporting adjustable playback qualities is described in Section 4. Section 5 concludes this report.

## 2. MPEG DECODING

### MPEG Systems

The MPEG standard defines the stream as a sequence of still frames. Data within the streams are constructed from many elementary streams, such as audio and video elementary streams. Each stream is partitioned into pieces of packets and constructed in hierarchy, from blocks to macro blocks, to slices, to pictures, then to Group of pictures (GOP), and finally the video sequences. The MPEG Systems is the specification that defines the headers for combining the data from different elementary

Figure 1: architecture in decoding MPEG streams



streams. When audio stream is received, the header is parsed and the data is put on the buffer used for playing audio. The same procedure is used for video data. The system architecture is shown in Figure 1.

**MPEG Video**

The video compression in MPEG standard is achieved by three techniques. The first is the transform coding. Since human eye is insensitive to high frequency visual information, through certain mathematical transforms, the image can be presented in fewer values by concentrating the energy of an image. The IDCT is applied. The second technique is called motion compensation. It is based on the fact that a frame may be similar to its predecessor. Thus a stream does not need to record each object's information in each frame. Instead, only a vector recording the movement of similar image information is kept. Through this vector, an image can be recovered by reference the previous picture frame.

Motion compensation causes some frames to reference its predecessor. The frame types are classified as I frame, B frame and P frame. I frame is self-contained that it does not need to reference other frames. P frame and B frames need to reference other frames.

The third technique is called the run-length coding. The data that compressed by DCT transforms and motion compensation will be represented as a sequence of numbers with a lot of zeros. Through the run-length coding, these zeros will be dropped and the data is represented in another form, which data size is largely reduced.

**MPEG Audio**[7]

In ISO 13813-3, MPEG-II defines the audio in 3 levels, which have different compression rates, audio qualities and computation complexity. The layer 1's compression rate is about 1:4. And it is 1:6~1:8 for layer 2 and 1:10~1:12 for layer 3.

The concept for audio compression is based on the fact that human ears are insensitive to some frequencies. By making use of Frequency Masking and Temporal Masking, the audio frequencies that cannot be captured by human ears are dropped. In this work, our decoder supports three layers in audio processing.

## 3. PERFORMANCE ANALYSIS

In this Section, we present the performance evaluations and the analysis of the MPEG-II decoder. The working platform is a PC in 233MHz with MultiMedia eXtensions (MMX). It runs on a

commercial operating system. The DRAM size is 64 megabytes. The level 2 cache is 512 kilobytes.

**MMX Impacts and Performance Bottlenecks**

We enhance the decoders by MMX in IDCT, Motion Compensation and IDCT parts. Other parts such as moving data in blocks and the component to parse the input streams are also enhanced by MMX by moving data in parallel and making use of MMX's large register sets. There are 12 MPEG-II streams are played. Table 1 lists the number of frames being played. This is used to show the performance improvements via MMX.

In this first evaluation, we found that MMX does help in increasing the frame numbers being played. The speedup is from 2.92 to 3.46 for different video streams. This is very different to previous published data, which showed MMX could only increase performance at most 20% to 40%, and the fact that memory bandwidth is the bottleneck, not the processor[5,9]. The observed fact is that the speedup is too good to be true.

The possible reasons for above reported numbers can be illustrated as follows. First, when using MMX technology, the decoder is written in assembly codes, because there is no compilers can automatically detect and generate efficient binary codes using MMX. The original player, however, used C language to implement the programs. The binary code generated from C has sizes more than 10 times the one implemented in assembly language. This causes the CPU consumption differs in more

Table 1: performance enhanced by MMX

| Stream name | No MMX | W /MMX | Stream size | Resolution |
|---|---|---|---|---|
| gi9 | 2.14 | 6.62 | 61085.00 | 720 * 480 |
| gitape | 2.36 | 7.09 | 60934.89 | 720 * 480 |
| hhilong | 1.90 | 6.14 | 18318.22 | 720 * 576 |
| mei60f | 2.43 | 7.29 | 62567.59 | 704 * 480 |
| tecla-10 | 1.95 | 6.24 | 16494.31 | 720 * 480 |
| sony-ct2 | 2.30 | 6.76 | 77224.90 | 704 * 480 |
| tech_v2 | 1.93 | 5.82 | 72704.65 | 720 * 576 |
| test1 | 2.97 | 10.16 | 20815.48 | 720 * 480 |
| utest0 | 2.91 | 9.87 | 20927.22 | 720 * 480 |
| cook3 | 2.94 | 10.18 | 16683.55 | 704 * 480 |
| foodlogo | 4.26 | 12.43 | 24985.05 | 704 * 480 |
| volcn | 2.82 | 9.47 | 27552.07 | 704 * 480 |

than 20 to 30 times for implementations with MMX and without MMX, also causing the high speedup for each evaluation.

Memory bandwidth is not the bottleneck because in these evaluations, the data streams are small, but the cache size is large. In our system, we have 512 kilobytes cache, but all benchmark streams have averaged frame sizes lower than 512 kilobytes. Most data accesses are on cache, not on memory directly. This can be observed that smaller data streams have higher speedups using a same player. The data in Table 1 shows the results.

**Impacts due to operating system overhead**

Another interested question is about how high the costs created from the operating system, because originally we expected that there should be at least 10 to 12 frames played per second in 233 MHz processor and MMX technology. But we cannot reach these numbers for every stream.

To figure out the overhead created from operating systems, we port this player to an embedded kernel[10-11] that we developed for building network appliances. We use the same player and play the same streams but run the player in a different kernel, a lighter kernel that is far smaller than the commercial kernel. The only difference is that the player does not output the video data to the display card in this experiment. Next section has showed that *display* occupies almost the same amount of time, which depends most on the display card's performance, not on the system behavior or processor computing power. We also modified the major components in assembly codes when MMX is not applied. The purpose is to let the comparison as fair as possible, even though the compilers we used are not the same. In this example, we use shareware compiler GCC. And in previous experiment, a

Figure 2: Running on top of embedded kernel. Presented in playing frames per second.

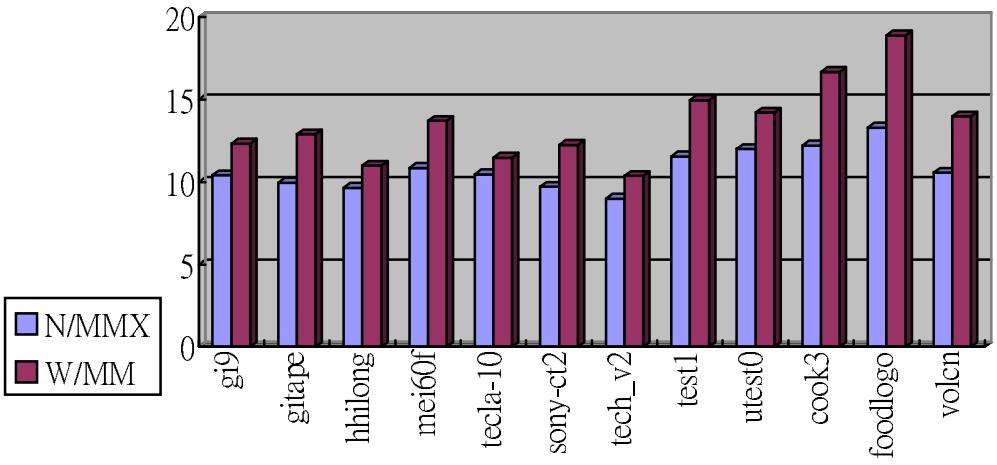
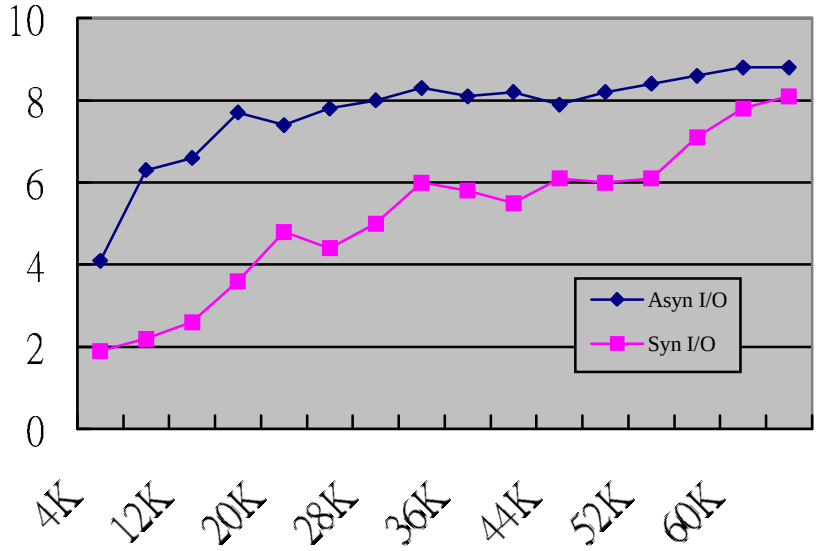


Figure3: Throughput comparison




commercial compiler is applied. Figure 2 lists the results. Compared to the numbers in Table 1, performance is better and almost achieves our expectation.

To answer what overhead dominates the performance bottleneck inside the commercial operating system is not a simple question. It requires a detailed component by component evaluations. But just by comparing with both systems' features, some guidelines can be concluded. First, I/O subsystem is also crucial to servicing multimedia applications, even though we though that software players will consume a lot of computing power, and should be almost dominated by processor capacity. This is not true as according to our observations. In the embedded kernel, the file system service is implemented in asynchronous I/O, and is implemented in the slow FAT16 architecture with only IDE disks. Asynchronous I/O in this system is something like a Streamed I/O. It means once a data stream is requested, because we know that the following data will be accessed in sequential, the I/O subsystem can fetch the data greedily to meet the requirements of MPEG players. The I/O services do not need to wait for any I/O requests for performing real I/O accessing operations. Of course, the buffer management in servicing this kind of service needs calibrate scheduling to avoid any data overflow problem.

To implement such an asynchronous I/O subsystem into the commercial operating system, again, is not an easy way. No source code exported from the vendor is the primary reason. To alleviate this problem, as according to our evaluations, using large block size in file system can help in reducing the performance gap. Figure 3 is the figure to show that once the disk block size is enlarged, asynchronous and synchronous I/O has almost the same throughput. Synchronous I/O, however, still has larger variance for each data accessing time.

Table2: The evaluated streams

| |
|---|
| Name : Under Siege, Resolutions：352 × 240<br>Frames/seconds：29.97<br>Frame order：IBBPBBPBBPBBPBBPBBI…<br>Average video frame size：4755.06 bytes<br>Average audio frame size：1008.01 bytes<br>Feature：Action movie with many people moving around and many scene changes. |
| Name : Love Letter , Resolutions：352 × 240<br>Frames/seconds：29.97 frames<br>Frame order：IBBPBBPBBPBBPBBI…<br>Average video frame size：4758.3 bytes<br>Average audio frame size： 992.9 bytes<br>Feature：A romantic story with very few moving objects and people, and almost with very few and slow scene changes. |
| Name : The Fifth Element, Resolutions：352 × 240<br>Frames/seconds：23.976<br>Frame order：IBBPBBPBBPBBPBBPBBI…<br>Average video frame size：6005.5 bytes<br>Average audio frame size：1261.3 bytes<br>Feature：This is a scientific story with common amount of people moving within the movie. There are frequently scene changes but the moving speed for scene is slow. It also has a lot of burst of colors, such as explosions, fire, etc. |

**Bottlenecks within the MPEG-II Player**

The second set of evaluation is to find which components are the major bottlenecks within our MPEG-II decoder. Though there was studies in similar evaluations[5], their evaluations are based on UNIX platform and MPEG-I decoders. We are eager to know the effects on commercial operating systems for MPEG-II decoders playing commercial and hot movies. The features and attributes of the selected movies are listed in Table 2. The processing time for each processing stage is listed in Table 3.

Table 3: Processing time distribution

| Functions | % Time |
|---|---|
| Display | 6.64% |
| Dither | 24.85% |
| IDCT | 27.67% |
| Stream Parsing | 19.55% |
| Reconstruction | 15.31% |
| MISC. | 5.97% |

Table 4:Evaluated if MMX is applied

| Functions | % Time |
|---|---|
| Display | 21.05% |
| Dither | 12.53% |
| IDCT | 8.77% |
| Stream Parsing | 31.44% |
| Reconstruction | 19.34% |
| MISC. | 6.89% |

The evaluated results when MMX technology is applied are listed in Table 4.

The results showed on Table 3 are different with previous studies. The reconstruction is much smaller than previous reported numbers, while the IDCT is large than what evaluated under UNIX. This leaves us a lesson that performance tuning should be done system by system. Different systems will have different problems and bottlenecks. Again this result showed the impacts by limited memory bandwidth and cache hit ratios are not so evident on this commercial operating system. This causes the MPEG-II decoder again being a computation-bound application.

The interested thing is that if the decoder is enhanced by MMX instruction sets, the new performance bottlenecks are the display and the stream parsing. The display part is not what we concerned since we did not use the default libraries in the commercial environment. This part can be improved by invoking system libraries to put pictures directly into the video frame buffers. The stream parsing, however, is very hard to be parallel processed. Most of the operations are bit-wise operations, which are time consuming. Each ship, test operation need several machine cycles and will cause more

CPU to memory traffic if the processed data is large. The results can supply enough information that can contribute the directions in processor design. The bit operations are not noticed in processor design but will become bottlenecks for current multimedia applications.

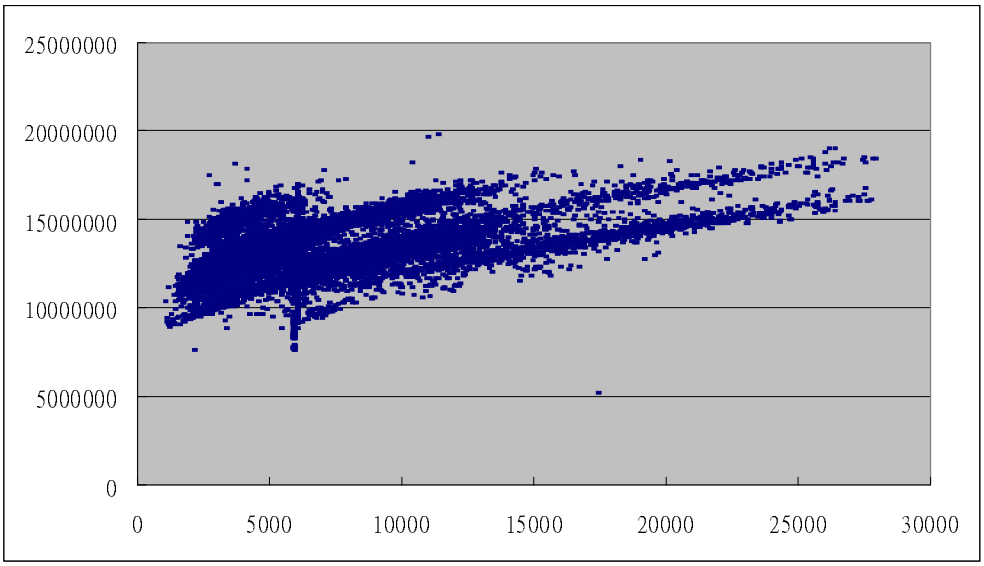
**Frame Size and Processing Time**

One important thing in exactly controlling the video player to support high quality video playback is to play each frame on time. Some researches focus on employing the real time kernel functions[12]. However, how to exactly set the timing parameters to the kernel is always missing. Since each frame's processing time is not always the same. To know each frame's processing a prior is important for software decoders.

A direct assumption is whether the frame size will determine the processing time. If so, it seems a way to decide the processing time when each frame is processed. We select a hot movie, THE FIFTH ELEMENT, for our experiment, which can supplies enough frames in statistic calculations. 32000 frames are recorded

Figure 4 illustrates the relationships between frame size and processing cycles. The I, B, P frames are plotted together for convenience. However, for same frame type, we found that most frame processing cycles are proportional to the frame size. This matches the descriptions in statistic *simple linear regression line model*[14] in relating two independent variables. The coefficients of determination are about 0.9725 0.9611 and 0.9384 for $r^2_I$, $r^2_P$, and $r^2_B$ respectively. This is an interesting result

Figure 4: processing cycles and frame sizes



because the coefficients are so close to 1, which means the frame size has strong relationship to processing time. This information can be used to inform the real time kernel of determining the timing attributes. Unfortunately, the commercial operating is not a real time kernel that we can not make use of these results directly. Whether other multimedia applications will have similar results is an interested problem to us.

In addition, there still remain problems waiting for solutions. The first is how to do dynamic prediction since the results we get from Figure 4 are via static analysis, while in run time we can not get all the information at once. Besides, how to make resource reservations and do timing attributes passing are still challenging.

**System clock granularity**

One important thing in system controls and evaluations is how to get the exact time of interested. Most systems use 10 to 20 mseconds time slot in controllling the whole system. The clock generator will generate clock interrupts to the CPU scheduler to trigger the scheduling operations. However, if the time slot granularity is on the order in 10 mseconds, the granularity is too rough and too large for multimedia applications. For example, MPEG-1 asks the playback quality in 30 frames per second. It means each frame needs about 33.3 mseconds in processing the data. If the system has only time granularity in 10 mseconds, the errors is 3 mseconds, which is also a 30% error. In addition, most system calls of time functions, such as sleep(), gettime(), settime(), etc., use 10 mseconds as unit. The reported data , thus, induce too large errors in values.

On the other hand, if we force the system to generate system events in finer time granularity, the system overhead will be too high to influence the whole system's behavior. For example, if one scheduling decision and event generation will consume about 0.1 mseconds, if we let the system clock time granularity be 1 msecond, the overhead is about 10%. This means, the whole system will spend 10% computing power in handling useless system affairs. To solve this problem, we adapt a two-level scheduling mechanism. We still use 10 mseconds as the base time granularity, but implement a new function to get the machine cycle count. Only several instructions are used to fetch the machine cycles. The induced cost is low. The only effort is that we need to know the clock rates first, and then we can calculate the total time after. The program is short. Appendix lists the program.

**Data alignment**

During going the experiments, another idea motivates us to explore more insights into the memory bandwidth's impacts[13]. If memory has different bandwidth for different physical addresses, then

whether data alignment will influence the total system performance is another interested issue. In fact, in our implementations, no matter in C or assembly languages, we all access data in alignment. But if mindless engineers ignore data alignment, the performance will degrade?

So, an experiment is designed to explore the performance differences. This experiment is partitioned into two parts. The first is to write a program in assembly language and access the memory sequentially. There are two ways in accessing. One is to access the memory in physical address order, the other is to jump several address slots, but still sequentially access the memory. The second is to do matrix multiplication. The data are put into the buffer sequentially. Another way to put data into the buffer is to let the matrix stored in column-majored, thus accessing to adjacent elements of one matrix will access to elsewhere addresses.

The results showed the impacts are large. Whether data is aligned or not will cause performance to degrade to 43% to 29%. Jumping-address accesses will cause performance to degrade more, even to 31% to 19%. Due to non-aligned data, the performance is degraded because of the physical memory's design. It touches the bus architecture, the memory bank architecture, the cache speed, cache hit ratio and the CPU internal and external clock cycles. Jumping-addresses accesses will degrade performance because of the high cache miss rates. Since accesses are not on the same memory bank, this also degrades the performance. Fortunately, commercial compilers almost handle data alignment problem automatically. Thus we do not need to handle this problem. But for the programmers that wish to use assembly languages to speed up the program execution and reduce the code sizes, the data alignment and access locality problems should be considered. For those who work for embedded systems, they should be careful in designing their compiling tools.

## 4. AN ADAPTIVE QoS FRAMEWORK

For dynamic adjusting the MPEG-II player to meet system loads, an adaptive QoS framework is also implemented in the current implementations. Our concept is based on the fact that human eyes are more sensitive to jitters-like variations than on the number of colors played on the screen. When a frame-dropped rate is higher than a default, the MPEG-II decoder will automatically output the frame images in less number of colors. There are four modes, as listed in Table 5. Three hot movies are used as input to the players. They are THE FIFTH ELEMENT, LOVER LETTER and the UNDER SIEGE. Only the second is a romantic story while the others are action movies.

**In GOP alignment to change QoS in colors**

The first result we found is that the best boundary for changing the color-bits is on GP boundary. This is due to the fact that B and P frames will reference other type frames. If the color bits are different, it will be very difficult in referencing other frames. Most GOP frames will reference each

Table 5: The QoS levels

| Mode | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Video QoS | RGB8 | RGB8 | RGB16 | RGB24 |
| Audio Qos | Half Samples | Normal | Normal | Normal |

Table 6: CPU utilization in each evaluation

| | Mode 1 | Mode 2 | Mode 3 | Mode 4 |
|---|---|---|---|---|
| Under Siege | 62.5% | 67% | 82.3% | 93.9% |
| Love Letter | 59.8% | 62.2% | 81% | 36.81% |
| The Fifth Element | 56.8% | 1.11% | 71.6% | 93.1% |

Table 7: Drop rates in each mode level

| | Mode 1 | Mode 2 | Mode 3 | Mode 4 |
|---|---|---|---|---|
| Under Siege | 0.35% | 0.54% | 4.71% | 36.81% |
| Love Letter | 0.82% | 1.11% | 7.23% | 35.6% |
| The Fifth Element | 0.018% | 0.085% | 0.33% | 22.7% |

other often, but outside the GOP, the references are not so frequent. This is why we suggest that to let GOP boundary be the time to change number of color-bits. This will reduce a lot of efforts in processing the difference of color-bits between different frames.

**Human is not so sensitive to these 4-mode plays**

In our experiments, we found that most of us do not notice the color changes for these MPEG-1 streams. It means we can spare the computing power to support higher QoS in other directions just by reducing the number of colors. For example, to increase the resolution will let human feel better in QoS because too small movie is not easy to view for general people. Another example is to provide better audio quality.

**Spared computing power is large**

The drop rates for each mode increase largely to the mode levels, and so do the processor utilization rates. Since human are not so sensitive to number of colors, to reduce the number of colors to spare processor power is a good way to provide different but better QoS to human. For example, a system can reduce its color to receive a video mail when he watches a movie. Software decoder has such good advantages to adapt itself to provide this kind of services. Since reduce color from 24 bits to 8 bits will reduce 4 times the data size, it means it can handle 4 movies at the same time.

MPEG-II also has already provided the Scalability Techniques to provide variable QoS playbacks. It defines 4 levels in scalability: *data partitioning scalable*, *signal to noise ratio scalable*, *spatial scalable* and *temporal scalable*. Its concept is to partition the data into different levels. If decoder is powerful to play the highest quality, it can decode all levels of the data. Otherwise, only base levels of data are processed. So far, we do not find any accessible MPEG-II data streams coded in these formats. We are interested to evaluate this technique in the future. Our approach presented here is just to highlight that reduction in colors can spare a lot of computing power, which can be used to provide other type of QoS.

## 5. CONCLUSIONS

We have addressed our work, experiences and results in implementing an MPEG-II player on a commercial operating system in this report. As addressed in the report, we summary important results into following.

The first is about the MMX's impacts to real player performance. We examine its influence to the

player in playing MPEG-II stream data. The second is about the external influence to the player, the operating system behaviors. Unless the operating system behavior is well controlled, it is no way to well control the player's behavior and performance. System impacts induce far larger influence to player's executions. Multimedia applications need new environments and system resource management schemes in providing better multimedia services. Third, in our evaluations, the frame size and processing time has a proportional relationship in running the MPEG-II video decoder. We so far do not know whether other multimedia applications have the same features. If so, this can be used to predict the processing time and can cause systems to provide better multimedia supports for concurrently running multiple applications. Finally, if a processor is overloaded, the player can reduce the colors to release the loads. The color-bit is not so important than jitters or media synchronous skews.

Based on above results and current work, instead of using commercial operating systems, our future work is to integrate the player into our new embedded kernel. We are interested at several topics. The first one is whether other multimedia applications, for example, video conferencing, still own the property that data processing time is proportional to the data size? The second thing is how to enhance this player to play remote data smoothly. This involves transportation between Internet, which has higher complexity than controlling a standalone platform. How to do media synchronization within the decoder is also another challenging problem. We already have a good starting point now.

## REFERENCES


[1] J. Mitchell, W. Pennebaker, C. Fogg and D. LeGall. *MPEG Video Compression Standard*, Chapman and Hall, 1996.

[2] Soung C. Liew and Derek Chi-yin Tse, "A Control-Theoretic Approach to Adapting VBR Compressed Video for Transport Over a CBR Communications Channel", In *IEEE/ACM Transactions on Networking*, Vol. 6, No. 1, February 1998.

[3] Joseph Chou, "Real Time MPEG-I and MPEG-II Compression with the Alpha Microprocessors", In *Proceedings of the 1997 Multi-Tech Conference*, p.p. 42-53, Taipei, Taiwan, December 1997.

[4] Alex Peleg at al. "Intel MMX for Multimedia PCs", *Communications of the ACM*, Vol. 40, No. 1, p.p. 25-38, January 1997.

[5] Ketan Patel, Brian C. Smith and Lawrence A. Rowe. "Performance of a Software MPEG Video Decoder", In *Proceedings of the 1993 ACM Multi Media Conference*, 1993.

[6] MPEG-2 Standard, ISO/IEC International Standard 13818.

[7] Davis Pan, A Tutorial on MPEG/Audio Compression, *IEEE Multimedia*, 1995, pp. 60-74.

[8] Helen Custer, *Inside Windows NT*, Microsoft Press, 1993.

[9] Peter Soderquist and Miriam Leeser, Optimizing the Data Cache Performance of a Software MPEG-2 Video Decoder, In *Proceedings of the ACM MultiMedia'97*, p.p. 291-301, November 1997.

[10] Chi-Wei Yang, Paul C. H. Lee and Ruei-Chuan Chang, Reuse Linux Device Drivers in Embedded Systems, In *proceedings of the 1998 International Computer Symposium (ICS'98),* Taipei, Taiwan, 1998.

[11] Paul C. H. Lee, Chi-Wei Yang and Ruei-Chuan Chang, An Integrated Core-Work for Fast Information-Appliance Buildup, IIS Technical Report, TR-98-006, Institute of Information Science, Academia Sinica, Taipei, Taiwan, 1998.

[12] Ralf Steinmetz. Analyzing the Multimedia Operating Systems. In *IEEE MultiMedia*, Vol. 2, No. 1, Spring 1995

[13] Sally A. McKee, Robert H. Klenke, Kenneth L. Wright, William A. Wulf and James H. Aylor, Smarter memory: Improving Bandwidth for Streamed References. *IEEE Computer*, July 1998, pp.54-63.

[14] Jay l. Devore, *Probability and Statistics for Engineering and the Sciences*, 2nd edition, Brooks/Cole Publishing Company, 1987.

[15] Herng-Yow Chen and Ja-Ling Wu, MultiSync: A Synchronization Model for Multimedia Systems. *IEEE Journal on Selected Areas in Communications*, Vol. 14, No. 1, January 1996, pp. 238-248.

# APPENDIX

### *Linear Regression Line Model*

There exist parameters $\beta_0$, $\beta_1$, and $\sigma^2$ such that for any fixed value of the independent variable *x*, the dependent variable is related to x through the model equation

$$Y = \beta_0 + \beta_1 + \varepsilon$$

The quantity $\varepsilon$ in the model equation is a random variable, assumed to be normally distributed with $E(\varepsilon) = 0$ and $V(\varepsilon) = \sigma^2$

### *Cofficient of Determination*

The coefficient of determination, denoted by $r^2$, is given by

$r^2 = 1 - SSE / SST$

It is interpreted as the proportion of observed *y* variation that can be explained by the simple linear regression model (attributes to an approximate linear relationship between *y* and *x*)

***Program to get system machine cycles***

```
unsigned long long mpeg2_get_time(void) {
    unsigned long long pc_value=0LL;
    register unsigned long eax asm("ax");
    register unsigned long edx asm("dx");
    __asm__(".byte 0x0f,0x31"
                        :"=a" (eax), "=d" (edx));
    pc_value = (unsigned long long)edx;
    pc_value = pc_value<<32;
    pc_value += (unsigned long long)eax;
return pc_value;
}
```