

Software Based MPEG-2 Encoding System with Scalable and Multithreaded Architecture

Ishfaq Ahmad, Dick-kwong Yeung, Weiguo Zheng, Shehzad Mehmood

Multimedia Technology Research Center
HKUST, Clear Water Bay, Hong Kong

Abstract¹

MPEG-2 video encoders are now available in a variety of forms using both hardware and software based approaches. The software-based approach potentially offers a better picture quality but is computationally quite intensive. MPEG-2 video encoding can be fast processed using parallelism. A number of approaches using parallel machines or networks of workstations have been reported. While these approaches promise good concepts they do not offer commercial solutions due to factors such as cost, size, etc. In this paper, we propose a new approach with the aim to build a cost-effective and a completely practical solution that is not only highly efficient but is also scalable from single-processor to multiple-processor PC. The highlights of the proposed work include an algorithm for enhancing the efficiency of motion estimation, speeding up the computation of motion estimation and DCT with Intel's SIMD (Single Instruction, Multiple Data) style MMX and SSE instruction sets within a single processor, and scheduling and allocation of a multithreading scheme on a multiple processor PC for managing I/O, synchronization, audio and video encoding, and multiplexing. The proposed multithreaded encoder exploits temporal parallelism in MPEG video sequences with small overhead. The encoder, providing a complete compression solution, achieves faster than the real-time and half of real-time encoding rates for CIF (352 x 288) and CCIR601 (720 x 576) video sequences, respectively, on multiple processor PC.

1. Introduction

Digital video sources need massive data rates. For example, uncompressed CCIR (ITU-R) 601 with resolution of 720 x 576 (PAL) pixels (16-bit with YUV 4:2:2 color space) has data rate of 166Mbps. Reducing the spatial and temporal redundancy of a video sequence is the main objective of video compression. After compression, MPEG-2 coding of the CCIR601 video sequence may require only 4 to 15Mbps with acceptable visual quality [1], [2], [3].

However, software based video encoding requires very high computational power, e.g. CCIR601 video encoding takes several Gigaflops if brute-force motion estimation is adopted. Consequently, software-based MPEG-2 video encoders can only achieve a speed of only few frames per second. One possibility to overcome this hurdle is to use parallel processing. Parallel approaches include the use of parallel supercomputers and networks of workstations. MPEG-2 encoder with a software implementation is desired to be scalable so that the encoding speed can adjust depending upon the number of available processors.

While some work [4], [5], [6], [7] has been reported with real-time encoding rates on parallel processors, their underlying assumption is the availability of raw video source on a disk system. These experiments have been carried out on parallel and distributed platforms, including network of SUN, SGI, HP workstations, Intel iPSC/860 hypercube and Intel Paragon, connected with a parallel file system (PFS) or network file system (NFS). In these experiments, I/O has been reported to be the main bottleneck. The basic philosophy in these works is to improve the overlap of video data distribution and the encoding task in order to prevent the encoder performance from being limited by the I/O performance.

In this paper, we exploit a different kind of parallelism and design a complete system for MPEG-2 video encoder on a multiprocessor platform with multithreaded operating system. We design efficient scheduling of multiple threads for managing I/O, compression of video, audio, and their multiplexing. In addition, we propose algorithms for optimization of motion estimation and parallelism within the single processor by exploiting MMX and SSE Instruction.

This paper is organized as follows: In Section 3, we describe the optimization of MPEG Software Simulation Group's (MSSG) MPEG-2 video encoder. In Section 2, we discuss the architecture of the multithreaded video encoding system and modules of the proposed system. Section 3 describes the multithreading and scheduling. Experimental results are presented in Section 4, and Section 5 concludes this paper.

2. Optimization of the Encoding Algorithms

The MPEG Software Simulation Group (MSSG), during the course of defining the standard, has developed MPEG reference software. The encoder can be used for

¹ This work was supported by Research Grants Council of Hong Kong under contract # CRC98/01. EG05, HKUST6228/99E, and HKTIIT98/99. EG02.

generating constant bit rate MPEG-2 video and is the first publicly available encoder based on the Test Model 5 (TM5) coding model. The architecture of the MPEG-2 encoder is depicted in Figure 1.

Our optimization consists of various techniques, such as B-picture reconstruction skipping, hierarchical motion estimation, fast DCT & IDCT, SIMD implementation, etc.

2.1 Skipping B-picture Reconstruction

In MPEG-2 encoder, for the purpose of the encoding quality evaluation, all encoded pictures are reconstructed after quantization. In practical applications, the PSNR calculation is unnecessary. So, we can avoid some computation for picture reconstruction.

Each video sequence is composed of a series of groups of pictures (GOP). Based on the referencing dependencies, frames that will be referenced are encoded first. In the decoding process, reference frames (*I* and *P* frames) will be decoded first and then referenced by other *P* or *B* frames before display. Due to the frame dependencies within each GOP, GOP structure is intended to provide random access into a sequence. Each GOP is an independently decodable unit as long as it begins with an *I*-frame. A GOP can be described as “open” or “closed”. In an open GOP, the last *B* frames of each GOP needs to reference to next GOP’s *I*-frame (Figure 2 is an open GOP). In a closed GOP, the last frame of each GOP is a *P* frame; frames inside each GOP do not reference to next GOP (for instance, *I, B, B, P, B, B, P, B, B, P, I, B, B, P...*).

B-pictures provide the maximum compression ratio since they can exploit the bi-directional prediction from the past and future pictures. But *B*-pictures do not server as reference pictures, and, therefore, there is no need to reconstruct *B*-pictures in the encoding loop.

2.2 Hierarchical Motion Estimation

Motion estimation (ME) is the most important part of the MPEG-2 encoder, since it reduces temporal redundancy from video sequences and significantly affects the output quality of the encoded sequence. This is also the most complex part of compression with an overwhelming computational complexity compared with other parts of the encoding process.

Using searching window size of ± 16 , motion estimation (full search) can consume more than 90% of processing resource. A myriad of algorithms are reported to improve the speed and performance of motion estimation, such as three-step search, new three-step search, 2-D logarithmic search, conjugate directional search and hierarchical search [8], etc.

First we design a fast hierarchical motion estimation algorithm. For simplicity, the pyramidal pictures is obtained by averaging:

$$p_l(i, j) = \left[\frac{1}{4} \sum_{p=0}^1 \sum_{q=0}^1 p_{l-1}(2i+p, 2j+q) \right] \quad (1)$$

$$1 < l < 4.$$

Where $p_l(i, j)$ represents the gray level at the position (i, j) of the l -th level and $p_0(i, j)$ represents gray level of original picture.

Using Equation 1, we can construct $L+1$ layer pyramid. Layer 0 denotes the original picture to be encoded. If $L=2$, we have 3 layers pyramid, and the size of top layer is 1/16 of the original picture. And each layer is quarter size of its lower layer. On the highest layer, we perform full search to find the predictive motion vectors, and refine the motion vector on the lower layers.

The hierarchical search algorithm is described as follows:

Step 1. Construct the pyramid with $L=2$.

Step 2. The top layer is mapped into multiple 8x8 non-overlapped blocks. Each block represents a 16x16 block in middle layer. Full search is executed on the top layer with search range (R_L, R_L) , where $R_L = R/4$, R is the defined search range for original picture size.

Step 3. We separate the middle layer into 16x16 non-overlapped blocks. Each block represents 4 16x16 macroblocks in the bottom layer. The predictive motion vector from the top layer is refined in middle with search range $[-1, 1]$. The new delta data are obtained for X and Y coordinates.

Step 4. Refinements are repeated in bottom layer with search range $[-1, 1]$. The final vector is obtained by adding the delta data to predictive motion vector.

2.3 Optimization of DCT & IDCT

The DCT function reduces spatial redundancy in video and image data. It provides the basis for compression on the 8x8 pixels block by decomposing pixels value into a weighted sum of spatial frequencies. IDCT is the inverse of DCT but has the similar structure. The 8x8 two-dimensional DCT and IDCT used in MPEG compression are defined in the following equations.

Forward 8x8 2D DCT:

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos \frac{(2x+1)\pi u}{16} \cos \frac{(2y+1)\pi v}{16} \quad (2)$$

Inverse 8x8 2D DCT (IDCT):

$$f(x, y) = \sum_{u=0}^7 \sum_{v=0}^7 \alpha(u)\alpha(v) C(u, v) \cos \frac{(2x+1)\pi u}{16} \cos \frac{(2y+1)\pi v}{16} \quad (3)$$

where $\alpha(k) = 1$, if and only if $k \neq 0$, otherwise,

$$\alpha(k) = \frac{1}{2\sqrt{2}}.$$

By these equations, the DCT of one 8x8 elements block takes 4096 multiplications and 4032 additions. Because of the large number of operations on transforming single block, different algorithms have been proposed for efficient calculation of the 2D DCT. Most fast 1D DCT and IDCT algorithms are variants of Lee's Fast DCT algorithm, or are based on variants of Winograd's FFT [9].

We have adopted a fast 8x8 DCT and IDCT function using the SIMD instructions (see next section below) based on the AAN algorithm [9].

2.4 SIMD Implementations

SIMD instructions for Intel processor architectures IA-32 and IA-64, known as MultiMedia eXtension (MMX) and Streaming SIMD Extensions (SSE), accelerate applications that rely heavily on operations using floating-point data (such as 3D graphics, real-time physics, and spatial audio). The principle data is the packed, fixed point integer or byte, where multiple data (byte, word, dword) can be grouped into a single 64-bit quantity. These 64-bit quantities are stored in a 64-bit SIMD register and processed by a single instruction in a data parallel fashion. Therefore, the computational performance of the processor is enhanced [10], [11]. However, the speedup is restricted by the overhead data alignment, data copying and data type convention.

We exploit SSE instruction for fast computation of the block matching part in motion estimation. Block matching is implemented by using *psadbw* instruction. Two *psadbw* instructions calculate the SAD (sum of Absolute Differences) between the pixels in one row of the reference and current macroblock. These two SADs are summed to produce a 16-bit (word) result. Normally, it takes about 20 instructions for calculating absolute differences of 16 pixels. But with SSE, only two *psadbw* instructions are required.

Similar to motion estimation, we use MMX and SSE for the calculation of DCT by processing four 16-bit data elements in parallel.

3. Multithreaded the MPEG-2 Encoding System

A thread is a piece of code within an application that runs concurrently with the application's other threads, sharing an address space with them, along with access to the application's variables, file handles, device contexts, objects, and other resources. Threads are different from processes, which typically do not share resources or an address space and they communicate only through the mechanisms provided by the operating system for inter-process communication, such as pipes and queues. Threads often use simpler and less resource-intensive forms of communication like semaphores, mutexes, and events.

Threads can improve the responsiveness, structure, and efficiency of the program code. In addition, some programs containing concurrent threads may run significantly faster on parallel computers under multiprocessor operating systems since each thread could make full use of its own respective CPU.

In using multithreading to implement parallelism, the overhead caused by thread creation and thread synchronization can counteract the benefits of parallelism. Creation of a thread is equivalent to 45000 single precision floating-point divides, so it is a common practice to create thread at program startup and keep on using the created thread [12]. Because threads require synchronization mechanisms to guard against race conditions in shared data, the volume of processing data block on each thread can be a major factor in determining whether a process is suitable for multithread processing. It is important to minimize the effect of synchronization overhead by processing larger data blocks in each thread [12], [13], [14]. In order to improve the encoding speed and to minimize the overhead mentioned before, the number of threads, the amount of data processed by each thread, management of shared data I/O and the concatenation of results mechanism should be determined properly. In addition, threads should be properly scheduled

In terms of data distribution, GOP level temporal parallelism [4] is the coarsest data distribution in MPEG-2 encoding. In order to minimize the I/O overhead and simplify the initial implementation, we choose closed GOP (with pattern like *IBBPBB...P*) as the data block unit (see Figure 2). Due to the nature of frame dependencies, no penalty frames exist because the frames inside each GOP do not make reference to the previous GOP.

Due to the bandwidth limitations of disks and networks, I/O bottlenecks are a common problem in MPEG-2 encoding. Previous parallel encoding approaches used an architecture connecting the encoding nodes through the network and parallel file system. In our approach, we use a RAID 0 disk driver connected with the encoder machine for raw video data storage. The access rate of this RAID 0 disk is about 260Mbps, which is higher than the bandwidth requirement of CCIR601 raw video data (166Mbps).

To overlap computation and I/O, we create a raw video input thread for reading raw video data into memory, operating concurrently with the encoding process. Double buffering with round robin scheduling is used for reducing the I/O wait time. As shown in Figure 3, a double buffer is pre-loaded with uncompressed video data before each encoding process.

The concatenator combines the encoded GOPs into a single MPEG-2 stream. As each of the encoded GOPs can

be considered as one independent MPEG-2 stream, operations for the concatenator are read as unordered encoded GOPs from different multithread encoders, refilling information (such as `vbv_delay`) of each frame inside the GOP and writing the single MPEG-2 stream in the right order. At the same time, the concatenator acts as a synchronizer for controlling the encoding of the next batch of GOPs.

Our MPEG-2 encoding system also includes the audio encoder and multiplexer for generating audio-visual MPEG-2 stream. The audio encoder and multiplexer are also initialized as threads. The master process creates tasks for communication of different modules in the system and schedules the startup of I/O thread, multithreaded video encoder, audio encoder and multiplexer

For the audio part, we have used the MPEG Audio Subgroup Software Simulation Group's MPEG-1 audio encoder. The audio encoder compresses the raw audio signal into a MPEG-1 audio layer-2 stream. The MPEG-2 System Multiplexer follows the ISO 13818-1 system syntax [1]. We have developed a multiplexer, which multiplexes the encoded audio and video data into a single file. As described in previous section, the multiplexer also acts as a synchronizer, after each batch of encoding task, N GOPs from N threads are assembled by the multiplexer. Similar to the audio encoder, the multiplexer incurs a very little computation on the overall system.

Other than the master process and multithreaded video encoder, three threads for execution of raw video pre-loading, audio encoder and multiplexer are created for the encoding system. Even the workloads of these three extra threads are far lower than the video encoder, thread switching and data sharing may cause huge overhead or resource deadlock with inappropriate thread scheduling.

In order to occupy the CPU time efficiently, multithreaded video encoders work exclusively. After encoding each batch of GOPs, raw video pre-loading thread, audio encoder and multiplexer grab all of the CPU time and work concurrently. Figure 4 illustrates the scheduling of thread execution. In the very beginning, audio signal corresponding to first N GOPs is encoded and first N GOPs are streamed to the buffer by the raw video pre-loading thread. Then multithreaded video encoders start encoding video frames. Next, encoded video and audio are multiplexed. At the same time, the next batch of GOPs is streamed to the buffer and the corresponding audio signal is encoded, and so on.

The thread scheduling that we designed minimizes waiting time of required data for operation and executes different sequential task (multiplexing, audio encoding and raw video data loading) in overlapping manner. This scheme can also work on single-processor machine as efficiently, because operations in the other threads can

occupy CPU time when disk I/O operations of audio encoder or raw video pre-loading thread are in the wait state.

4. Experimental Results

We performed tests on a machine with four Intel Pentium III Xeon 550 MHz processors. In order to deal with the high data rate for raw video pre-loading (166Mbps for CCIR601), a RAID disk system is connected to the machine. We used five video sequences: a live concert, three action movies and a variety show. The original resolution of all these sequences is CCIR601 (720 x 576) at 25 fps (PAL TV mode). The same sequences were also down-sampled to CIF (352 x 288) for separate encoding.

The tests were done using one to four video encoding threads. The CCIR601 sequences are compressed to a 4Mbps constant bit-rate bit stream and closed GOP with a size of 13 frames. Audio data is compressed to 160kbps and audio-visual stream is multiplexed to 4.5Mbps. For CIF, the bit stream is compressed to 1.7Mbps and multiplexed to 2Mbps with the same audio bitrate.

Table 1 and 2 show the encoding speeds (frames per second) of CCIR601 and CIF sequences with various numbers of video encoding threads on the 4-processor machine, respectively. For CCIR601 sequences, two and three video encoding threads indicate a linearly increasing speedup for the encoding rate, with an average of 1.9 and 2.9, respectively. However the speedup with four threads incurs some drop. This is because the overhead caused by thread switching in two or three video encoding threads is lower than the one with four threads. In two or three video encoding threads, the loads of the encoders tasks can be distributed to three processors while one processor is assigned for other threads (audio encoder, multiplexer and raw video pre-loading thread) and OS task. For four video encoding threads, the overhead increases due to thread switching.

For CIF sequences, the average speedup with two, three and four video encoding threads is nearly the same. That is because the bandwidth of the disk array is 260Mbps, which is equivalent to the size of about 39 CCIR601 frames. In other words, the highest access rate for both CCIR601 and CIF video sequence is about 39fps.

5. Conclusions

In this paper, we proposed a parallel MPEG-2 encoding system with scalable and multithreaded architecture. We discussed strategies for parallelization and data distribution. With the aid of SIMD instructions and various other optimization techniques, different modules inside the video encoder are optimized in order to achieve faster encoding rate. The multithreading scheme is scalable in that it generates and schedules the number of threads according to the number of processors. With proper scheduling, different modules of MPEG-2

encoding system such as audio encoder and multiplexer (which require less computational time than the video encoder) are grouped together. The parallelization strategy yields encouraging improvements in speedup for the encoding rate. The experimental results show that encoding rate for the CIF format video is about 40fps, which is faster than real-time. For CCIR601, about 14fps encoding rate is achieved, which can further increase if more processors are available.

References

- [1] ISO/IEC, "Information Technology - Generic Coding of Moving Pictures and Associated Audio: Systems," *Draft International Standard 13818-1*, November 1993.
- [2] ISO/IEC, "Information Technology - Generic Coding of Moving Pictures and Associated Audio: Video," *Draft International Standard 13818-2*, November 1993.
- [3] ISO/IEC, "Information Technology - Generic Coding of Moving Pictures and Associated Audio: Audio," *Draft International Standard 13818-3*, November 1993.
- [4] S. M. Akramullah, I. Ahmad and M. L. Liou, "A Data-Parallel Approach for Real-Time MPEG-2 Video Encoding," *Journal of Parallel and Distributed Computing*, Vol.30, No.2, November 1995, pp. 129-146.
- [5] T. Olivares, F.J. Quiles, P. Cuenca, L. Orozco-Barbosa, I. Ahmad, "Study of Data Distribution Techniques for The Implementation of an MPEG-2 Video Encoder," *Parallel and Distributed Computing Systems '99, Proceedings of the Eleventh IASTED International Conference*, November 3-6, 1999, pp. 537-542. MIT, Cambridge, Massachusetts (USA).
- [6] S. M. Akramullah, I. Ahmad and M. L. Liou, "Performance of a Software-Based MPEG-2 Video Encoder on Parallel and Distributed Systems," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.7, No.4, August 1997, pp. 687-695.
- [7] K. L. Gong and L. A. Rowe, "Parallel MPEG-1 Video Encoding," *Picture Coding Symposium*, California, September 1994.
- [8] R.Li, B.Zeng, and M.L.Liou, "A New Three-Step Search Algorithm for Fast Motion Estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, Vol.4, pp. 438-442, Aug. 1994.
- [9] A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX Instructions, *Intel Application Note AP-922*, Order No: 742474-001.
- [10] Using Streaming SIMD Extensions in a Fast DCT Algorithm for MPEG Encoding, *Intel Application Note AP-817*, Order No: 243651-002.
- [11] Using Streaming SIMD Extensions in a Motion Estimation Algorithm for MPEG Encoding, *Intel Application Note AP-818*, Order No: 243652-002.
- [12] Coarse-Grain Multithreading, *Intel Application Note AP-802*, Order No: 243636-002.
- [13] Efficient Multithreading on Windows NT, *Intel Developer Note*, WEB: <http://developer.intel.com/>.
- [14] Multi-Threading: Taking Advantage of Intel Architecture Multiprocessor Workstations, *Intel Developer Note*, WEB: <http://developer.intel.com/>.

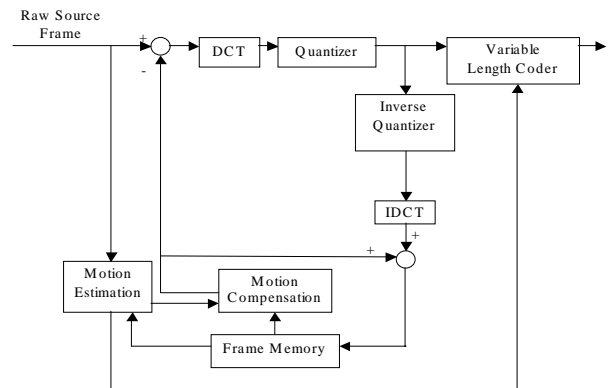


Fig.1: MPEG-2 video encoding architecture.

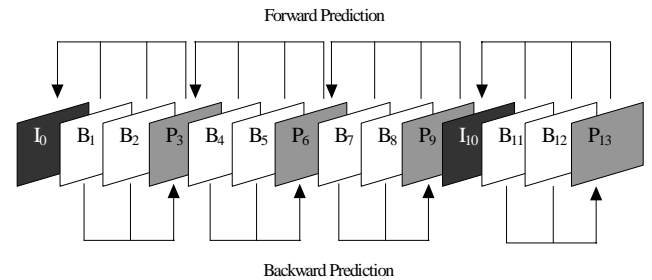


Fig.2: Group of pictures (GOP).

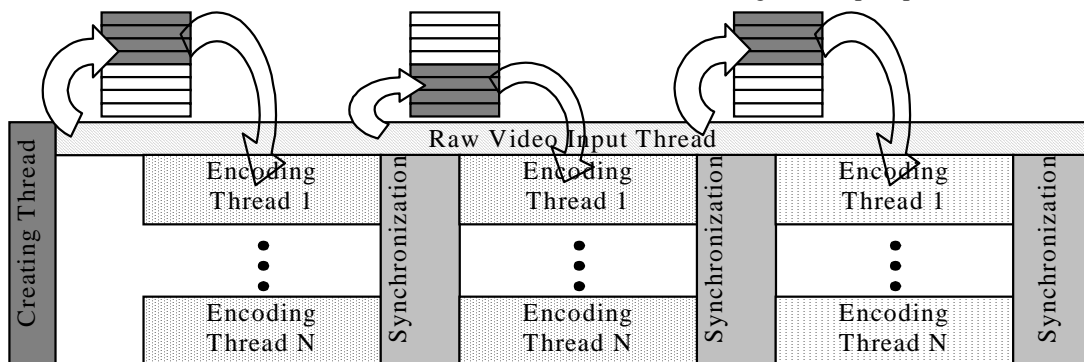


Fig.3: Video input thread with double buffer.

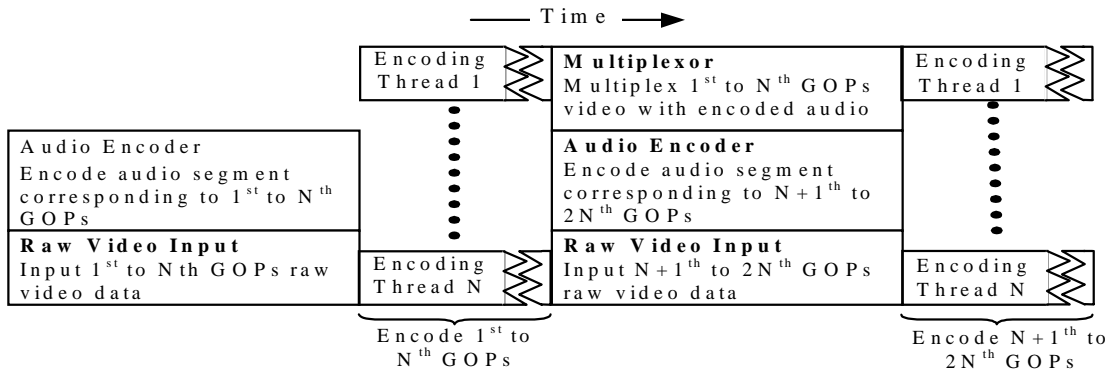


Fig. 4: Scheduling of thread execution.

Table 1: CCIR601 (720x576) frame encoding rate on four-processor machine.

Sequence \ Number of threads	1	2	3	4
Live Concert	4.526	8.816	12.962	13.461
Action Movie 1	4.366	8.523	12.561	14.235
Action Movie 2	4.690	9.132	13.346	14.816
Action Movie 3	4.783	9.292	13.566	15.288
Variety Show	4.895	9.484	13.983	15.564

Table 2: CIF (352x288) frame encoding rate on four-processor machine.

Sequence \ Number of threads	1	2	3	4
Live Concert	24.67	37.625	38	38
Action Movie 1	24.556	40.09	38.379	39.986
Action Movie 2	26.238	36.987	40.07	38.348
Action Movie 3	26.280	40.355	41.406	40.253
Variety Show	26.703	37.308	37.621	40.157