PARALLELIZATION OF MPEG-2 VIDEO ENCODER FOR PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS

Shahriar M. Akramullah[†], Ishfaq Ahmad[‡], Ming L. Liou[†]
[†]*Department of EEE, HKUST*, [‡]*Department of Computer Science, HKUST*
Clear Water Bay, Kowloon, Hongkong

Abstract—**In this paper, a parallel implementation of the MPEG-2 video encoder on various parallel and distributed platforms is presented. We use a data-parallel approach and exploit parallelism within each frame, which makes our encoder suitable for real-time applications where the complete video sequence may not be present on the disk. The *Express* environment is employed as the underlying message-passing system making our encoder portable across a wide range of parallel and distributed architectures. The encoder also provides control over various parameters such as number of processors, size of search window, buffer management and bit-rate. It is flexible and allows inclusion of fast and new algorithms for different stages of the codec, replacing current algorithms. Comparisons of execution times, speedups as well as frame encoding rates using different number of processors are provided. In addition, our study reveals the degrees of parallelism and bottlenecks in various computational modules of MPEG-2.**

1 Introduction

A VIDEO codec is comprised of an encoder and a decoder, which respectively performs compression and decompression of video data. As a video consists of a huge amount of data, these operations require a great deal of processing in the order of billion operations/sec. Since video encoding, specially in software, is much more complex and time-consuming for real-time applications compared to decoding, it is always advantageous to speed up the computation. This paper presents a software parallel implementation of a video codec (MPEG-2).

MPEG-2 embodies different modules some of which are very computation intensive. It is a *generic* standard designed to support variety of applications, several bit-rates of 2 Mbps and up, and various qualities and services. The encoder requires extensive computation to fully support applications such as HDTV, video-on-demand (VOD), video communications on ATM networks etc.

In [7] a SIMD implementation of the H.261 has been reported with a frame rate of about 5 fps (frames/sec). Parallel MPEG-1 video encoding with a performance of about 4 fps has been documented in [4]. It was modified [10] to run on the Intel Touchstone Delta and the Intel Paragon. Although faster than real-time performance has been claimed in [10], the drawbacks are crucial. For instance, the complete video sequence should be available before encoding begins. Also, usable number of processors to encode video of given length is limited, which restricts the scalability of the problem. Furthermore, it has used some special I/O capability offered by the Delta or the Paragon for improved performance, and therefore is not portable to other hardware platforms, e.g. a network of workstations. There have been some other approaches to parallelize codec operations of video sequences [1], [11], [12]. However, they need to use specialized hardware.

For our implementation, we have chosen the data-parallel approach. Our implementation does not employ any special-purpose hardware or programming primitives, rather it is completely portable, flexible and scalable. The implementation is performed on the Intel iPSC/860 and various types of networks of workstations.

The rest of the paper is arranged as follows. Section 2 describes the parallelization methodology and discusses data distribution and communication strategies. Section 3 provides experimental results. The last section concludes the paper.

2 Parallelization Methodology

The parallel implementation of MPEG-2 video encoder has been carried out using a *data-parallel* or *single-program multiple-data* (SPMD) programming paradigm. The SPMD paradigm under *Express* [9] allows our software to be portable across a wide range of architectures. In order to make our parallel implementation scalable, we assume that our target processor topology is a 2-D grid. This has been achieved using *Express*'s *Cubix* programming model which, in addition to providing overlapped data reading capability, can setup a virtual processor grid regardless of the hardware topology and then automatically map the data onto this array of processors. This allows us to control the granularity of the problem by enabling it to run on a few workstations in a coarse-grained fashion as well as on massively parallel systems in a fine-grained fashion.

The frame data is distributed among the processors, each processor having some 8×8 blocks of data, depending upon the number of processors available. Motion estimation is performed on independent macroblocks (16×16 block of pixels, abbreviated MB) while other operations used 8×8 blocks as the basic unit of parallel processing, unlike some approaches which use *slice* as the basic unit.

2.1 Data Distribution

Overhead due to interprocessor communication can be the major limiting factor for any parallel application. Therefore, partitioning the data among the processors should be such that minimal interprocessor communication is employed. In the current implementation, the whole frame is distributed as evenly as possible to each processor. It is also possible to partition the data by just apportioning the requisite part of the frame data (one or more 16 $\times$ 16 macroblock) to the corresponding processors as the processors are mapped onto the 2-D grid (see Figure 1(a)). But in that case, it necessitates essential communication between processors as the search window moves to the boundary during motion estimation .

Since each processor has enough memory to store the entire search window, it is possible to eliminate use of overwhelming amount of communication. In this case, the frame data is distributed among the processors allowing overlap (Figure 1(b)). Here, each processor is allocated some redundant data, which is necessary to form the complete search area.

Let us consider P be the height and Q be the width of the frame respectively, and let p be the total number of processors to be used, with p_h and p_v as the number of processors in the horizontal and vertical dimension respectively. Thus, $p = p_h \times p_v$. If the search window size is the size of the MBs in a particular processor $\pm W$ in both dimensions, with overlapped (redundant) data distribution, given p_h and p_v, one can determine the size of the local frame in each processor, which is given by

$$X_{local} = \left\lceil \frac{Q}{p_h} + 2W \right\rceil \times \left\lceil \frac{P}{p_v} + 2W \right\rceil \qquad (1)$$

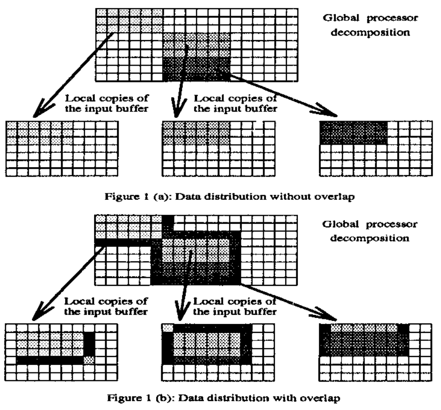
Global processor decomposition
Local copies of the input buffer
Local copies of the input buffer
Figure 1 (a): Data distribution without overlap
Global processor decomposition
Local copies of the input buffer
Local copies of the input buffer
Figure 1 (b): Data distribution with overlap

Fig. 1. Data distribution with and without overlap.

In our implementation, the number of processors to be used is an input parameter. Therefore, it can be ported to environments with a few powerful processors to those with a large number of relatively slow processors as well as to hardware platforms with limited memory or slow communication.

2.2 Implementation Features

Our implementation of the MPEG-2 encoder generates constant bit-rate streams and supports *progressive* as well as *interlaced* video. It can also generate MPEG-1 sequences and can support up to three input formats: separate YUV, combined YUV, and PPM. It outputs the encoded sequence as well as relevant statistics and verifies legality of the user-given parameters within profile and level. The current implementation does not support variable bit rate encoding, scalable extensions, integer pel motion vectors for MPEG-1 (always produces half-pel motion vector, which however, gives better quality), low-delay, concealment motion vectors, editing of encoded video and scene change rate control. Our parallel implementation is based on a sequential MPEG-2 implementation [8].

2.3 Motion Estimation

Block Matching Algorithm (BMA) is the adopted motion estimation technique in MPEG. It finds the best match for a pixel-block (e.g., 16 $\times$ 16) belonging to the current frame, within a user-defined search area in the previous frame. Since this employs a search for the best-matching block, a huge computation is involved, which leads to the motivation of parallel processing.

As the matching criterion we have chosen *Mean Absolute Difference* (MAD), while the search range remains as an input parameter. We have employed both exhaustive and fast search (2-D log-search [6]) patterns. The motion estimation is performed only on the luminance samples. The chrominance displacement is approximated by halving the luminance displacement. In order to further improve prediction accuracy, after doing an integral full-pel search, a half-pel search is also done on a neighborhood of eight bilinearly interpolated luminance samples from the reconstructed reference frame.

2.4 DCT and IDCT

Discrete Cosine Transform (DCT) is used for spatial redundancy reduction. In our implementation with full-search motion estimation, for DCT, standard *row-column approach* is used, while for IDCT, *Wang's algorithm* [13] is used. With log-search for motion estimation, both DCT and IDCT have used *Wang's algorithm* with double precision. The DCT and IDCT are performed on the 8 $\times$ 8 pixel blocks. The same serial program is executed on each processor to compute DCT or IDCT for as many blocks belonging to its local share of the frame data. So there is no interprocessor data movement.

2.5 Rate Control and Adaptive Quantization

Our implementation holds fast a single pass coding viewpoint and does not use any *a priori* measurement to guide the allocation of bits at the global layers. The complex bit allocation process is splitted into a number of independent stages, coincident with the various layers of MPEG-2 video. At the highest stage, alike [3], a *group of pictures* (GOP) becomes the edge where variable size coded pictures are mapped into a constant channel rate. The allocation of target bit for the current picture being encoded is based on a global bit budget for the GOP, and a ratio of weighted relative coding complexities of the three picture types (I, P, B). Coding complexity is estimated in each processor as the product of the average MB quantization stepsize and the number of bits generated by each processor. The local bit allocation for the current MB is based on two measurements: the deviance from estimated buffer fullness for the current MB and the normalized spatial activity. The picture-fragment in each processor is approximated and estimated to have a uniform distribution of bits. If the local trend of generated bits begin to deviate from this estimation, a compensation factor appears to control the MB quantization scale. The global bit budget is broadcasted to all the processors to perform the allocation of target bit.

3 Experimental Results

Experiments were performed on the Intel iPSC/860 hypercube, a network of HP 9000/735 and a network of Sun Sparcstations using various number of processors. The measured time was averaged over 50 frames of a video sequence, using a set of five video sequences: *Football*, *Table Tennis*, *Salesman*, *Miss America*, and *Swing*. All of these sequences are very representative of different kind of motion and are very useful regarding motion estimation.

The time to process 50 frames was not necessarily the same in each processor, so the average was also taken over all the processors. Depending on the availability of processors, several such set of measurements were taken using 1, 2, 4, 8, 16, 32 and 64 processors for each set. All the timing data were measured by using an *Express* function *extime()*, which provides microsecond granularity.

As input, we used a constant bit-rate of 5 Mbps, with a I-P frame distance of 3, while the search window was of ±11 pels for P-pictures and ±10 pels for B-pictures. To measure the quality of the video, we used the *Peak Signal-to-Noise Ratio* (PSNR), as there exists no good and simple metric for this measure [4]. The PSNR of a video is defined as follows:

$$PSNR = 10\log_{10}\frac{255\times 255}{MSE} \qquad (2)$$

where MSE is the Mean Square Error. The larger the PSNR, the better the quality. Table 1 shows the average PSNR for different sequences.

Table 1: Picture Quality (luminance only).

Name of Sequence	Average PSNR	
	Full Serach (dB)	Log Search (dB)
Football	37.1060	34.5281
Table Tennis	38.5159	35.7497
Salesman	39.8703	34.5039
Miss America	42.2098	37.1647
Swing	41.4682	37.3657

Figures 2 and 3 provide comparison among the modules regarding their time-requirement for both search methods. Figures 4 and 5 show the plots of overall speedup of the encoding process for both search methods. Figures 6 and 7 depict the encoding rates for both search methods. Table 2 shows the timings of the computational modules of the *Swing* sequence using various number of processors on the Intel iPSC/860 for log-search implementation. Due to stringency of space, similar tables for other sequences and for full-search are omitted.

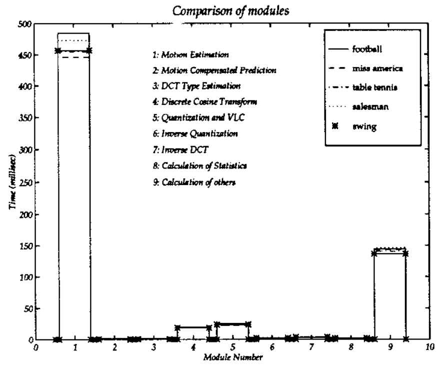

Fig. 2. Comparisons of modules: full search.

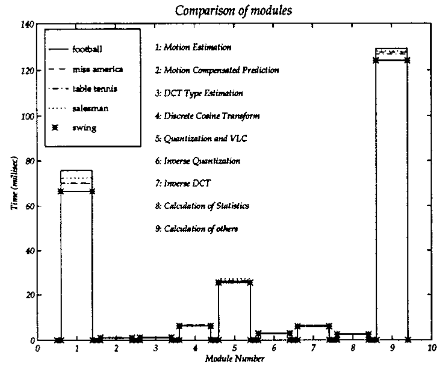

Fig. 3. Comparisons of modules: 2D-log search.

4 Conclusions

In this paper, an efficient and scalable parallel implementation of the MPEG-2 encoder was described. The data distribution strategies were discussed. The implementation was performed using the SPMD paradigm while various MPEG-2 modules were parallelized. Noticeable improvements in speedup were achieved. In our implementation, the I/O was not handled by dedicated processors, otherwise further improvement in speedup is expected. We used full-search and 2-D log-search for motion estimation but our implementation allows inclusion of faster algorithms which can further reduce the total computation time. We have used only 64 processors of the Intel IPSC/860 and even fewer processors for the networks of workstations. We have achieved an average frame rate of 4.15 fps and the figures show that the speedup is increasing with the number of processors. Therefore, by using more processors, it is easy to have real-time MPEG-2 video encoder [2].

TABLE 2: TIMINGS (MILLISECOND) FOR *Swing* SEQUENCE: *iPSC/860.*

Name of module	Number of Processors						
	1	2	4	8	16	32	64
Motion Estimation	3687.91	1855.45	938.85	478.44	245.49	127.07	66.41
Motion Compensated Prediction	56.87	28.65	14.56	7.45	3.85	1.98	1.03
DCT Type Estimation	59.56	29.97	15.24	7.82	4.04	2.10	1.09
Discrete Cosine Transform	339.86	170.99	86.65	44.13	22.52	11.62	6.05
Quantization and VLC	1395.28	702.34	356.46	181.96	93.60	48.49	25.27
Inverse Quantization	155.73	78.40	39.77	20.34	10.48	5.45	2.85
Inverse DCT	328.41	165.30	83.89	42.80	21.98	11.37	5.90
Calculation of Statistics	130.80	65.85	33.42	17.10	8.80	4.59	2.43
Calculation of Others	141.09	151.22	145.31	129.28	136.34	127.50	124.04
Total	6295.51	3248.17	1714.15	929.32	547.10	340.17	235.07

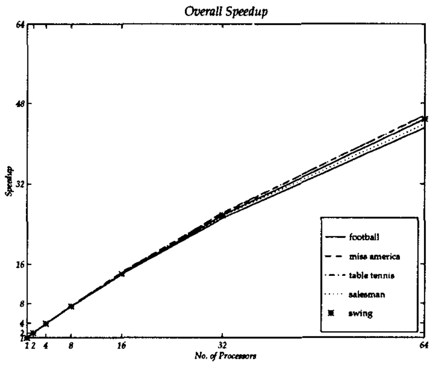

Fig. 4. Overall speedup: full search.

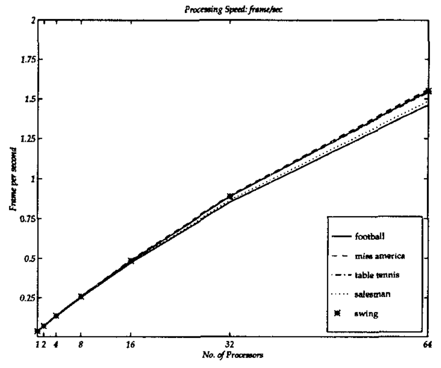

Fig. 6. Encoding rate: full search.

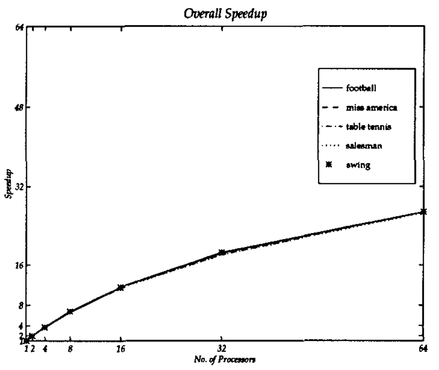

Fig. 5. Overall speedup: 2D-log search.

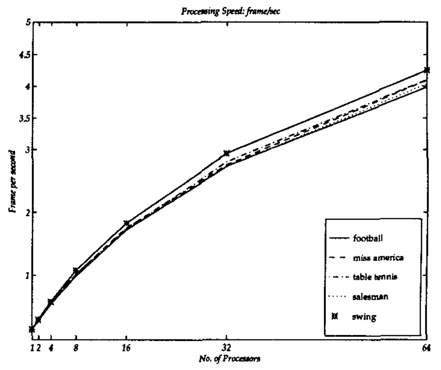

Fig. 7. Encoding rate: 2D-log search.

REFERENCES

[1] T. Akiyama *et al.*, "MPEG2 video codec using image compression DSP", *IEEE Tran. on Consumer Electronics*, VOL. 40, NO. 3, August 1994, pp. 466-472.

[2] S. M. Akramullah, *Real-Time MPEG-2 Video Encoding on Parallel and Distributed Systems*, MPhil Thesis, The Hong Kong University of Science and Technology, July 1995.

[3] C. Fogg *et al.*, "ISO/IEC Software Implementation of MPEG1 Video", *Proc. of the SPIE*, VOL. 2187, Feb. 1994, pp. 249-257.

[4] K. L. Gong, L. A. Rowe, "Parallel MPEG-1 Video Encoding", *1994 Picture Coding Symposium*, Sacramento, CA, Sept. 1994.

[5] ISO Committee Draft 13818-2, *Generic Coding of Moving Pictures and Associated Audio: Recommendation H.262*, ISO/IEC JTC1/SC29 WG11/602, Seoul, November 1993.

[6] J. R. Jain and A. K. Jain, "Displacement Measurement and its Application in Interframe Image Coding", *IEEE Tran. on Comm.*, VOL. 29, NO. 12, Dec. 1981, pp. 1799-1808.

[7] A. C. P. Loui *et al.*, "A Parallel Implementation of the H.261 Video Coding Algorithm", *Proc. of the IEEE Workshop on VSPC*, Raleigh, NC, September 2-3, 1992, pp. 80-85.

[8] *MPEG-2 Video Encoder, Version 1.1a*, MPEG Software Simulation Group, July 1994.

[9] *Express System User's Guide*, Parasoft Corp., CA, 1992.

[10] K. Shen, L. A. Rowe, E. J. Delp, "A parallel implementation of an MPEG1 encoder: Faster than real-time!", *Proc. of the SPIE*, San Jose, CA, Feb. 1995.

[11] J. van der Meer, "The Full Motion System for CD-I", *IEEE Tran. on Consumer Elec.*, VOL. 38, NO. 4, Nov. 1992.

[12] H. H. Taylor *et al.* "An MPEG Encoder Implementation on the Princeton Engine Video Supercomputer", *Data Compression Conf. 1993*, Los Alamitos, CA, 1993, pp. 420-429.

[13] Z. Wang, "Fast algorithms for the Discrete W Transform and for the Discrete Fourier Transform", *IEEE Tran. on ASSP*, VOL. ASSP-32, NO. 4, August 1984, pp. 803-816.