Parallel H.263 video encoder in normal coding mode

J.P. Cosmas, Y. Paker and A.J. Pearmain

A parallel H.263 video encoder, which utilises spatial parallelism, has been modelled using a multi-threaded program. Spatial parallelism is a technique where an image is subdivided into equal parts (as far as physically possible) and each part is processed by a separate processor by computing motion and texture coding with all processors each acting on a different part of the image. This method leads to a performance increase, which is roughly in proportion to the number of parallel processors used.

Introduction: In the coding of images using parallel techniques there are two main approaches: functional parallelism and spatial parallelism. With functional parallelism a separate unit in a parallel pipeline computes each function in the coding algorithm [1]. Using this method it is difficult to reassign processing power when there is a large variation in the processing power requirements at a functional level and impossible if dedicated hardware is used.

With spatial parallelism, an image is subdivided into equal parts, which are processed by separate processors [2, 3]. Spatial parallelism leads to more equal distribution of the processing power required among the available parallel processors and is also able to redistribute the processing load by reassigning larger/smaller areas of the image to the processors. This makes the spatial parallel technique ideal for increasing the processing power of the H.263 encoder [4] which is required to operate on up to five standard picture formats (e.g. sub-QCIF, QCIF, CIF 4CIF and 16CIF).

Algorithm enhancements: The key issue to parellelising the H.263 encoder is whether the macroblock texture and motion coding algorithms require picture element (pel) information from within the macroblock that is being coded or whether additional pel information from outside is also required. If the former is true then the macroblocks can be shared among the available processors whereas if the latter is true then it is much more difficult to share the macroblocks among the available processors.

A parallel image-coding scheme has been designed so that the image is split as equally as possible into horizontal regions along macroblock boundaries in order to minimise data dependency between blocks.

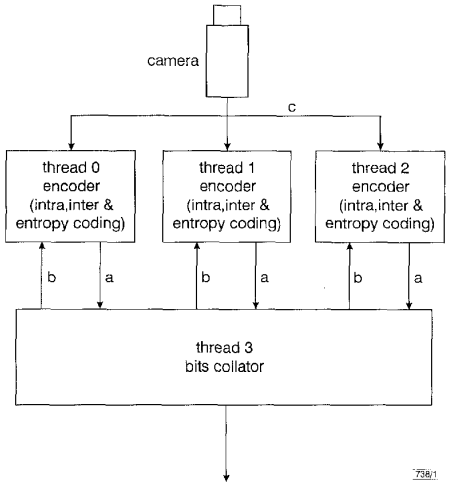

Fig. 1 *Parallel system diagram*

a Entropy coded bits, reconstructed regional image
b Fully reconstructed image
c New image

When using INTRA mode coding, all the pixels that are required for processing the DCT lie within the 16 × 16 macroblock that is being computed. This poses no problems for the parallel coding since the four, luminance and two, chrominance 8 × 8 blocks lie within the macroblock boundary. When using INTER mode coding in normal mode, the motion vector components of the macroblocks (horizontal and vertical) are coded differentially by using a spatial neighbourhood of three motion vectors already computed [3]. These three motion vectors are candidate predictors for the differential coding. Therefore, to perform motion prediction in parallel on each horizontal region of an image, motion estimation is performed on each region plus the row of macroblocks immediately above the region. Motion prediction is then computed only for the given region using the motion information of the blocks immediately above for the macroblocks at the boundary. In normal mode, the symbols generated by the intra and inter-coding are Huffman coded. Since Huffman coding of a particular symbol is independent of previous symbols then the macroblocks within each region can be coded separately and combined by simply collating the bits, taking care to exclude any padding that may have been added. When coding the new image the fully reconstructed image is passed back from the collator to the separate threads for coding.

```
Main() {
   For k = 1 to 3 Start Thread k
   Get Images
   For k = 1 to 3 Image Ready k = TRUE
   Wait for Image Coded 1 to 3
   Collate bits
   For ever loop {
      Get Images
      For k = 1 to 3 Image Ready k = TRUE
      Wait for Image Coded 1 to 3
      For k = 1 to 3 Image Coded k = FALSE
      Collate bits
   }
}

thread(n) {
   Wait for Image Ready n
   Image Ready n = FALSE
   Set Mutex
   Code Intra
   Image Coded n = TRUE
   Release Mutex
   For ever loop {
      Wait for Image Ready n
      Image Ready n = FALSE
      Set Mutex
      Code Inter/Intra
      Image Coded n = TRUE
      Release mutex
   }
}
```

Fig. 2 *Pseudocode of software multi-threaded model of parallel system*

Results: The parallel image coding has been implemented using 2, 3 and 4 parallel threads, each with an additional thread for re-collating the output bitstream. The structure of the parallel system is shown in Fig. 1 and the software implementation is shown in Fig. 2. The processing power of the CPU was directed towards each thread in sequence by using mutexes within each thread. Mutexes allow all the processing power to be directed towards one thread while all other threads are put to sleep and consume minimal processing power. Synchronisation of the threads and mutexes are co-ordinated using flags: Image Ready and Image Coded, as shown in Fig. 2. Tests were made on 35 frames of QCIF images for the Newsreader scene and 35 frames of CIF images for the Children scene using a 200MHz Pentium PC with 64 Mbyte of RAM. The results of the speed-up are given in Tables 1 – 4.

Table 1: Results of intra-coding speed-up (QCIF images of Newsreader scene)

Number of threads	Coding type	Thread 1 Av. time	Thread 2 Av. time	Thread 3 Av. time	Thread 4 Av. time	Average speed-up	Worst-case speed-up
		s	s	s	s		
1	intra	0.22				1.00	0.22
2	intra	0.13	0.15			1.57	1.47
3	intra	0.08	0.17	0.12		1.74	1.29
4	intra	0.08	0.12	0.12	0.09	2.19	1.88

Table 2: Results of inter-coding speed-up (QCIF images of Newsreader scene)

Number of threads	Coding type	Thread 1 Av. time	Thread 2 Av. time	Thread 3 Av. time	Thread 4 Av. time	Average speed-up	Worst-case speed-up
		s	s	s	s		
1	inter	0.21				1.00	1.00
2	inter	0.14	0.09			1.91	1.57
3	inter	0.07	0.13	0.07		2.44	1.69
4	inter	0.07	0.05	0.05	0.12	3.03	1.83

The time required to collate the bits was negligible and therefore was not tabulated. For each thread, the average time to code a horizontal region was measured over a sequence of 35 frames. The average speed-up was computed as the time to code the images using a single thread divided by the average time to code

Table 3: Results of intra-coding speed-up (CIF images of Children scene)

Number of threads	Coding type	Thread 1 Av. time	Thread 2 Av. time	Thread 3 Av. time	Thread 4 Av. time	Average speed-up	Worst-case speed-up
		s	s	s	s		
1	intra	0.78				1.00	1.00
2	intra	0.46	0.62			1.44	1.13
3	intra	0.261	0.41	0.461		2.07	1.69
4	intra	0.26	0.19	0.14	0.36	3.28	2.16

Table 4: Results of inter-coding speed-up (CIF images of Children scene)

Number of threads	Coding type	Thread 1 Av. time	Thread 2 Av. time	Thread 3 Av. time	Thread 4 Av. time	Average speed-up	Worst-case speed-up
		s	s	s	s		
1	inter	0.95				1.00	1.00
2	inter	0.56	0.76			1.44	1.25
3	inter	0.37	0.69	0.65		1.67	1.37
4	inter	0.35	0.43	0.35	0.58	2.22	1.63

the images using N threads. The worst-case speed-up was computed as the time to code the images using a single thread divided by the worst-case coding time experienced by a thread.

As the number of threads increased, the average speed-up decreased for both intra- and inter-coding modes. The total time required for coding the image increased due to the overhead of repeating various initialisation functions. The variation in the amount of time required to code image regions for intra-coding was due to the varying computational requirements of the Huffman coder and for inter-coding the variation was due to the varying computational requirements of the motion estimator and the Huffman coder.

Conclusions: The speed-up for a spatially parallel H.263 video encoder was in proportion to the number of threads used but not as great as expected for CIF/QCIF images for both inter- and intra-modes. This was due to additional overhead time to initialise parameters and to allocate temporary arrays that was repeated for each image region. The worst-case speed-up shows what can be obtained using multi-threads on a single processor but the parallelisation algorithms would really show an improvement in performance if run on multiple processors particularly if interprocessor communication techniques using DMA or shared memory are used. The average speed-up shows what can be obtained when load-sharing algorithms are in place. For the speed-up to be improved, further work is needed to remove unnecessary allocation/reallocation of memory space and to balance the load among the threads.

16 September 1998

Electronics Letters Online No: 19981487

J.P. Cosmas, Y. Paker and A.J. Pearmain (*Department of Electrical Engineering, Queen Mary & Westfield College, London E1 4NS, United Kingdom*)

References

1 LIN, W., GOH, K.H., TYE, B.J., POWELL, G.A., OHYA, T., and ADACHI, S.: 'Real time H.263 video codec using parallel DSP'. IEEE Int. Conf. Image Processing, 1997, Vol. 2, pp. 586–589

2 AKRAMULLAH, A., AHMAD, I., and LIOU, M.L.: 'A data parallel approach for real time MPEG-2 video encoding', *J. Parallel Distribut. Comput.*, 1995, **30**, pp. 129–146

3 BOUVILLE, C., HOULIER, P., DUBOIS, J.L., MARCHAL, I., THEBAULT, B., and KLEFSTAD, M.: 'A flexible MPEG real time video codec'. ICIP-96, Lausanne, Switzerland, September 1996, Vol. 3, pp. 829–832

4 Draft ITU-T Recommendation H.263. 'Video coding for low bitrate communication'. March 1996

Reduction of ringing noise in transform image coding using simple adaptive filter

A. Kaup

A numerically simple filter for reducing ringing noise in transform coded images is proposed. The filter adapts to the local image characteristics and has been specifically designed to amend the current deblocking filter in H.263. It is shown that by adding the proposed filter both subjective and objective image quality improve.

Introduction: A well-known deficiency of current video coding standards is the visibility of image degradations at high compression ratios. These image degradations manifest themselves in blocking artefacts due to the rigid block partitioning of the image and ringing noise preferably around edges due to coarse quantisation. Both effects are visually very annoying and have a substantial impact on the subjectively perceived image quality.

A widespread principle for overcoming this problem is lowpass filtering of the decoded image in either the spatial [1] or temporal direction [2]. To reduce the numerical complexity, these filters are sometimes restricted to block boundaries, thus specifically tackling blocking noise. A very efficient filter of this type has been standardised and included as optional Annex J in the recently released Version 2 of H.263 [3]. However, while this filter removes much of the blocking noise it does not tackle ringing and mosquito noise.

Since global smoothing for reducing ringing artefacts tends to eliminate important image details, some proposals try to enhance the decoded image by incorporating prior knowledge about typical image data. This leads to maximum *a posteriori* (MAP) approaches where the Bayesian paradigm can be used to solve an estimation problem involving both *a priori* knowledge and the decoded image data [4, 5]. However, since this estimation process usually involves numerical optimisation of non-convex functionals, the principle is computationally very demanding.

Apart from image restoration after decoding it is also possible to diminish blocking and ringing noise by image preprocessing at the encoder site. This train of thought has for example, been followed in [6], where the quantisation noise of DCT coefficients is shifted from the block boundaries to the inner part of the block. Other proposals employ Dolby-like noise suppression techniques for reducing blocking artefacts [7]. Although such noise shaping requires a matched receiver for best performance, a standardised receiver which does not know about the encoder modifications can still decode an image of reasonable quality.

In this Letter we will follow a different approach motivated by the recently released deblocking option in H.263. This filter has been placed within the prediction loop such that the decoded and filtered image serves as reference for the next frame to code. While this makes the filter description normative, it has the advantage that only single frame storage is needed for prediction as well as display. Following this reasoning, we will describe a possible amendment to this option which specifically takes care of the remaining ringing and mosquito noise.

Deringing filter: Consider a motion compensated reconstructed image in the prediction loop of H.263 after the deblocking filtering as described in Annex J of [3] has taken place. While the resulting image typically has only very little blocking noise remaining at block boundaries, it does still show considerable ringing artefacts especially towards the centre of the image blocks. A *deringing* filter thus should remove this noise without unduly destroying important high frequency image details. This can be achieved by an adaptive lowpass filter where the filter mask varies depending on the local image characteristics.

Table 1: Local 3 × 3 neighbourhood considered for filtering

g_1	g_2	g_3
g_4	g_5	g_6
g_7	g_8	g_9

Consider a local 3 × 3 neighbourhood of decoded image pixels as depicted in Table 1 having the grey levels g_1 to g_9. Grey level here refers to either luminance or chrominance data. The deblock-