A Case Study in Pipeline Processor Farming: Parallelising the H.263 Encoder

H. Sava,
Department of Electronic Systems Engineering, University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, U.K

M. Fleury,
Department of Electronic Systems Engineering, University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, U.K

A. C. Downton
Department of Electronic Systems Engineering, University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, U.K

A. F. Clark
Department of Electronic Systems Engineering, University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, U.K

Abstract

This paper describes the parallelisation of the H.263 hybrid video encoder algorithm based upon a pipelines of processor farms (PPF) paradigm. In addition, a data-farming template, which can be very useful for several image coding algorithms, was incorporated in the PPF model. A variety of parallel topologies were implemented in order to obtain the best time performance for an eight processor distributed-memory machine. Results show that, due to communication overheads and algorithm constraints, the speed-up performance is below the value predicted by static analysis. However, the design examples indicated how to modify the PPF methodology in identifying those algorithm components which restrict scaling performance. The paper highlights the problems associated with the parallelisation of sequential algorithms and emphasises the need for generic tools to facilitate such conversion.

1 Introduction

H.263 is a new standard for very low bit-rate videocoding (<64kbps) which is in the process of ratification by ITU-T. One key application will be PSTN videotelephony, i.e. videotelephony on normal analogue telephone lines. The standard has been developed collaboratively by researchers in telecommunications organisations around the world, and algorithms for an H.263 decoder and encoder implemented by Telenor (Norwegian Telecom) are freely available over the Internet [1]. On a Sparcstation 20, the decoder runs in real-time, though

the encoder runs at only about 2 frames/s. In the short term, before real-time VLSI hardware implementations become widely available, there is therefore significant interest in parallelising the encoder algorithm to obtain real-time performance in distributed- or shared-memory multiprocessor environments.
This paper describes a case study which applies a pipeline processor farming (PPF) methodology [2] to parallelise the Telenor H.263 image encoding algorithm. The goal is to obtain a portable, parallel, real-time H.263 encoder which is capable of running in a range of multiprocessor environments (e.g., i860-based MIMD machine, TMS320C40 parallel DSPs, multiple workstations running PVM [5], shared-memory multiprocessor workstations) with minimal software changes. The paper not only describes the architecture of the parallelisation solution adopted for this application but also addresses general issues related to the conversion of sequential algorithms to parallel ones and emphasises the need for generic tools to facilitate such conversion.

2 H.263 algorithm characteristics

Figure 1 shows a block diagram of the H.263 encoder.

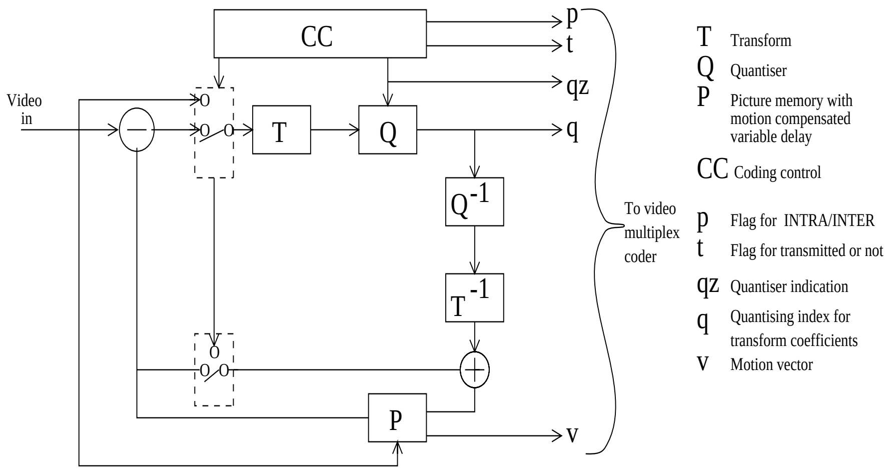

Figure 1: Block diagram of the structure of the H.263 encoder

The basic configuration of the video source coding algorithm has been developed from ITU-T recommendation H.261 and is a hybrid of inter-picture prediction to utilise temporal redundancy and transform coding of the remaining signal to reduce spatial redundancy [1, 7]. In addition to the basic video source coding algorithm, four negotiable coding options are available: Unrestricted Motion Vectors, Syntax-based Arithmetic Coding, Advanced Prediction and PB-frames. The coding method uses 16×16 macro-blocks, 8×8 sub-blocks, motion estimation and compensation, DCT transform of prediction errors, run-length coding and variable-length codewords.

3 Parallelising the H.263 algorithm

3.1 Parallel design model

The PPF design methodology can be used to decompose existing sequential applications onto any type of parallel processor network with any communication model [2]. The methodology proceeds from the observation that embedded signal processing systems with continuous data flow may be characterised as consisting of a series of independent processing stages. It therefore maps the sequential algorithm structure to a generalised parallel architecture based upon a pipeline of stages with well-defined data communication patterns between them. Each stage of the pipeline then exploits parallelism in the most appropriate way, for example, data parallelism applied at various levels, algorithmic parallelism, or temporal multiplexing of complete data sets. This decomposition has been applied successfully to similar applications in the past [3, 6].

3.2 Static analysis of H.263 algorithm

The first step of the design model is to decompose the sequential software into pipeline stages using top-down profiling data derived from sequential execution. This stage identifies the individual stages and their relative computational requirements and allows the required number of processors in each stage to be calculated directly. In its simplest form, the model ignores communication overheads between processors and assumes static task execution times. Although practically not achievable, this is useful in determining speed-up trends.
Analysis of the execution profile of the H.263 encoder using the *gprof* profiler on test sequences such as "Mother and Daughter" and "Car Phone" shows that only three functions have execution times large enough to be measurable at the sampling accuracy used. One of these functions, *CodeOneIntra*, is called only once to code the initial frame using intra-frame coding alone. Therefore, this function was excluded from the analysis, although it is structurally similar to the inter-frame coding used for other frames and so could in principle be included in the encoder/decoder stage of the pipeline. The analysis of the other two major functions, *CodeOneOrTwo* and *ComputeSNR*, shows that 98% of the execution time is spent on *CodeOneOrTwo* regardless of the coding options chosen.
Table 1 shows the breakdown of the execution time of the main function *CodeOneOrTwo* into sub-functions, and includes the 8 most significant sub-functions which constitute 97.9%of the execution time of *CodeOneOrTwo*.

Function Name	%*CodeOneOrTwo* execution time (default mode)	%*CodeOneOrTwo* execution time (full options)
MotionEstimatePicture	63.8	64.8
MB_Decode	13.1	9.1
MB_Encode	12.8	9.4
PredictP	2.6	5.2
InterpolateImage	1.9	1.9
MB_Recon_P	1.3	5.1
Clip	1.2	1.1
ReconImage	1.2	1.3

Table 1: Top-down profiling data for the 8 most intensive functions within the CodeOneOrTwo function of the H.263 coder

Nearly 64% of the execution time of *CodeOneOrTwo* is expended in the motion estimation function and the remaining time on the *Encoder-Decoder* function. From the above analysis it was concluded that these two functions should be placed in separate PPF pipeline stages within the frame feedback loop, and each image frame subdivided into half-frames for processing concurrently in consecutive pipeline stages. Comparison of the execution times of the two stages shows that their static execution times are approximately in the ratio 2:1; hence a balanced pipeline can be achieved by populating the respective processor farms with workers in this ratio. An implementation based on this architecture is shown in Figure 2.
According to Amdahl's law, the speed-up which can be achieved is defined largely by any residual sequential elements within an application's algorithm [4]. Hence, an implementation based upon this architecture would result in an overall speed-up of up to six according to Amdahl's law.

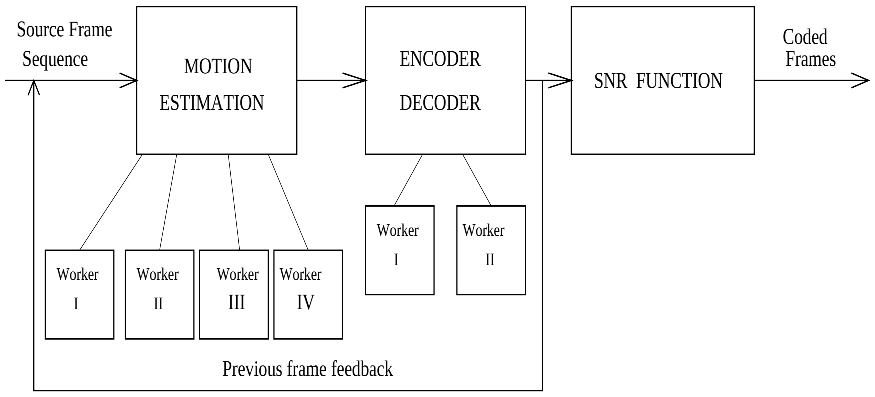

Figure 2: Pipeline architecture of H.263 encoder

In the present study, the scale of the structure was restricted to eight processors because of the number of processors in the Transtech Paramid machine used. The Paramid [8] consists of compound processor nodes comprising a transputer communication co-processor and i860 computational engine [9]. The processes were implemented on i860s which run at 50 MHz, the transputer communication links being set at 20 MHz. The “Car Phone” image sequence, comprising some 74 frames, was used as a test data.

4 General parallel design issues

Although the overall structure of processors in the parallel implementation was determined directly from the execution time of the sequential program, the conversion to a parallel algorithm suitable for distributed-memory processors presents numerous software conversion problems. In the case of a single processor, where shared memory is used, efficient inter-function communication is generally provided by passing pointers to data structures. However, to support the general case of distributed-memory MIMD processors (e.g., the Paramid machine), required data must be explicitly passed down the pipeline and distributed to worker processors in each farm. Thus, to support parallelisation of existing application codes, generic analysis tools are required to track memory use and to convert communication at application partitioning points from referential to explicit. Subsequently, generic design tools are needed to support and simplify the partitioning process and to enable the application to be rescaled easily.

At this stage, *Purify*, a memory usage analyser, was found to be useful when modifying the sequential code. The *Purify* software minimises debugging time due to its capability of identifying when an error occurs, the origin of the error and the relevant line number in the source code. These features are very useful, especially when dealing with complex and large software applications such as H.263 algorithm. Furthermore, most of the bugs occuring at this stage are related to memory leaks as a result of converting pointer reference variables to statically allocated variables. In this context, the availability of *Purify* was found very useful.

Table 2 shows comparative execution for Sparc 20, Sparc 5 and single i860 of the sequential program.

Processor	Execution speed(secs./frame)
Sparc 20 (passing pointers)	0.40
Sparc 5 (passing pointers)	3.12
i860 (passing pointers)	3.20
i860 (passing data)	4.47

Table 2: Comparative overall performance of Sparc 20, Sparc5, and i860 for H.263 algorithm

The results presented in Table 2 suggest that a practical implementation using up to eight i860 nodes is unlikely to exceed the performance of a Sparc 20, as the i860 has a somewhat lower computational performance than the Sparc 20. However, such an implementation would at least give an insight to parallelisation of H.263 on a more powerful machine.
Another issue that requires particular attention, especially in the case of data-intensive applications, is that of data distribution. To handle this problem, a data-broadcast primitive was incorporated into the PPF model.

5 PPF Primitives

In order to facilitate the application of the PPF model to a wide range of embedded applications, and to minimise the debugging time for each new application, processor farm primitives are required.
Although there are cases (e.g., 3L Parallel C) where such primitives already exist, this was not the case for the Paramid system. Furthermore, existing primitives are largely designed to support only single processor farms, rather than the pipeline processor farm model required for embedded applications with continuous data flow. Thus, to support the PPF model a generic, reusable, demand-based processor farm primitive was developed [10].
Another evident problem associated with parallel image processing applications is that of immense data communications. Functions like *MotionEstimatePicture* or *EncodeDecode*, which constitute the bulk of hybrid coding algorithms, require several images and motion vectors which in themselves contain large amounts of data. One way to reduce the data communication would be to send to each worker only the required data for processing. However, this would lead not only to a complicated software structure, but, moreover, to an unscalable solution. To tackle this problem a data-farming template incorporating a multi-cast facility was implemented. This template distributes the data from the master processor to all workers simultaneously and can easily be scaled to the required number of processors. Furthermore, a buffering facility was designed in order to hide communication latencies. The data-farming template can also be adapted to use a data-flow synchronisation mode between workers during the initial phase of multi-casting. This type of synchronisation was used in this application at the start of processing each half-image by all the workers of a master.

6 Results and discussion

Due to the complexity of the H.263 algorithm, an incremental approach was adopted during the parallel implementation of the algorithm. Figure 3 depicts the developing stages of the parallel implementation and Table 3 presents the corresponding results on a time-per-frame basis. In all these cases the image frame is sub-divided so that, while one half-frame is being coded, motion estimation is simultaneously being carried out on the next half.

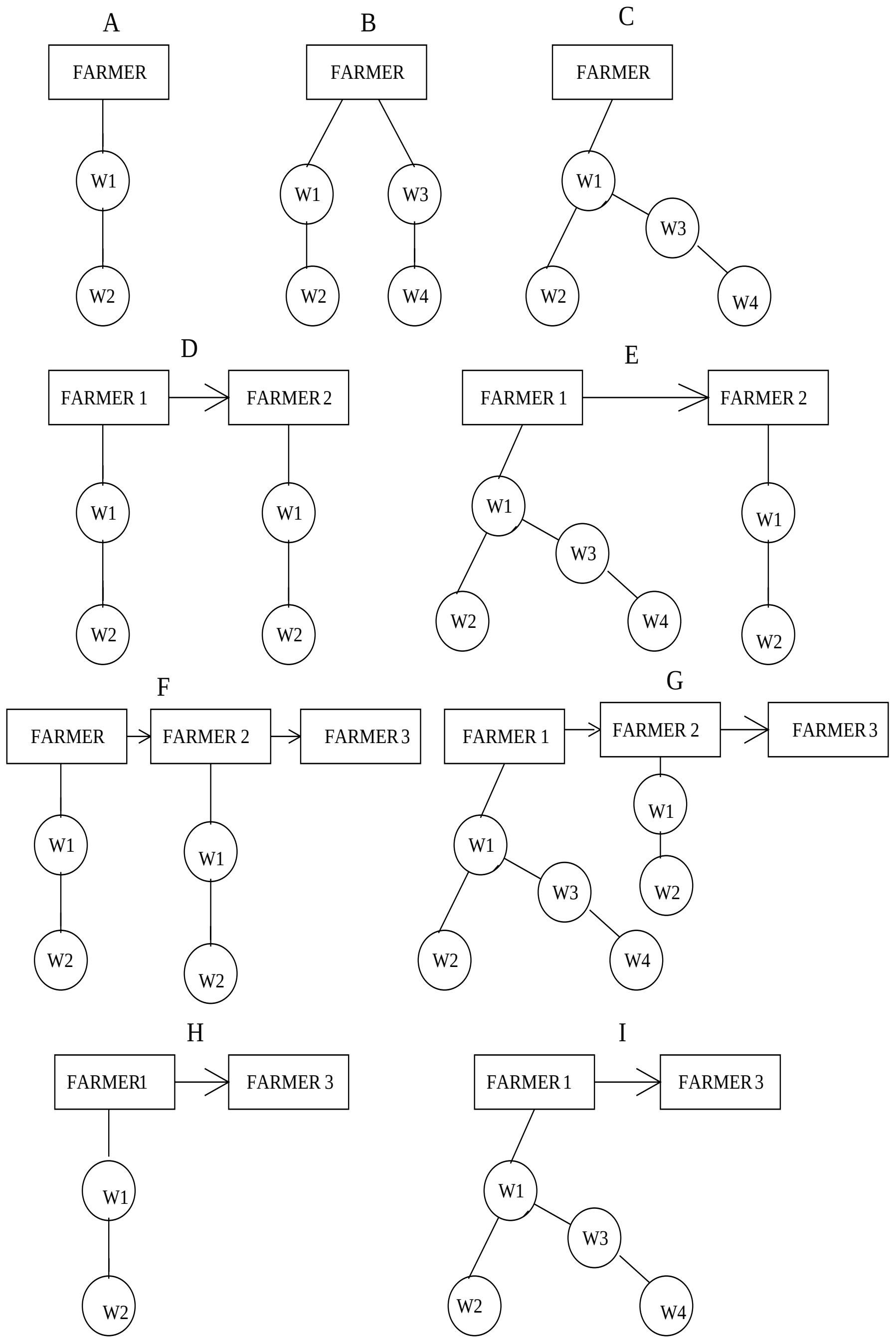

Figure 3: Implemented parallel topologies for the H.263 encoder

Parallel Topology	Execution speed(secs./frame)
Single i860 (passing pointers)	3.20
Single i860 (passing data)	4.47
Figure 3(a)	2.04
Figure 3(b)	2.2
Figure 3(c)	2.05
Figure 3(d)	2.52
Figure 3(e)	2.6
Figure 3(f)	2.46
Figure 3(g)	2.48
Figure 3(h)	1.86
Figure 3(i)	1.89

Table 3: Comparative overall performance of several parallel topologies for H.263

From Table 3 it is easy to see that the proposed topology from the static analysis (Figure 3g) does not match the speed-up predicted accorded to Amdahl's law. Indeed, the speed-up achieved in this case is about three times less than the theoretically predicted speed-up. This is mainly caused by the large amount of data that has to be communicated between farmers and respective workers. From Table 3 it can also be seen that the extension to a second farmer, which pipelines the encode-decode part of the algorithm, deteriorates the performance of the parallel structure. This can easily be seen from the comparison of time performance of Figure 3a-c with Figure 3d-e. In the latter case a 25% increase in per-frame processing time is measured. This is mainly caused by an inherent restriction within the H.263 algorithm which requires a row-by-row update of the quantisation variable. This constrains the encoder and decoder to process on only a row-by-row basis and increases the number of messages sent from the the workers to the relevant farmer. The measurement of the processing and communication times spent in the second farmer showed that both these times are equal. This finding suggests that, for the present hardware system, the extension to a second farmer is inappropriate. However, for another machine with higher communication bandwidth, a second pipeline stage utilising a greater number of processors may be worthwhile. Furthermore, the requirement for row-by-row update of quantisation can be relaxed in many cases and, if this is possible, the communication overhead of using a second pipeline stage could be reduced further.
According to Table 3, the best results are achieved by the topology described in Figure 3h. In this case the processors are entirely balanced and a further reduction in time is achieved from the introduction of a third farmer. However, it must be said that even for this case the time performance is far below the upper-bound speed-ups required for real-time implementation.

7 Conclusions

This paper has presented an application of the PPF methodology to parallelising the H.263 encoder. Although a real-time version of H.263 has not yet been achieved, this is largely due to communication bandwidth constraints of the current parallel machine. Work is currently under way to port the present parallel implementation to a TMS320C40 parallel DSP environment which has an order of magnitude higher communication bandwidth. This porting process is assisted by the use of a generic design methodology utilising data-farming templates developed specifically to support continuous data-flow embedded applications. The application highlights the impact of the communication overheads and inherent coding restrictions in achieving the upper-bound speed-ups predicted by a static analysis of the algorithm. These algorithm constraints are also relevant in designing real-time VLSI implementations of H.263.

8 Acknowledgements

This work is being carried out under EPSRC research contract GR/K40277 'Portable software tools for embedded signal processing applications' as part of the EPSRC PSTPA (Portable Software Tools for Parallel Architectures) directed programme.

References

[1] "ITU-T Recommendation H.263", *Technical Report*, 1995.

[2] A. C. Downton, R. W.S. Tregidgo, A. Cuhadar "Top-down Structured Parallelisation of Embedded Image Processing Applications", *IEE Proc. Vis. Image Signal Process.*, Vol. 141, No. 6, pp. 431–437, Dec. 1994.

[3] A. C. Downton "Generalised Approach to Parallelising Image Sequence Coding Algorithms", *IEE Proc. Vis. Image Signal Process.*, Vol. 141, No. 6, pp. 438–445, Dec. 1994.

[4] G. M. Amdahl, "Limits of Expectation", *Int. J. Supercomput. Appl.*, pp. 88–94, No. 2, 1982.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, "PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing", *The MIT Press, Cambridge*, MA, USA, 1994.

[6] A. C. Downton "Speed-Up Trend Analysis for H.261 and Model-Based Image Coding Algorithms Using a Parallel-Pipeline Model", *Signal Processing: Image Communication*, Vol. 7, pp. 489–502, 1995.

[7] D. Bailey, M. Cressa, J. Fandrianto, D. Neubauer, H. Rainnie, Ch-Sh. Wang, "Programmable Vision Processor/Controller for Flexible Implementation of Current and Future Image Compression Standards", *IEEE Micro*, pp. 33–39. Oct. 1992.

[8] Transtech Parallel Systems Corporation "The Paramid Users Guide", 1993.

[9] M. Atkins "Performance of the i860 Microprocessor", *IEEE Micro*, pp. 24–78, Oct. 1991.

[10] M. Fleury, H.P. Sava, A.C. Downton and A.F. Clark "Designing and Instrumenting a Software Template for Embedded Parallel Systems", *In this volume.*