Evaluation of Two Compiler-Based Approaches for the Parallelisation of an MPEG-2 Decoder

Arnaud Laffitte, Rizos Sakellariou and John R. Gurd

Department of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, U.K.

Abstract. In this paper, we evaluate two different approaches for the compiler-based parallelisation of a C program for MPEG-2 decoding. The first approach experiments with a commercial auto-parallelising compiler for exploiting coarse-grain parallelism on a Silicon Graphics Power Challenge, while the second approach experiments with a simple assembly code scheduler for exploiting instruction-level parallelism. Results obtained through the high-level auto-parallelisation of the code are disappointing, far from those that can be achieved through manual parallelisation. On the other hand, better results are achieved when exploiting instruction-level parallelism; the corresponding compiler technology appears to be more efficient.

1 Introduction

In recent years, research developments into producing digital representations of audiovisual information have led to a whole spectrum of new needs and possibilities; applications such as Digital Television have already become a practical reality. An enabling factor towards this aim has been the standardisation of video coding. MPEG-2 is the most famous of a series of adopted standards, but its computational complexity still limits its widespread use. This is particularly important for decoding, which must be performed in real-time. Although hardware solutions are available, software decoding offers greater flexibility.

In this paper, we examine the extent to which general-purpose approaches for compiler-based parallelisation of codes can lead to performance improvements of an MPEG-2 software decoder. Two architectural models are examined. After a short overview of the MPEG-2 decoder, the results are given in the next sections.

2 The MPEG-2 Video Decoding Codec

The MPEG video coding standards define a compression technique which exploits spatial and temporal correlation to achieve high compression ratios. MPEG-2 [4] targets primarily professional video and Digital Television applications and has rapidly become popular. MPEG-2 video streams present a layered structure, with the following hierarchy: The *Video Sequence* is the highest layer and consists of *Groups of Pictures* (GOP) each of which groups a number of adjacent pictures. The *Picture* is the primary coding unit of a video sequence and can be encoded into one of three types. Pictures are subdivided into *slices*, each of which corresponds to a fragment of a row in the picture and consists of one or more contiguous *macroblocks* which are formed by a number of *blocks*. The block is an 8×8 group of pixels which is the smallest coding unit. Given an MPEG-2 stream, five stages are performed in order to decode it into a sequence of pictures: Huffman decoding, run-length decoding, inverse quantization, inverse discrete cosine transform, and motion compensation.

3 Exploiting Coarse-Grain Parallelism

The behaviour of two parallel implementations of the MPEG-2 decoder on an SGI Challenge has been analysed in [2]. The first implementation exploits parallelism at the GOP level (that is, by decoding different GOPs on different processors), while the second implementation exploits parallelism at the slice level. In both cases, close to linear speed-up was obtained. How close to these results could be the performance obtained through autoparallelising techniques, or, more generally, techniques for incremental development of parallel programs where little or no knowledge of the nature of the code exists [8], has been the main objective of our experiments. The program used was the C language implementation of the MPEG-2 decoder provided by [7], which was tested using four medium sized video sequences taken from films [5].

First, the code was instrumented on an SGI Challenge using `prof` to determine where time is spent. Then, the program was parallelised using the native SGI autoparalleliser, PCA. The parallelisation led to a performance degradation, while, in one program function, PCA transformed a loop in an erroneous way. An analysis of the results showed that PCA parallelised only parts with a low computational cost [5]. At a second stage, we tried to improve performance by means of program transformations that PCA failed to apply. Using mainly induction variable elimination, loop unrolling, and statement reordering, a speed-up of up to 1.25 on 4 processors was obtained. The overhead was found to be mostly due to barrier synchronisation as a result of the repeated parallel execution of loops with a small number of iterations. The inability of the compiler to exploit parallelism beyond the loop level (which corresponds to the macroblock level of the algorithm) has been a major obstacle in obtaining high performance.

4 Exploiting Instruction-Level Parallelism

The second part of our study concerned with the investigation and measurement of parallelism at the assembly code level (i.e., instruction-level parallelism). In order to do this, we assumed a model Very Long Instruction Word (VLIW) architecture, loosely resembling, in terms of functional units available, the Philips Trimedia TM1000 [3]. A prototype scheduler, based on the SPARC instruction set, which takes into account resource constraints and instruction latency and schedules instructions locally within each basic block, has been implemented.

Using this scheduler an approximately 46% compaction of the sequential assembly has been obtained; no large deviations were observed between different parts of the code. For the most time-consuming parts of the code, this leads to an estimated speed-up of up to 1.85 (on five functional units), which can be increased up to 1.95 by applying source-level transformations, such as loop unrolling and inlining. There have been no significant performance losses due to resource saturation. However, more advanced low-level optimisations as well as more aggressive compilation techniques [1] could uncover more parallelism [6].

5 Conclusion

The two sets of experiments briefly presented above aimed to study the performance impact of a compiler-based parallelisation approach at both the source level and the assembly level. In the first case, the results obtained were unsatisfactory. Although there is much parallelism, its detection appears to be beyond the reach of state-of-the-art parallelising compiler techniques; the particular characteristics of the code are not suitable for exploiting loop parallelism and techniques developed in the context of automatic parallelisation of scientific codes are not sufficient for the code examined. This prompts us to investigate, in the future, the feasibility of producing automatically algorithmic-level specifications of the source program that would be more amenable to an automatic parallelisation tool. Comparatively better results have been achieved when exploiting instruction-level parallelism. Although at this level the potential performance improvements may not be impressive, they may prove to be more cost-effective for the general class of multimedia and embedded applications [1].

Acknowledgements: Part of this work has been funded by the ESPRIT IV reactive LTR project OCEANS, under contract No. 22729.

References

1. B. Aarts, *et. al.* OCEANS: Optimizing Compilers for Embedded Applications. In C. Lengauer, M. Griebl, S. Gorlatch (Eds.), *Proceedings of Euro-Par'97*, Lecture Notes in Computer Science 1300, Springer-Verlag, 1997, pp. 1351–1356.
2. A. Bilas, J. Fritts, and J. P. Singh. Real-Time Parallel MPEG-2 Decoding in Software. Technical Report 516-96, Department of Computer Science, Princeton University, 1996. A shorter version is also available in the *Proceedings of the 11th International Parallel Processing Symposium (IPPS)* (Geneva, Apr. 1997), IEEE Computer Society Press, 1997, pp. 197–203.
3. B. Case. Philips Hope to Displace DSPs with VLIW. *Microprocessor Report*, 8(16), 5 Dec. 1994, pp. 12–15. See also `http://www.trimedia-philips.com/`
4. ISO/IEC 13818. *Generic Coding of Moving Pictures and Associated Audio.* 1995.
5. A. Laffitte. *Obtaining High Performance for MPEG-2 Video Decoder Software.* MSc Thesis, Department of Computer Science, University of Manchester, October 1997.
6. H. Liao and A. Wolfe. Available Parallelism in Video Applications. *Proceedings of MICRO-30*, IEEE Computer Society Press, 1997, pp. 321–329.
7. MPEG Simulation Group. `http://www.mpeg.org/`
8. G. D. Riley, J. M. Bull, and J. R. Gurd. Performance Improvement Through Overhead Analysis: A Case Study in Molecular Dynamics. *Proceedings of the 1997 International Conference on Supercomputing*, ACM Press, 1997, pp. 36–43.
9. R. Sakellariou, E. A. Stöhr, and M. F. P. O'Boyle. Compiling Multimedia Applications on a VLIW Architecture. *Proceedings of the 13th International Conference on Digital Signal Processing (DSP97)* (Santorini, July 1997), vol. 2, IEEE Press, 1997, pp. 1007–1010.