p264: Open Platform for Designing Parallel H.264/AVC Video Encoders on Multi-Core Systems

António Rodrigues, Nuno Roma, and Leonel Sousa
IST / INESC-ID
Rua Alves Redol, 9
1000-029 Lisboa - PORTUGAL
antonio.c.rodrigues@ist.utl.pt, nuno.roma@inesc-id.pt, leonel.sousa@inesc-id.pt

ABSTRACT

A highly modular and configurable platform for designing parallel H.264 video encoders on multi-core processors is presented. Departing from the H.264/AVC reference software, preliminary optimizations were conducted and new data structures were developed, in order to support the encoder's parallelization and to confer the developed platform with a flexible, user configurable and highly scalable characteristics in what concerns the number of available cores to be used in the target concretization. After a careful assessment using different instantiations of the platform, the experimental results have shown that significant and close to linear speedups in what concerns the achieved frame-rate can be obtained, by simultaneously exploiting the several different parallelization models that are made available by this platform.

Categories and Subject Descriptors

C.1.2 [**Multiple Data Stream Architectures (Multiprocessors)**]: Parallel processors; D.1.3 [**Concurrent Programming**]: Parallel programming; H.5.1 [**Multimedia Information Systems**]: Video

General Terms

Design

Keywords

H.264, Video Encoder, Multi-Core Processors, Parallel Systems

1. INTRODUCTION

The H.264/AVC standard has been widely adopted by most recent video applications to answer the consumers' needs and the most demanding encoding requirements. The standard is divided in several profiles to define the applied encoding techniques, targeting specific classes of applications. For each profile, several levels are also defined, specifying upper bounds for the bit stream or lower bounds for the decoder capabilities, processing rate, memory size for multi-picture buffers, video rate, and motion vector range [9].

To achieve the offered encoding performance, this standard incorporates a set of new and powerful techniques: 4×4 integer transform, inter-prediction with variable block-size, quarter-pixel motion estimation (ME), in-loop deblocking filter, improved entropy coding based on Context-Adaptive Variable-Length Coding (CAVLC) or on Content-Adaptive Binary Arithmetic Coding (CABAC), new modes for intra prediction, etc. Moreover, the adoption of bi-predictive frames (B-frames) along with the previous features provides a considerable bit-rate reduction with negligible quality losses. As a result, when compared with previous standards (such as MPEG-2), the H.264/AVC has proved to provide greater coding efficiency levels, with an excellent tradeoff between the output video quality and bit-rate.

However, the simultaneous exploitation of those new features significantly increased the encoder's computational cost, thus reducing the application domains where it can be used. As an example, a direct execution of a straight compilation of the JM reference software in a latest generation processor (running at 2.7 GHz), leads to frame-rate performance levels as low as a single or at most a couple of 4CIF frames per second. In fact, several complexity analysis have shown that the *Inter prediction* block is usually the main time consuming - about 80% - followed by the *Interpolation* process, which uses between 15% and 7% [1]. To account for this problem, several approaches have been adopted, such as the application of new low complexity ME algorithms that have been studied and developed [5], dedicated hardware (HW) structures and, more recently, multi-processor solutions. Nevertheless, the innumerous data dependencies imposed by this video standard frequently inflict a very difficult challenge in order to efficiently take advantage of the several possible parallelization strategies that may be applied.

Up until now, most parallelization efforts around the H.264 standard have been mainly focused on the *decoder* implementation [2]. When the most challenging and rewarding goal of parallelizing the *encoder* is concerned, it has been observed that a significant part of the efforts have been devised in the design of specialized and dedicated systems [7, 6]. Most of these approaches are based on parallel or pipeline topologies, using dedicated HW structures to implement several parts of the encoder. When only pure software (SW) approaches are considered, fewer parallel solutions have been proposed. Most of them are based on the exploitation of the data independency between Group-of-Pictures (GOPs) or slices. However, since such implementations are often ful-

NOSSDAV'10, June 2–4, 2010, Amsterdam, The Netherlands.

filled with multi-threads [4, 3] or even with clustering of independent computers [8], they can hardly provide real-time streaming of the encoded video, since not only great amounts of memory would be required to accommodate the several encoded sub-streams, but a significant delay would be introduced in the resulting encoding procedure.

Meanwhile, with the recent advent of multi-core processors, further levels of parallelism are worth exploiting, by using the increased computational facility provided by the set of CPUs sharing the same chip. However, the usage of such powerful platforms has been often limited by the absence of a parallel encoding framework that easily adapts to and efficiently exploits the set of variable resources offered by such concurrent platforms. In this scope, a flexible and highly modular multi-core parallel programming platform for H.264/AVC software encoders is now proposed. The aimed challenge is to provide a linear speedup using several identical processors, without sacrificing the output video quality or increasing the bit-rate. The conducted assessments, using a wide set of different instantiations of the platform, have shown that linear and close to optimal speedup values, in what concerns the achieved frame-rate, can be obtained. Moreover, the provided modularity and flexibility allows an easy reconfiguration in order to better adapt it to targeted multi-core system.

2. TARGET MULTI-CORE SYSTEMS

Moore's law describes a long-term tendency in the history of computing HW, where the chip transistor density doubles approximately every two years. This law has been synonym of both performance and speed: chip complexity grew up and so they became capable of doing more complex functions; through miniaturization, clocks have become faster, leading to CPUs into the GHz range. However, physical and technological limits have confined the amount of performance improvement achievable by single-processors. Chip transistor density is limited by heat and electromagnetic interference, and even when these problems are solved, processor speeds will be always constrained by the speed of light. These issues led to the most feasible way to improve the processor performance: developing multi-core systems, in which computational load is distributed among several processors.

These new architectures can be classified by the adopted strategy to share information among the CPUs. In Uniform Memory Access (UMA) architectures, the main memory or pull of memories is physically shared uniformly by all CPUs. Therefore, access time is independent regardless the processor or memory address. On the other hand, in Non-Uniform Memory Access (NUMA) architectures, such as clusters, the access time is memory and processor dependent.

In what concerns the data, it can be classified as private or shared. While private data may only be accessed by a single processor, shared data can be used by all cores, providing an easy communication means among the processors. However, with the introduction of cache subsystems, several coherency problems often arise around the shared data, since multiple copies can simultaneously exist in different caches, creating inconsistent data views. To solve these problems, HW protocols and control mechanisms have been implemented. Furthermore, synchronization schemes must also be introduced to control the access to shared data and resources. Although these mechanisms assure the correct program execution, they provide it at the cost of processing time.

To take full advantage of multi-core systems, parallel programming SW is usually developed using either the *process fork model*, where the subroutines are assigned to processes that are distributed by the several cores, or the *thread fork model*, where subroutines are organized in threads. The first model is mainly used in cluster systems while the second has been widely adopted in multi-core computers, due to its lightweight. As a consequence, the developed parallel platform was implemented using the thread fork model, due to its easier implementation and established support by most multi-core systems.

Two main interfaces are usually available to develop SW based on this model: POSIX threads (Pthreads) and OpenMP. When POSIX threads are considered, a careful development has to be conducted, since threads creation has to be done using special and dedicated functions and the parallel regions should be taken out-of-line and put into a separate subroutine. Moreover, data structures containing private information must be explicitly defined in separate. Also, accessing shared data is often assured through a pointer that is passed to each thread. As a result, huge code transformations frequently have to be done when parallelizing a fully sequential program. In contrast, when OpenMP is used, parallel regions are defined using the `#pragma` keyword, reducing the parallelization complexity. By default, all variables, other than those specified in a private clause, are shared, providing more transparency and an easier implementation. Furthermore, expedite methods are available to instantiate the work in the CPUs using a schedule clause.

Theoretically, the speedup provided by parallel systems would be linear - in an N-core system, a program with N tasks would have a runtime of $\frac{1}{N}$ or a speedup of N. However, in practice very few programs can achieve a near-linear speedup. Synchronization, data coherency, bus bandwidth, etc. are the main constrains that often establish an upper limit to the real speedup. Moreover, programs often have code that can only be processed sequentially. As a consequence, the theoretical speedup on a parallel computing platform is given by Amdahl's law, which defines the maximum speedup S that can be obtained through optimization:

$$S = \frac{1}{(1-F) + \frac{F}{S_p}} \quad (1)$$

where F is the parallelized fraction of the code and S_p is the partial speedup in this parallel fraction.

3. OPEN PLATFORM FOR DEVELOPING PARALLEL H.264 VIDEO ENCODERS

Several analyses and parallel optimizations have been presented about H.264/AVC encoders [3, 4, 8]. Due to the encoder's nature, many of these parallelization approaches exploit concurrent execution at: *frame-level*, *slice-level*, *macroblock-level*. However, careful design methodologies in what concerns the parameterization and modularity have to be considered, in order to avoid the introduction of performance losses in terms of the final bit-rate and PSNR.

At *frame-level*, the input video stream is divided in GOPs. Since GOPs are usually made independent from each other, it is possible to develop a parallel architecture where a controller is in charge of distributing the GOPs among the available cores (Fig. 1). The advantages of this model are clear: PSNR and bit-rate do not change and it is easy to

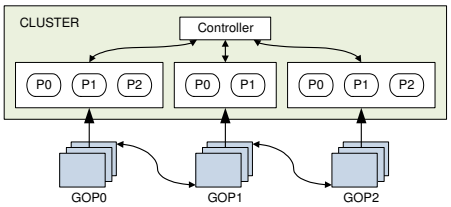

Figure 1: Frame-level parallelism in a cluster system.

implement, since GOPs' independency is assured with minimal changes in the code. However, the memory consumption significantly increases, since each encoder must have its own Decoded Picture Buffer (DPB), where all GOP's references are stored. Moreover, real-time encoding is hardly implemented using this approach, making it more suitable for video storage purposes. Consequently, this solution is mainly used in cluster systems. Further parallelism levels have to be exploited in order to improve even more the speedup [8].

In *slice-level* parallelism (Fig. 2), frames are divided in several independent slices, making the processing of macroblocks from different slices completely independent. In the H.264 standard, a maximum of sixteen slices are allowed in each frame. This approach allows to exploit parallelism at a finer granularity, which is suitable, for example, for multi-core computers. As an example, in multi-core architectures, parallel encoding of the several defined slices may be concurrently executed in multiple threads for each individual frame. Moreover, the increment in memory usage is smaller, because only one DPB is required. The main issues of this solution are: the limited number of slices per frame (sixteen in the H.264 standard); a greater effort has also to be done to ensure a good parallel performance; redesigned structures and algorithms simplifications are required in order to avoid caching of unnecessary data.

The parallelism at *macroblock-level* implies that independent MBs can be encoded at the same time [2]. According to the standard, a given MB is predicted using its left and upper three neighbors, which can be performed by following a wave-front approach, as depicted in Fig. 3. Since the encoding is performed in raster order, any two MBs are said to be *independent* if there isn't any data dependency in their prediction model. This strategy is used to further exploit the parallelism. The main design issues of this approach are: a centralized control is needed, to guarantee that only independent MBs are processed in parallel; and the computational weight may not be uniformly distributed among the cores. However, in middle and high resolution video sequences, as well as when a great number of processors is available, this model may be preferable to the slice-level, since the parallelism is only limited to $\lceil N/2 \rceil$, where N denotes the diagonal of the frame/slice (see Fig. 3).

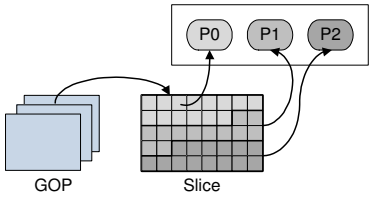

Figure 2: Slice-level parallelism in a multi-core system.

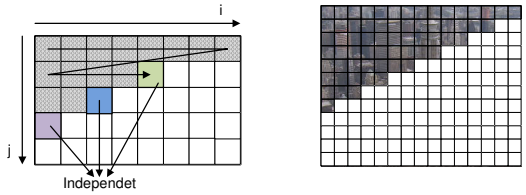

Figure 3: Exploitation of MB parallelism level.

The developed platform for implementing parallel H.264 video encoders is based on version JM14.0 of the reference software available at `http://iphome.hhi.de`, thus maintaining full compliancy with the original encoder. In order to achieve an efficient parallel execution, the conducted research was focused on: i) code profiling; ii) performance improvement through structures redesign and code optimization; iii) definition of the concurrent modules set; iv) parallelization. In the first step, code profiling was extensively performed, in order to identify the most time consuming operations. Several 4CIF test video sequences with distinct characteristics were used for more accurate results. As expected, *Inter Prediction* is the heaviest component of the encoder, making it the must suited target of the parallelization. Steps ii) and iii) are also independent of the parallel architecture that will support the execution, while the last step depends on the characteristics of such architecture.

An important step to improve the SW performance was the redesign of the data structures. In fact, appropriate data structures provide more efficient ways to manipulate and correlate information, without spending much time acquiring data. Hence, the main goal of this step was the proper reorganization of data in order to efficiently exploit the cache access patterns: spatial locality and temporal locality. In fact, by appropriately resizing the data structures, spatial locality can be further exploited, since the probability to store the whole structure in cache is higher. As a result, compulsory misses are more probably followed by hits. Similarly, temporal locality is also accomplished by data resize (since cache conflicts are reduced) and by joining together the information needed to process wide data sets in the several modules of the encoder. Such resizing was done by removing non-used or duplicated parameters and by adjusting their size to their effective range. Furthermore, all data structures were statically allocated in memory, allowing this platform to be easily executed in embedded systems that do not necessarily include an operating system. Besides the original data structures already present at the original reference SW, other new structures were also included, in order to support the programming of the provided MB parallelism level.

Besides the redesign of the data structures, some code improvements also had to be performed: the modules of the encoder were re-organized and optimized through reutilization of common functions and careful cleaning of the code. Moreover, suitable initialization functions were also added to each module, so that all the parameters needed by each block are gathered, pre-computed and stored in local structures, thus avoiding wasting time during video encoding. As a simple example, during the encoding some dedicated functions are used to find all the neighbors of a certain MB. Since MBs inter-relations do not change along the time, it is possible to store them in the form of addresses, instead of executing these operations to every frame.

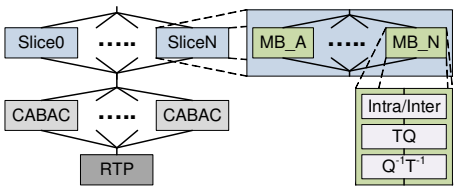

Figure 4: Set of concurrent functional modules of the implemented framework.

The organization of the several functional modules that integrate the proposed platform is depicted in Fig. 4. The set of processing modules chosen to be executed in parallel in macroblock-level are: Intra/Inter Prediction, Transformation & Quantization (TQ) and Inverse Quantization & Transformation($Q^{-1}T^{-1}$). The entropic encoder modules are executed at slice-level, to avoid memory bottlenecks and the usage of shared data. The modularity and flexibility provided by this platform allows an easy customization in order to suit it to a large number of multi-computers, multi-core and embedded systems. Such customization to the target architecture can easily be done during source compilation (through the supplied Makefile) by properly choosing the most suitable options to the target system. Also, due to the performed code simplification and division of the encoder into functional modules, the end user can easily add, remove and modify the sources with minimum effort.

To proof the concept, as well as the advantages provided by the developed platform, several parallel instantiations based on multiple cores were considered. As such, *slice-level* and *macroblock-level* parallelizations were extensively exploited. With *slice-level*, the several slices were assigned to a group of cores, where each core is responsible to encode a given slice. Due to the limitation of a maximum of sixteen slices (imposed by the H.264 standard), MB parallelization level was also used. With *macroblock parallelization level*, the cores from the same group work together in order to speedup the encoding of the same slice, as depicted in Fig. 5.

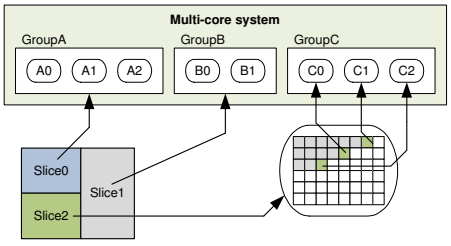

Figure 5: Simultaneous use of slice and macroblock parallelism levels.

4. PROGRAMMING AND ASSESSING PARALLEL H.264 VIDEO ENCODERS

To efficiently exploit the execution of parallel tasks in H.264/AVC encoders, several coding dependencies had to be broken. As an example, H.264 introduced three MB encoding models using rate distortion optimization (RD-Opt): simplified, complex and with losses. In the *simplified* model, the MB prediction mode is computed using only the prediction error (difference between prediction MB and current MB). In *complex* and *lossy* models, the prediction error is computed by considering the number of bits that are required to entropy encode in a certain mode. Hence, only after entropic encoding had been performed can this value be computed. This block dependency significantly slows down the encoder, making it hard to parallelize.

Among the several parallel programming APIs that could be adopted to implement the proposal platform, OpenMP offered particularly well suited characteristics to exploit the *slice-level* parallelism, since this API allows an easy and simple way of assigning parallel tasks to threads. The partitioning into slices and concurrency organization is automatically performed by the implemented platform, depending on the number of available cores. To further increase concurrency, *macroblock-level* parallelization was also added to the platform, by adopting nested threads. As soon as the slices are assigned to threads, these threads are responsible to create other threads, in order to form *teams of threads*. In each team, *independent* MBs are distributed among the remaining available cores for Intra or Inter processing. At the end, entropy coding is performed separately and at *slice level*.

The synchronization between the several concurrent threads is guaranteed using appropriate synchronization barriers. Only threads belonging to the same team are blocked in these barriers. This mechanism is used before and after the execution of the control mechanism that assigns the set of MBs processed by each slice (Fig. 6). While the first barrier guarantees that all threads have finished their tasks, the last barrier assures the correct MB assignment, since only after the team master thread has executed the procedure that controls and assigns the set of MBs that will be processed in the next run, can the remaining threads of the team start executing. No further synchronization is needed, since all the threads encounter a synchronization barrier at the end of the **#pragma** construct.

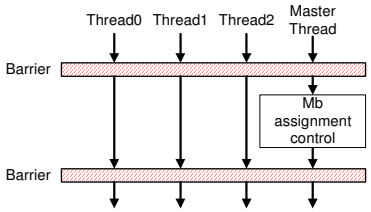

Figure 6: Thread synchronization mechanism.

In the remaining of this section, the developed parallel encoding platform is compared with the reference software. All the experimental results were obtained by considering the following encoding parameters:

- GOP structure: one I frame followed by thirty B-B-P frames;
- Intra prediction: all modes available;
- Inter prediction: all prediction modes available;
- Reference frames: a maximum of 3 backward references and 1 forward reference;
- ME search algorithm: simplified UMHexa;
- ME precision: quarter-pixel precision;
- ME error metric: SAD;
- Entropy coding: CABAC;
- In-loop deblocking filter: enabled.

The performed evaluations considered the *Soccer*, *Harbour*, *City* and *Crew* video sequences in 4CIF format (see Fig. 7). While *Soccer* and *Crew* are characterized by higher

(a) Soccer. (b) Harbour. (c) City. (d) Crew.

Figure 7: Used test video sequences.

amounts of movement, the *Harbour* and *City* can be classified as sequences with a higher spatial detail.

Table 1 presents the relative time consumption obtained from the conducted profiling analysis using `gprof`. It clearly shows that *Inter Prediction*, which includes motion estimation and motion compensation functions, represents the most computational demanding block. Alternative and more efficient ME algorithms could reduce its relative weight. Likewise, the usage of fewer Inter Prediction modes is a possible alternative to balance the computational effort.

When the memory resources of a particular instantiation of the proposed H.264 parallel platform are compared with those of the reference (original) software, the result of the extensive code improvements that were conducted becomes clear, leading to a global memory usage reduction of about 77% (see Table 2). Moreover, such optimizations provide a baseline speedup by a factor of 2, even without the exploitation of any level of parallelism.

Such an optimized H.264/AVC parallel encoder (obtained with the proposed platform) was run on a 32-core NUMA computational architecture, with eight AMD 8384 quad-core chip processors running at 2.7GHz. In each chip processor, all four cores have an individual cache L1 of 512KB and share a L2 cache with 6MB. The total amount of memory available in the system is 64GB, divided in memory pools for each chip processor. The operating system is Linux Ubunto 4.3.2.

Table 1: Profiling results of the H.264/AVC reference software obtained with `gprof`.

Function	Video Sequences (4CIF)			
	Soccer	Harbour	Crew	City
Inter Prediction	89.1%	86.0%	88.4%	87.7%
Intra Prediction	0.9%	1.1%	0.9%	1.1%
Transf. & Quant.	1.4%	1.7%	1.6%	1.7%
Interpolation	2.3%	2.9%	2.4%	2.7%
Filter	0.4%	0.7%	0.5%	0.5%
CABAC	0.4%	0.8%	0.6%	0.3%
Others	6.9%	8.5%	7.2%	7.8%

Table 2: Comparison of the memory resources required by the data structures in the original (reference) and optimized versions.

Structure	Ref.	Opt.	Mem. Saved(%)
Image Parameters	$82.4K$	232	99.7%
Input Parameters	$5.9K$	$6.1K$	−2.4%
PPS	248	144	41.9%
SSP	$2.1K$	$1.7K$	17.4%
Slice	$2.3M$	$2.3M$	0.0%
Macroblock	$171.7K$	$10.8K$	93.7%
DPB	$203.1M$	$42.6M$	79.0%
Total	$205.7M$	$46.3M$	77.5%

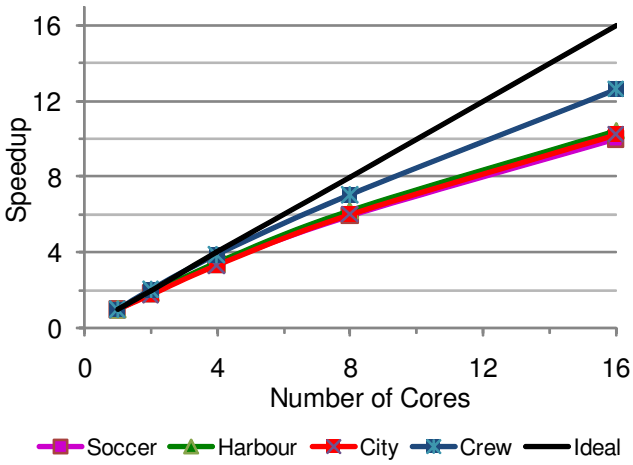

Figure 8: Provided speedup using slice-level parallelism.

The first test evaluated the performance provided by an isolated exploitation of the *slice-level* of parallelization. The obtained results, illustrated in Fig. 8, are very close to the theoretical optimum speedup in what concerns the achieved frame-rate, leading to a high processing efficiency. In part, the slices independency allows a parallelization level without data sharing between them. Therefore, the overheads of segmentation and data scheduling are avoided and the obtained encoder speedup is increased. However, since frames can be divided to a maximum of 16 slices in the H.264 standard, other levels of parallelization will have to be applied in order to avoid this constraint and make the solution, as well as the platform, more scalable.

Table 3 illustrates the performance provided by the isolated exploitation of the *macroblock-level* of parallelism. Contrasting with the results obtained with the *slice-level* model, the obtained speedup values are somewhat more modest, achieving a maximum value of about 4. Data sharing between the several cores of this particular NUMA multiprocessor is probably the main reason for these results: since each chip-processor has its own memory pool, data has to be shared, leading to a memory bottleneck and higher latency times. Nevertheless, this model offers a somewhat wider scalability space, entirely compatible with a mutual and simultaneous exploitation of the *slice-level* model. In the event of an UMA parallel architecture had been adopted, the obtained speedups would have been certainly greater.

The simultaneous usage of *slice* and *macroblock* parallelism levels was also validated using a set of configurations characterized by a fixed amount of cores (32). The results

Table 3: Provided speedup using macroblock-level parallelism.

#MBs	Video Sequences (4CIF)			
	Soccer	Harbour	City	Crew
1	1.0	1.0	1.0	1.0
2	1.6	1.6	1.6	1.6
4	2.5	2.5	2.6	2.5
8	3.2	3.0	3.2	3.1
16	3.6	3.5	3.7	3.6
32	3.7	3.8	3.7	3.9

Table 4: Provided speedup with slice and macroblock parallelism, using the whole set of 32 cores.

#Slices	#MBs	Video Sequences (4CIF)			
		Soccer	Harbour	City	Crew
1	32	3.7	3.8	3.7	3.9
2	16	5.0	4.7	4.7	5.3
4	8	6.8	6.4	6.4	7.4
8	4	5.5	5.3	5.3	6.0
16	2	2.7	2.8	2.5	2.8

presented in Table 4 illustrate the best combinations between *slice* and *macroblock* parallelism levels. When compared with the previous results, it can be observed that further speedup levels can be obtained using the combination of the two levels of parallelism. It was also observed that when more than 4 slices are used, communication and data sharing constraints, introduced by macroblock parallelism, are responsible for the observed decrease of the speedup. Nevertheless, all the obtained results denote the achievement of speedup performances that are considerably higher than those that have been obtained by other proposals (such as [4], whose speedup did not exceeded 4.5).

Finally, Fig. 9 presents the speedup results corresponding to the best evaluated configurations of the developed platform. As it was expected from the previous results, the speedup increase through *slice* parallelism level (#slice) is only possible by decreasing the number of macroblocks concurrently processed (#MBs). As a result, threads are less time idle (in synchronization barriers), thus increasing the core efficiency.

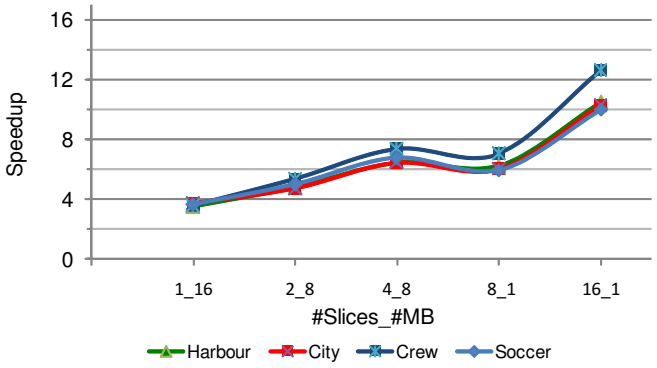

Figure 9: Maximum speedup achieved for slice and macroblock levels of parallelism.

5. CONCLUSIONS

A new open software platform to implement parallel H.264/AVC encoders was presented, in order to exploit the parallelism provided by multi-core architectures, as an efficient means to increase the performance. Two parallelism levels are exploited: *slice* level and *macroblock* level. Despite the limitations imposed by the H.264 standard, slice level parallelism proved to be the most efficient approach, with speedup gains quite close to the theoretical maximum. On the other hand, *macroblock* parallelism level has shown to offer a lower parallel efficiency. Nevertheless, it can still be used to further increase the speedup in UMA architectures when the number of available cores is greater than 16.

There is still a considerable amount of future work that can be conducted. Most of the microprocessors available today support Single Instruction Multiple Data (SIMD) multimedia instruction set extentions to accelerate their execution. With them, multiple data can be processed in parallel in a single instruction. The usage of these instructions on an already task parallel encoder can surely provide even better results.

The developed source code of this platform, as well as the considered test video sequences, are publicly available at `p264` project webpage:
`http://sips.inesc-id.pt/prototypes.php#p264`.

6. ACKNOWLEDGMENTS

The authors would like to thank Dr. Alexandros Stamatakis, from the *The Exelixis Lab*, TU München - Germany, for granting access to the 32-core computational architecture adopted in the experimental procedure.

7. REFERENCES

[1] M. A. B. Ayed, A. Samet, and N. Masmoudi. H.264/AVC prediction modules complexity analysis. *European Trans. on Telecommunications*, 18:169–177, Aug. 2007.

[2] A. Azevedo, B. Juurlink, C. Meenderinck, A. Terechko, J. Hoogerbrugge, M. Alvarez, A. Ramirez, and M. Valero. A highly scalable parallel implementation of H.264. *Trans. on High-Performance Embedded Architectures and Compilers (HiPEAC)*, Sep. 2009.

[3] Y.-K. Chen, E. Q. Li, X. Zhou, and S. L. Ge. Implementation of H.264 encoder and decoder on personal computers. *Journal of Visual Communications and Image Representations*, 17(2):509–532, Apr. 2006.

[4] Y.-K. Chen, X. Tian, S. Ge, and M. Girkar. Towards efficient multi-level threading of H.264 encoder on Intel hyper-threading architectures. In *Proc. of Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 63–72, Apr. 2004.

[5] S. Hiratsuka, S. Goto, and T. Ikenaga. An ultra-low complexity motion estimation algorithm and its implementation of specific processor. In *Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS)*, May 2006.

[6] S. Kim and M. Sunwoo. ASIP approach for implementation of H.264/AVC. *Journal of Signal Processing Systems*, 50(1):53–67, Jan. 2008.

[7] H. Mizosoe, D. Yoshida, and T. Nakamura. A single chip H.264/AVC HDTV encoder/decoder/transcoder system LSI. *IEEE Transactions on Consumer Electronics*, 53(2):630–635, May 2007.

[8] A. Rodríguez, A. González, and M. Malumbres. Hierarchical parallelization of an H.264/AVC video encoder. In *Int. Conf. on Parallel Computing in Electrical Engineering (PARLEC)*, pages 363–368, 2006.

[9] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *Trans. on Circuits and Systems for Video Technology*, 13:560–576, Jul. 2003.