Scalability of Macroblock-level Parallelism for H.264 Decoding

Mauricio Alvarez Mesa*, Alex Ramírez*†, Arnaldo Azevedo‡, Cor Meenderinck‡, Ben Juurlink‡, and Mateo Valero*†
*Universitat Politècnica de Catalunya. Barcelona. Spain
Email: alvarez@ac.upc.edu
†Barcelona Supercomputing Center. Barcelona. Spain.
Email: alex.ramirez@bsc.es, mateo.valero@bsc.es
‡Delft University of Technology. Delft. The Netherlands.
Email: azevedo@ce.et.tudelft.nl, cor@ce.et.tudelft.nl, B.H.H.Juurlink@tudelft.nl

***Abstract*—This paper investigates the scalability of MacroBlock (MB) level parallelization of the H.264 decoder for High Definition (HD) applications. The study includes three parts. First, a formal model for predicting the maximum performance that can be obtained taking into account variable processing time of tasks and thread synchronization overhead. Second, an implementation on a real multiprocessor architecture including a comparison of different scheduling strategies and a profiling analysis for identifying the performance bottlenecks. Finally, a trace-driven simulation methodology has been used for identifying the opportunities of acceleration for removing the main bottlenecks. It includes the acceleration potential for the entropy decoding stage and thread synchronization and scheduling. Our study presents a quantitative analysis of the main bottlenecks of the application and estimates the acceleration levels that are required to make the MB-level parallel decoder scalable.**

***Keywords*-Video CODEC Parallelization, H.264/AVC, Multicores, Chip-multiprocessors, parallel scalability.**

I. Introduction

The trends in the video coding application domain are toward systems with higher levels of quality and at the same time with a high compression efficiency [1]. The trend towards high quality systems has pushed the adoption of High Definition (HD) digital video and even higher definitions are being proposed. To provide higher compression efficiency without sacrificing quality advanced video codecs like H.264 and VC-1 have been developed [2]. The combination of the complexity of these video codecs and the higher quality of HD systems has resulted in an important increase in the computational requirements of the emerging video applications [3].

At the same time, there is a paradigm shift in computer architecture towards chip multiprocessors (CMPs) due to the scalability limits of single core processors. As a consequence, it is expected that the number of cores on a CMP will double every processor generation, resulting in hundreds of cores per die in the near future [4]. An important question is whether video coding applications can benefit from the performance offered by CMP architectures. As a result, an important research effort has been made in the last years for developing techniques for parallelization of codecs like H.264. One of the most promising techniques is the parallelization at the level of MacroBlocks (MBs) [5], [6], [7]. This type of parallelization has been presented as scalable and efficient, but most of the analysis have been made using simplified theoretical or simulation models, or have been based on real executions with low definition videos and a small number of cores. The primary aim of this paper is to provide a deeper understanding of the scalability of MB-level parallelism on multicore architectures for HD applications. Our objective is to identify the main bottlenecks and to evaluate the impact of acceleration for removing them.

Scalability has been analysed from different perspectives. First, we have enhanced a formal model that take into account the variable processing time of the inner kernels and the overhead of thread synchronization for estimating the upper limits of the parallelization. Second, we have compared this model with an implementation on a cache coherent Non-Uniform Memory Access (cc-NUMA) Shared Memory Multiprocessor (SMP). The implementation study includes the analysis of different scheduling algorithms and the identification of bottlenecks and sources of overhead. And, finally, we have used a trace-driven simulation approach for analysing the potential of acceleration for removing the bottlenecks that inhibit to obtain the full potential performance. As new bottlenecks become exposed larger fractions of the original code requires optimization and become candidates for acceleration [8].

The paper is organized as follows. First, in section II we present an introduction to H.264 and the parallelization strategy. Second, in section III we present the formal model and abstract simulation results. Third, in section IV we present the analysis of the implementation on the cc-NUMA machine. Fourth, in section V we present the results for the acceleration study. Fifth, in section VI we discuss the related work on the field. And finally, in section VII we present the conclusions and future work.

II. Parallelization of H.264 decoder

H.264 is based on the same block-based motion compensation and transform-based coding framework of prior

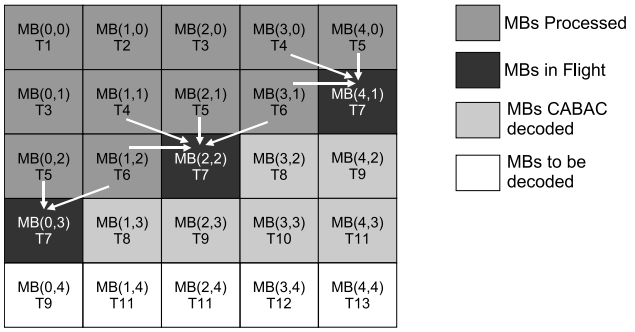

Figure 1. MacroBlock-level parallelism inside a frame.

MPEG video coding standards, but it provides higher coding efficiency through added features and functionality that, in turn, entail additional complexity. The higher coding efficiency and quality come from the new coding tools included, like: variable block-size motion compensation, multiple reference frames with weighted prediction, fractional (1/2, 1/4) motion compensation, integer and adaptive DCT-like transform, adaptive deblocking filter, context adaptive arithmetic coding (CABAC), and others [2].

In H.264/AVC (as in other hybrid video codecs) a video sequence consist of multiples video pictures called frames. Each frame can consist of several slices, which are self contained partitions of a frame, that, in turn, contain some number of MacroBlocks (MBs). MBs, which are blocks of 16×16 pixels, are the basic data unit for coding and decoding. The main computing kernels are applied at the MB level, although the standard allows some kernels to operate on smaller blocks. Main kernels are Prediction (intra prediction or motion estimation), Discrete Cosine Transform (DCT), Quantization, Deblocking filter, and Entropy Decoding.

As a result of the of the increased computational requirements current high performance uniprocessor architectures are not capable of providing the performance required for real-time HD processing [3] and, therefore, it is necessary to exploit thread level parallelism. Among different approaches MB-level parallelization has been proposed as a scalable technique for the H.264 decoder [5], [9], [7], [10], [11]. It can scale to a large number of processors without depending on coding options of the input videos and without affecting the latency of the decoding process. (Only Macroblock-level parallelism is described in this work; a discussion of the other levels can be found in [7]).

In H.264, usually MBs in a frame are processed in scan order, which means starting from the top left corner of the frame and moving to the right, row after row. To exploit parallelism between MBs inside a frame it is necessary to take into account the dependencies between them. In H.264, motion vector prediction, intra prediction, and the deblocking filter use data from neighbouring MBs defining a structured set of dependencies. Processing MBs in a diagonal wavefront manner satisfies all the dependencies and, at the same time, allows to exploit parallelism between MBs (as shown in Figure 1). We refer to this parallelization technique as 2D-Wave [5].

It's important to note that due to the sequential behaviour of the entropy decoding kernel it should be decoupled from the MB reconstruction process.

III. Theoretical Analysis

We can represent the processing of MBs in H.264 decoding as a Directed Acyclic Graph (DAG). Each node in the DAG represents the decoding of one MB by one processor. The decoding of each MB consists of a sequential ordering of kernels applied to some input data. Edges in the graph represent the data dependencies between MBs. Figure 2 shows the DAG for a $5x5$ MBs sample frame. Each frame in a video sequence can be represented with a finite DAG. The first MB in the frame is the *source node* which has no incoming edges and the last MB in the frame is the *sink node* which has not outgoing edges. We define the *depth* as the length of the longest path from the source node to the sink node. For a finite DAG G representing a frame F we define the computational work T_s as the number of nodes in G, and T_∞ as the depth of G. Although the structure of the dependencies is known the actual shape of the DAG is input dependent and cannot be known before the processing of all nodes.

A. Theoretical Maximum Speed-up

Assuming that the time to process each node in the DAG is constant and that there is no overhead for thread synchronization then we can estimate the theoretical maximum speedup. Let mb_width and mb_height be the width and height of the frame in macroblocks respectively. Then, $T_s = mb\_width * mb\_height$ and $T_\infty = mb\_width + (mb\_height - 1) * 2)$. The maximum speedup (MSU) is defined as:

$$MSU = \frac{mb\_width * mb\_height}{mb\_width + (mb\_height - 1) * 2} \quad (1)$$

Taken that into account, we can calculate the maximum number of processors (MP) as:

$$MP = round\left(\frac{mb\_width + 1}{2}\right) \quad (2)$$

In Table I, these values are shown for different video resolutions. For FHD resolution the theoretical maximum speedup is 32.13 when using 60 processors.

B. Abstract Trace-driven Simulation

The theoretical maximum speedup is based on the assumption that MB processing time is constant and there is not thread synchronization overhead. Both assumptions are not true in real applications. On one hand, although the same

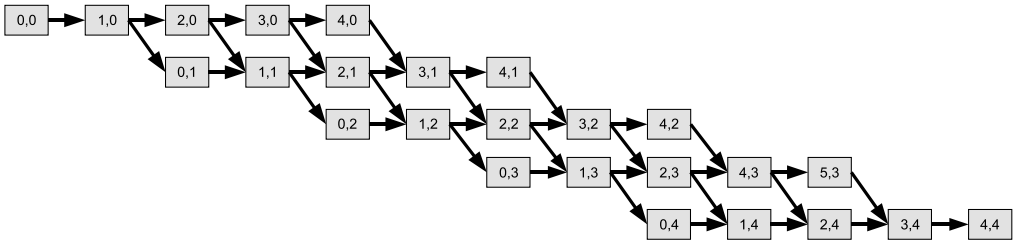

Figure 2. Directed Acyclic Graph (DAG) of MacroBlocks.

Video Resolution	Pixel Resolution	MB Resolution	T_s	T_∞	Max. Speedup	Max. processors
Standard (SD)	720x576	45x36	1620	115	14.09	23
High (HD)	1280x720	80x45	3600	168	21.43	40
Full High (FHD)	1920x1080	120x80	8160	254	32.13	60
Quad Full High (QFHD)	3840x2160	240x160	32400	508	63.78	120
Ultra High (UHD)	7680x4320	480x320	129600	1018	127.31	240

Table I
THEORETICAL MAXIMUM SPEEDUP FOR DIFFERENT VIDEO RESOLUTIONS.

set of filters are applied to each MB, the processing time is input dependent because the exact operations that are applied to the image samples depend on conditions of those samples. On the other hand, thread synchronization overhead is not negligible. Every time a MB is processed a table of dependencies should be updated and some scheduling decision has to be taken. Those steps require the synchronization of parallel threads.

In order to analyse the effects of those conditions we have build an abstract MB trace-driven simulator which creates the DAG for each frame and then calculates the Task Processing Time (TPT) of every node as:

$$TPT(n) = w_n + s_n + MAX\left(TFT(pr_n)\right) \qquad (3)$$

Where, w_n is the time required to process the task, s_n is the time required for thread synchronization; and $MAX(TFT(pr_n)$ is the maximum task finish time (TFT) of the immediate predecessors tasks of that task. When the DAG has been fully processed we take the data from the *end node* and its finish time represents the best time that we can achieve from the parallel execution of that DAG. Because this is input dependent we have analysed the DAGs for different frames and different input videos at FHD.

C. Effects of variable decoding time

Table II shows the speedup of the parallel execution for different input videos. It includes the maximum theoretical speedup and the maximum speedup taking into account the variable processing time. In average for all the input videos the speedup is reduced a 33 percent compared to the theoretical maximum. The values presented in this table are average per frame, because the actual performance changes from frame to frame due to the differences in input content and type of MBs.

Input Video	speedup const. time	speedup var. time	slow-down
Blue_sky	32.13	19.22	0.40
Pedestrian_area	32.13	21.92	0.31
Riverbed	32.13	24.01	0.25
Rush_hour	32.13	22.22	0.30

Table II
MAXIMUM SPEEDUP TAKING INTO ACCOUNT VARIABLE DECODING TIME.

D. Effects of Thread Synchronization Overhead

We have modelled the synchronization overhead as an extra time for MB decoding. The base value for the overhead is the average processing time of each MB in a frame. Figure 3 shows the average speedup for each video sequence. A zero value represents the maximum speedup taking into account the variable processing time. As the value of overhead increases the speedup decreases correspondingly. For example, consider the 1088p25_blue_sky video sequence: with zero synchronization overhead the maximum speedup is 19.23. Adding a synchronization overhead of 1 the speedup reduces to 11.93 (38%). By using these data a system designer can decide when thread synchronization optimizations are useful in terms of the cost to design and implement them compared to the benefit in speedup. Although synchronization overhead values bigger than the processing time may seem unreasonable we have found in our experiments values up to 12 times the average MB decoding time.

IV. PERFORMANCE ANALYSIS ON THE CC-NUMA ARCHITECTURE

Our implementation is based on a dynamic task model using task pools. In this model, a set of threads is activated when a parallel region is encountered. In our case a parallel

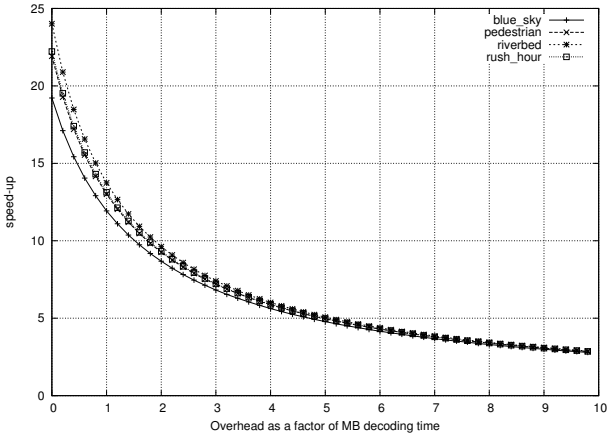

Figure 3. Effect of thread synchronization on final performance

region is the decoding of all MBs in a frame. Each parallel region is controlled by a frame manager, which consist of a thread pool, a task queue, a dependence table and a control thread.

The thread pool consists of a group of worker threads that wait for work on the task queue [12]. The dependencies of each MB are expressed in a dependence table. When all the dependencies for a MB are resolved a new task is inserted on the task queue. The control thread is responsible for handling all the initialization and finalization tasks that are not parallelizable. Synchronization between threads and the access to the task pool were implemented using POSIX threads (Pthreads) and real-time semaphores. Both synchronization objects are blocking, which means that the operating system is responsible for the activation of threads. The access to the table of dependencies was implemented with atomic instructions like *dec_and_fetch*.

A. Evaluation Platform

For these experiments we have used a modified version of the FFmpeg H.264 decoder with FHD video inputs taken from HD-VideoBench [13]. The application was tested on a SGI Altix which is a shared memory machine, with a cc-NUMA architecture with 64 dual core IA-64 processors. Each one of the 128 cores works at 1,6 GHz, with a 8MB L3 cache and 533 MHz Bus, and the system has a total 512 GB RAM. The compiler used was gcc 4.1.0 and the operating system was Linux kernel version 2.6.16.27.

B. Scheduling Strategies

One of the main factors that affects the scalability of the 2D-wave parallelization is the allocation (or scheduling) of MBs to processors. We have evaluated three different scheduling algorithms: static scheduling, dynamic scheduling and dynamic scheduling with tail submit optimization. In Figure 4 the average speedup for the different scheduling approaches is presented. Speedup is calculated against the original sequential version and corresponds to the section of MB decoding (without CABAC).

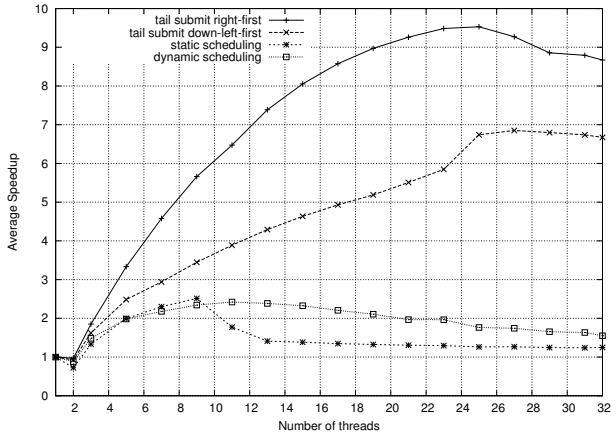

Figure 4. Speedup of MB decoding using different scheduling approaches.

C. Static scheduling

Static scheduling means that the decoding order of MBs is fixed and a master thread is responsible for sending MBs to the decoder threads. The predefined order is a zigzag scan order which can lead to an optimal schedule if MB processing time is constant. When the dependencies of an MB are not ready the master thread waits for them. Figure 4 shows the speedup of static scheduling. The maximum speedup reached is 2.51 when using 8 processors (efficiency of 31%). The low scalability is due to the fact that MB processing time is variable, and static scheduling results in load unbalance: most of the time the master thread is waiting for other threads to finish. This shows that MB-level parallelization requires a dynamic allocation of MBs to processors in order to be scalable.

D. Dynamic scheduling

In this scheme, worker threads take MBs from the task queue, process them, update the dependence table and, if that is the case, submit new MBs to the task queue. Production and consumption of MBs is made through the centralized task queue. Figure 4 shows the speedup for the dynamic scheduling. A maximum speedup of 2.42 is found when 10 processors are used (efficiency of 24%). This is lower than the maximum speedup for the static scheduling. Although the dynamic scheduling is able to discover more parallelism than static scheduling, the overhead for submitting MBs to (and getting MBs from) the task queue is so big that it jeopardizes the parallelization gains. Most of this overhead comes from the intervention of the OS in the scheduling process and for contention in the access to the queue.

In order to analyze the performance of worker threads we divided the execution of each one into the following six phases:

Threads	decode_mb	copy_mb	get_mb	update_mb	ready_mb	submit_mb	overhead-ratio
1t	22.65	1.62	4.89	1.01	2.38	5.03	0.67
4t	33.09	2.95	9.71	1.36	2.86	12.44	1.30
8t	41.88	3.90	16.67	1.61	3.02	20.84	2.05
16t	61.78	5.94	55.95	2.25	3.55	80.28	6.57
24t	58.08	5.15	105.03	2.09	3.49	120.37	10.49
32t	78.75	7.25	209.37	2.70	4.36	201.01	18.88

Table III
AVERAGE EXECUTION TIME FOR WORKER THREADS WITH DYNAMIC SCHEDULING (TIME IN US.)

- *get_mb*: Take one element from the task queue.
- *copy_mb*: Copy of entropy decoded parameters to the local thread structures.
- *decode_mb*: Actual work of MB decoding.
- *update_mb*: Update the table of MB dependencies.
- *ready_mb*: Analysis of new ready to process MB.
- *submit_mb*: Put one element into the task queue.

Table III shows the execution time of the different phases. It can be noted that the MB decoding time increases with the number of processors. This is mainly due to the fact that the dynamic scheduling algorithm does not consider data locality when assigning tasks to processors. When a processor takes a MB which has its data dependencies in a remote node, then all the memory accesses should cross the NUMA interconnection network. Other phases that exhibit a major increase in execution time are: *get_mb* and *submit_mb*. This reveals a contention problem because in dynamic scheduling all the worker threads get MBs from (and submit MBs to) a centralized task queue creating an important pressure on it. The last column of the table shows the ratio of actual computation and overhead. The overhead increases significantly when the number of processors goes beyond 8. From this, we can conclude that the centralized task queue becomes the bottleneck. A more distributed algorithm like tail submit [14] or work stealing [15] could help to reduce this contention.

E. Dynamic scheduling with Tail submit

As a way to reduce the contention on the task queue, the dynamic scheduling approach was enhanced with a tail submit optimization. With tail submit when a thread founds a ready to process MB it can process that MB directly without any further synchronization. If more than one MB is discovered, one is submitted to the task queue and the other one is processed directly [14]. There are two ordering options for doing the tail submit process: execute directly the right neighbor of the current MB and submit the other, or execute directly the down-left neighbor and submit the other. Figure 4 shows the speedup of tail-submit implementations. The down-left-first version achieves a maximum speedup of 6.85 with 26 processors (efficiency of 26%). The right-first version achieves a maximum speedup of is 9.53 with 24 processors (efficiency of 39.7%). The better scalability of the right-first order is due to the fact that it exploits the data locality between MBs. Data from the left block is required by the deblocking filter and by using the right-first order the values of the previous MB remain in the cache.

Table IV shows the profiling results for tail submit version with right-first order. In this case, MB decoding time remains almost constant with the number of threads due to the exploitation of the data locality between neighbor MBs. Another effect of the tail submit optimization is the reduction in the time spent in *submit_mb*. This time still increases with the number of processors but the absolute value is less than the dynamic scheduling version. With tail submit there is less contention because there are less submissions to the task queue as shown in the last column of Table IV. The most significant contributor to the execution time is *get_mb* indicating a lack of parallel MBs, meaning that the scalability limit of the tail submit version has been reached.

F. Impact of the Serial Part of the Application: The CABAC Bottleneck

In order to allow a parallel decode of MBs CABAC entropy decoding is decoupled from the MB decoding loop. The decoupling is done by using an intermediate buffer in which the CABAC decoder stores the decoded information for every MB. After finishing the CABAC decoding of a frame the decoder threads start to decode MBs in parallel. Because CABAC decoding cannot be parallelized at MB-level it should be executed sequentially in one processor. Then, according to the Amdahl's law it can become the limiting factor. Figure 5 shows the execution time of the application including CABAC time. The execution time of MB decoding (*hl_decode_mb*) reduces with the number of processors as a result of the parallel execution. But, the execution time associated with CABAC (*decode_cabac*) augments with the number of processors. This is a side effect of the shared-memory model and the coherence protocol. When a new frame is being processed the CABAC decoder should overwrite the values in the intermediate buffer and this generates cache invalidations that go out of the chip and cross all the interconnection network to reach the local caches that have these values.

V. REMOVING THE BOTTLENECKS WITH MULTICORE ACCELERATION

Without significant CABAC acceleration MB-level parallelization is useless, and without very fast synchronization

Threads	decode_mb	copy_mb	get_mb	update_dep	ready_mb	submit_mb	overhead ratio	% of tail submit
1t	21.7	1.5	6.1	1.0	1.0	7.5	0.17	90.8
4t	24.2	1.9	55.9	1.1	1.1	7.8	0.22	79.8
8t	24.9	2.1	132.4	1.3	1.1	8.6	0.30	75.2
16t	27.5	2.4	265.3	1.6	1.1	10.1	0.68	58.5
24t	30.6	2.9	683.7	1.9	1.2	24.6	1.00	51.4
32t	30.1	2.8	853.1	2.1	1.1	24.8	1.85	48.4

Table IV
AVERAGE EXECUTION TIME FOR WORKER THREADS WITH TAIL SUBMIT OPTIMIZATION (TIME IN US.)

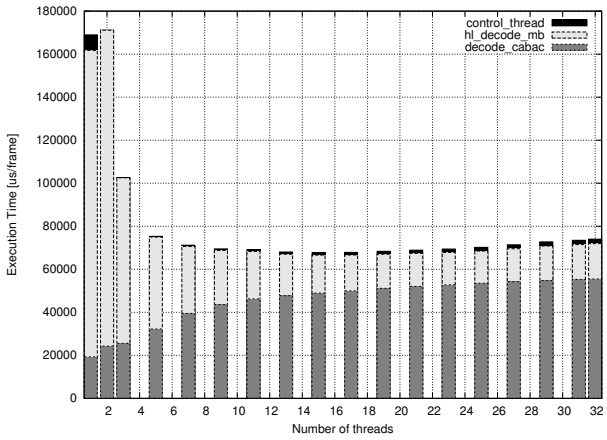

Figure 5. Execution time per frame including CABAC entropy decoding.

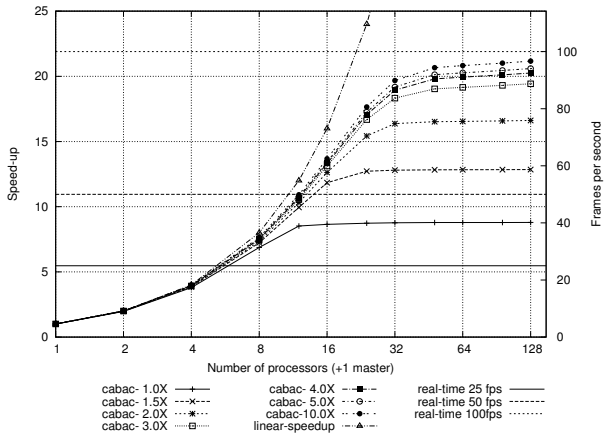

Figure 6. CABAC acceleration

and scheduling operations MB-level parallelization will not scale to a manycore system. Those limiting factors offer a potential for acceleration, being this in the form of special purpose units added to the base processor, computational offload units or separate special purpose processors [8]. In the next sections we are going to evaluate the effect of acceleration on the MB-level parallel H.264 decoder.

A. Fast multicore simulation using Tasksim

For this study we have collected traces from the parallel execution of the H.264 decoder on the Altix multiprocessor machine. Those traces contain CPU phases, synchronization events and memory operations. CPU phases are collections of instructions related to a portion of the program. The analysed phases of the H.264 decoder are the same that are mentioned in section IV-D. The simulation is done using a fast trace-driven simulator called Tasksim. It simulates CPU phases not instructions, which means that the duration of the CPU phases has to be taken from an execution trace. Apart, it simulates a synchronization network that supports the two basic operations of the semaphore semantics: *wait* and *signal*. Additionally, it simulates a memory hierarchy composed of local memories, shared on-chip L2 caches and an external DRAM memory.

B. Accelerating Entropy Decoding

There are some proposals in the literature with different approaches for CABAC acceleration [16], [17] that can be integrated in a multicore architecture. The question that remains open is what is the acceleration required on the CABAC engine in order to provide the required performance for a MB-level parallel decoder. In order to solve that question we made an experiment in which the time required to perform the CABAC decoding was accelerated by different ratios. The baseline is the execution of the CABAC decoding on the Itanium-II processor assuming no overhead for thread synchronization. Figure 6 shows the effect of CABAC acceleration. The curves for 25, 50 and 100 frames per second (fps) are included as a reference. When no acceleration is applied to CABAC a maximum speedup of 4.6 is reached using 16 processors. A maximum speedup of 23.5 (which is close to the theoretical maximum) is reached when CABAC is accelerated by a factor of 10. These results can help to choose the appropriate CABAC accelerator depending on the required performance of the final application. For example, the required CABAC acceleration for meeting different real-time requirements are: for 25fps 1X (no cabac acceleration) and 6 worker processors; for 50fps 1.5X acceleration and 14 worker processors; and for 100fps the 2Dwave parallelization is not enough independent of the CABAC acceleration. As a further step, acceleration

level can be dynamically adjusted by combining accelerators and frame-level parallelism for CABAC decoding.

C. Accelerating Synchronization and Scheduling

For showing the effects of accelerating the synchronization we have conducted an experiment in which we assign different durations to the synchronization operations (sync-ops) of the task pool ranging from 1ns to 100000 ns. Figure 7 shows the speedup for various durations of the sync-ops. As a reference, the figure also includes the speedup for *sync-altix-sw* version which corresponds to the implementation on the real machine with tail submit. Additionally, the figure includes results for software synchronization using the duration of the sync-ops of the parallel decoder with only one worker thread. This value can be seen as the maximum speedup that can be achieved with a software implementation of the task pool. According to the figure, at 16 processors, the real system achieves a speedup of 8.6, close to the 1000ns sync-ops, which corresponds to 40 fps. The best software approach obtains a speedup of 10.0 that corresponds to 46.5fps. In order to meet the real-time operation at 50fps, and using 16 worker processors, the sync-ops should be in the range of 100 to 500ns. This is consistent with published results of hardware supported schedulers [18], [19].

The results in Figure 7 show the trade-off between synchronization acceleration and number or processors (area). Using this information a system designer can decide if implementing highly sophisticated and low-latency hardware schedulers pays the cost of the reduction in the number of cores required to meet certain real-time performance target.

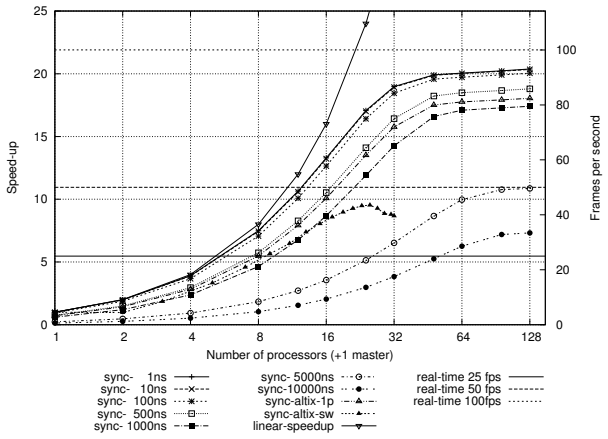

Figure 7. Acceleration of thread synchronization

VI. Related Work

Several papers deal with H.264 parallelization. Some works have presented results for functional parallelization [20]. Others present GOP, frame and slice-level parallelism [21], [6], [22]. This kind of coarse grain parallelization techniques are not scalable for multicore architectures.

MB-level parallelization for H.264 has been discussed in several works. Van der Tol et al. [5] proposed the technique but they did not present a scalability analysis. Chen et al. [9] evaluated an implementation on Pentium machines with a reduced number of processors and for low resolution videos. Hoogerbrugge et al. [14] have evaluated this scheme using a simulated embedded multicore architecture composed of VLIW media processors with a cache coherent memory organization. They do not take into account the effects of CABAC decoding and use an application specific hardware unit for thread synchronization. In our paper we presents an evaluation that includes an analysis of the scalability effects of CABAC and different levels of synchronization acceleration.

Other works have proposed techniques for combining temporal and spatial MB-level parallelism. Zhao et al. [10] studied an scheme for low resolution video encoding using a static scheduling scheme that results in poor load balancing and does not scale for multicores. Azevedo et al. [23] have evaluated a dynamic technique called 3D-wave and compared it with the 2D-wave and showed that the former is more scalable than the later. The analysis has been made on a simulation platform of embedded media processors. Those results are complementary to the presented in this paper because their architecture include sophisticated hardware support for thread synchronization and then then overhead of thread synchronization is minimal. This confirms the results from our paper in which we show the necessity of special support for thread synchronization.

VII. Conclusions

In this paper we have investigated the scalability of the macroblock-level parallelization of the H.264 decoder. A formal model and an abstract trace driven simulation were used to estimate the impact of variable decoding time and thread synchronization overhead on the maximum performance. Variability in processing time of tasks demand the use of dynamic load balancing techniques. And, the analysis of the thread synchronization allows to estimate the impact of optimizations in the synchronization infrastructure.

The implementation of the 2D-wave parallelization on the cc-NUMA machine shows that the best scheduling strategy is the combination of dynamic scheduling with tail submit. Dynamic scheduling deals with the unbalance that results from variable decoding time and tail-submit reduces the synchronization overhead and, at the same time, exploits data locality reducing the external memory pressure.

The study of acceleration impact shows on one hand the required performance of the CABAC accelerator. These demands of the CABAC accelerator depends on the resolution of the image and the input content. In order to cover these variations in run-time the performance of the CABAC

accelerator should be adjusted dynamically. This is an area of future work. On the other hand, the presented study shows the limits of software synchronization and presents the opportunities for hardware based schedulers. A comparison with current schemes shows that there is room for improvement in hardware acceleration of synchronization. This is part of our current work on the field.

ACKNOWLEDGMENT

This work has been supported by the European Commission in the context of the SARC project (contract no. 27648), and the Spanish Ministry of Education (contract no. TIN2007-60625).

REFERENCES

[1] T. Sikora, "Trends and Perspectives in Image and Video Coding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 6–17, Jan 2005.

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A.Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[3] M. Alvarez, E. Salami, A. Ramirez, and M. Valero, "A Performance Characterization of High Definition Digital Video Decoding Using H.264/AVC," in *IEEE International Symposium on Workload Characterization*, Oct 2005, pp. 24–33.

[4] S. Borkar, "Thousand core chips: a technology perspective," in *DAC '07: Proceedings of the 44th annual conference on Design automation*. ACM, 2007, pp. 746–749.

[5] E. B. van der Tol, E. G. T. Jaspers, and R. H. Gelderblom, "Mapping of H.264 Decoding on a Multiprocessor Architecture," in *Proceedings of SPIE*, 2003.

[6] 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 *Proceedings International Parallel and Distributed Processing Symposium*, Apr 2004.

[7] C. Meenderinck, A. Azevedo, M. Alvarez, B. Juurlink, and A. Ramirez, "Parallel Scalability of Video Decoders," *Journal Journal of Signal Processing Systems*, August 2008.

[8] S. Patel and W. mei Hwu, "Accelerator Architectures," *IEEE Micro*, vol. 28, no. 4, pp. 4–12, 2008.

[9] Y. Chen, E. Li, X. Zhou, and S. Ge, "Implementation of H. 264 Encoder and Decoder on Personal Computers," *Journal of Visual Communications and Image Representation*, vol. 17, 2006.

[10] Z. Zhao and P. Liang, "Data partition for wavefront parallelization of H.264 video encoder," in *IEEE International Symposium on Circuits and Systems.*, 2006.

[11] J. Chong, N. R. Satish, B. Catanzaro, K. Ravindran, and K. Keutzer, "Efficient parallelization of h.264 decoding with macro block level scheduling," in *IEEE International Conference on Multimedia and Expo*, July 2007, pp. 1874–1877.

[12] M. Korch and T. Rauber, "A comparison of task pools for dynamic load balancing of irregular algorithms," *Concurr. Comput. : Pract. Exper.*, vol. 16, no. 1, pp. 1–47, 2003.

[13] M. Alvarez, E. Salami, A. Ramirez, and M. Valero, "HD-VideoBench: A Benchmark for Evaluating High Definition Digital Video Applications," in *IEEE Int. Symp. on Workload Characterization*, 2007. [Online]. Available: http://people.ac.upc.edu/alvarez/hdvideobench

[14] J. Hoogerbrugge and A. Terechko, "A Multithreaded Multicore System for Embedded Media Processing," *Transactions on High-Performance Embedded Architectures and Compilers*, vol. 3, no. 2, pp. 168–187, June 2008.

[15] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the cilk-5 multithreaded language," in *Proceedings of the ACM Conference on Programming Language Design and Implementation*, 1998, pp. 212–223.

[16] R. Osorio and J. Bruguera, "An FPGA architecture for CABAC decoding in manycore systems," in *International Conference on Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008*, July 2008, pp. 293–298.

[17] J.-W. van de Waerdt, S. Vassiliadis, S. Das, S. Mirolo, C. Yen, B. Zhong, C. Basto, J.-P. van Itegem, D. Amirtharaj, K. Kalra, P. Rodriguez, and H. van Antwerpen, "The TM3270 Media-Processor," in *MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, Nov 2005, pp. 331–342.

[18] G. Al-Kadi and A. S. Terechko, "A hardware task scheduler for embedded video processing," in *Proceedings of the International Conference on High Performance Embedded Architectures and Compilers*, 2009, pp. 140–152.

[19] K. Sanjeev, H. C. J., and N. Anthony, "Carbon: architectural support for fine-grained parallelism on chip multiprocessors," in *Proceedings of the 34th annual international symposium on Computer architecture*, 2007, pp. 162–173.

[20] O. L. Klaus Schoffmann, Markus Fauster and L. Böszörmeny, "An Evaluation of Parallelization Concepts for Baseline-Profile Compliant H.264/AVC Decoders," in *Lecture Notes in Computer Science. Euro-Par 2007 Parallel Processing*, August 2007.

[21] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, "Hierarchical parallelization of an h.264/avc video encoder," in *Proc. Int. Symp. on Parallel Computing in Electrical Engineering*, 2006, pp. 363–368.

[22] T. Jacobs, V. Chouliaras, and D. Mulvaney, "Thread-parallel mpeg-2, mpeg-4 and h.264 video encoders for soc multi-processor architectures," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 1, pp. 269–275, Feb. 2006.

[23] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, and A. Rammirez, "Parallel H.264 Decoding on an Embedded Multicore Processor," in *Proceedings of the 4th International Conference on High Performance and Embedded Architectures and Compilers - HIPEAC*, Jan 2009.