Scalability of Parallel Video Decoding on Heterogeneous Manycore Architectures

Mauricio Alvarez Mesa[1], Felipe Cabarcas[2,3], Alex Ramirez[1,2]

Cor Meenderinck[4], Ben Juurlink[5] and Mateo Valero[1,2]

[1]Universitat Politècnica de Catalunya, Barcelona, Spain.
[2]Barcelona Supercomputing Center, Barcelona, Spain.
[3]Universidad de Antioquia, Medellín, Colombia.
[4]Delft University of Technology, Delft, The Netherlands.
[5]Technische Universität Berlin, Berlin, Germany.
Email: alvarez@ac.upc.edu, {felipe.cabarcas, alex.ramirez, mateo.valero}@bsc.es,
cor@ce.et.tudelft.nl, juurlink@cs.tu-berlin.de

***Abstract*—This paper presents an analysis of the scalability of the parallel video decoding on heterogeneous many core architectures. As benchmark, we use a highly parallel H.264/AVC video decoder that generates a large number of independent tasks. In order to translate task-level parallelism into performance gains both the video decoder and the architecture have been optimized. The video decoder was modified for exploiting coarse-grain frame-level parallelism in the entropy decoding kernel which has been considered the main bottleneck. Second, a heterogeneous combination of cores is evaluated for executing different type of tasks. Finally, an evaluation of the memory requirements of the whole system has been carried out. Experiments conducted using a trace-driven simulation methodology shows that the evaluated system exhibits a good parallel scalability up to 68 cores. At this point the parallel video decoder is able to decode more than 200 HD frames per second using simple low power processors.**

I. Introduction

Video coding is one of the key technologies that enable many digital video applications such as Internet video streaming, mobile video, DVD and Bluray video playback, and others. Currently the highest resolution in use is Full High Definition (FHD: 1920×1080) but to provide even better viewing experiences higher definitions, like Quad-Full High Definition (QFHD: 3840×2160) or Ultra High Definition (UHD: 7680×4320) are being proposed [26].

Although frame resolution has been increasing over time, frame rate has not. Currently most FHD systems support 25 or 50 frames per second (fps) but it has been demonstrated that these rates results in insufficient dynamic resolution generating negative visual effects like smearing and jerkiness [3]. As a solution, higher frame rates, like 100 or 200 fps, are being considered for FHD and higher resolutions. The main limitation for the adoption of higher frame rates has been the storage and computing requirements of the resulting videos.

Storage and transmission requirements of high quality video can be provided with the advanced compression techniques developed in the last generation video codecs like H.264/AVC [28]. H.264/AVC has a higher compression performance compared to all previous video coding standards such as MPEG-2 and MPEG-4. The compressing gains come at the expense of an important increase in the computational complexity [18]. Moreover, the combination of the complexity of this video codec and the higher quality of HD systems has resulted in an enormous increase in the computational requirements of the emerging high-end video coding applications.

On the other hand, performance scaling on current and future multicores and manycore architectures is based mainly in the exploitation of Thread-Level Parallelism (TLP). If the increase in the number of cores continue with every generation, in the near future there will be systems with hundreds of cores [7]. In order to exploit the performance of those parallel architectures fine-grain parallelization of applications is required.

In this paper we evaluate a heterogeneous manycore architecture for parallel H.264 decoding. Because entropy decoding has been identified as the main bottleneck we propose a solution based on the simultaneous exploitation of multiple levels of parallelism. Moreover we analyze the impact of using different type of processors for entropy decoding and macroblock decoding. Finally, we include an evaluation requirements of main memory bandwidth and on-chip data cache.

This paper is organized as follows. First, in section II we present a brief introduction to H.264 and to the

parallelization technique that has been implemented. In section III we present the scalability problems and proposed solutions. Next, in section IV we present the methodology used in the experiments, including a description of the baseline heterogeneous architecture. Then, in section V we present a discussion of the experimental results. In section VI we include an analysis of related work in the field. Finally, in section VII we present the conclusions and future work.

II. Parallel H.264 Decoding

H.264/AVC is based on the same block-based motion compensation and transform-based coding framework of prior MPEG and ITU-T video coding standards. It includes a lot of new coding tools for different application scenarios but here we present only a summary of the tools that have more effect on the performance of the video decoder which are the focus of this work. More details about the H.264/AVC standard can be found in the literature [28, 14].

A. H.264 Fundamentals

In H.264/AVC 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 (MBlocks). MBlocks, consist of one 16×16 pixel block of luma samples and two 8×8 blocks of chroma samples. MBlocks are the basic data unit for coding and decoding.

In order to decompress a video, the decoder has to apply a sequence of kernels on input data, as shown in Figure 1. The first step is entropy decoding, that reads from the input compressed bitstream and produces, as output, a set of quantized transformed coefficients, motion vectors and side information like MBlock type, coding options, etc. After entropy decoding an "inverse quantization" (or more accurately, a re-scaling operation) is applied to the quantized coefficients in order to produce a reconstructed version of the original coefficients. Then, an inverse transform is applied to them for obtaining the error signal. For reconstructing the original block a prediction signal if created. MBlock prediction is performed with intra- or inter-prediction techniques, depending on MBlock type. After prediction, the error signal is added to the prediction signal for generating the decoded MBlock. Finally, an in-loop deblocking filter is applied for improving the visual quality of the output video. This decoded MBlock is sent to the output device for displaying or storage.

B. H.264 Parallelization

H.264/AVC decoding can be parallelized in many ways, but few of them are able to scale to manycore architectures (A comparison of different parallelization techniques can be found in the literature [16, 23]). In this work we focus on a parallelization technique that operates at the MBlock-level. We use the dynamic 3D-wave algorithm that is a technique that exploits spatial and temporal MBlock-level parallelism [16]. Below we present a summary of this technique.

In H.264/AVC, MBlocks in a frame are usually 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 MBlocks inside a frame it is necessary to take into account the dependencies between them.

Processing MBlocks in a diagonal wavefront manner satisfies all these dependencies and, at the same time, allows to exploit parallelism between MBlocks [12]. This is shown in the left side of Figure 2. This parallelization technique (known as 2D-Wave) generates a variable number of independent MBlocks during frame time reducing in practice the parallelization efficiency.

Additionally to spatial MBlock-level it is possible to exploit temporal MBlock-level parallelism in which multiple frames are being decoded at the same time. This require to detect and respect the data dependencies between frames. Inter-frame dependencies are the result of data accesses in the motion compensation kernel. This kernel access an area in a previous frame pointed by a motion vector. When this reference area has been decoded it is possible to start the processing of a MBlock of the next frame even if the current frame has not been completely decoded.

The combination of spatial and temporal MBlock-level parallelism is what is known as the dynamic 3D-wave algorithm (see Figure 2). It is dynamic because the assignment of MBlocks to processors is performed at run-time based on data dependencies. This algorithm is able to extract thousands of independent MBlocks depending on the input content and the number of frames in flight [16].

III. The entropy-decoding bottleneck

Due to data dependencies entropy decoding has to be performed sequentially for all the macroblocks inside a frame. The main problem for the parallel 3D-wave decoder is that performance of the whole application is dominated by the performance of the entropy decoding stage, according to Amdahl's law. But, it is possible to parallelize entropy decoding at the frame-level because the context tables that create data dependencies are restarted every frame [15]. The only dependency that exists is related to DIRECT MBlocks in B-frames [14]. In this type of MBlocks there is not data transmitted and the motion vector is taken from the co-located MBlock in a subsequent reference frame. If motion vector prediction is performed at the entropy decoding stage, which is

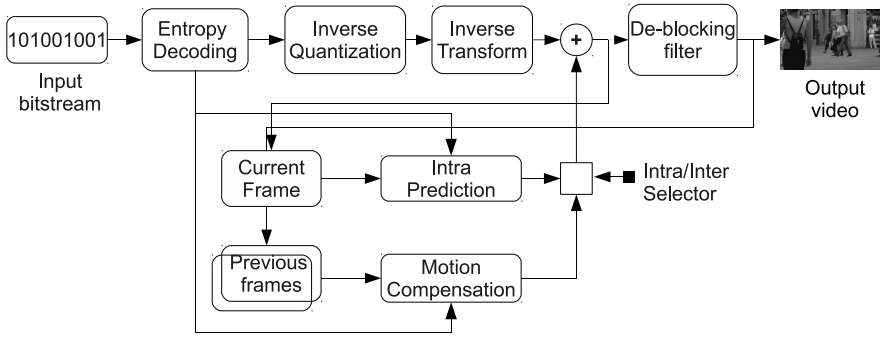


Fig. 1: H.264/AVC Decoder

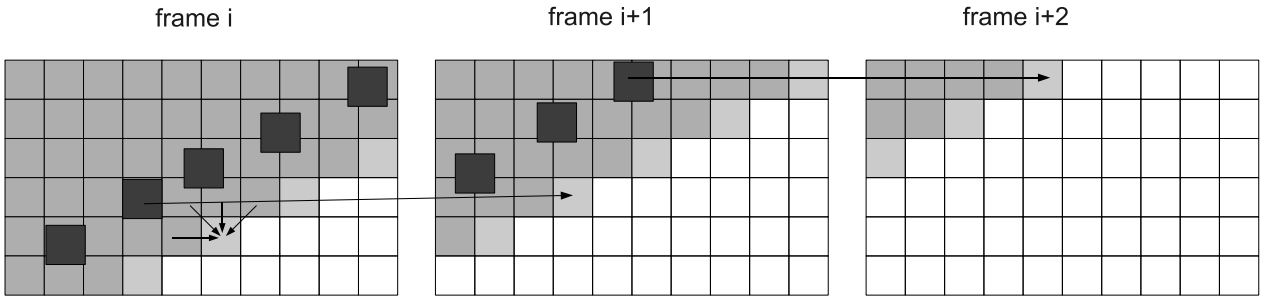


Fig. 2: 3D-wave H.264 decoding: spatial and temporal MBlock-level parallelism

usually the case, then entropy decoding of the current frame should be at least one MBlock in advance of entropy decoding of the next frame.

In order to parallelize the CABAC decoder it has to be decoupled from the MBlock decoding kernels. After decoupling, the video decoder can be seen as a macro-pipeline with a front-end and a back-end. In the front-end, there are two stages: a parsing stage that reads the compressed bitstream and parse the frame headers and a CABAC decoding stage. In the back-end there are two stages: one is MBlock decoding and the other is frame display[1]. The resulting structure of the application is shown in Figure3. This macro-pipeline combines coarse-grain parallelization for the entropy decoder and fine-grain parallelization for MBlock decoding kernels.

The front-end can use multiple processors to perform entropy decoding of multiple frames in parallel. It communicates with the back-end using a frame-buffer. The size of this buffer (in frames) defines the maximum number of frames that can be "in-flight" at any point in time. Having more frames in-flight improves the utilization of CABAC processors at the cost of more memory. Each CABAC frame requires 10.5MB for FHD resolution.

Figure 4 shows a diagram with the general concept of combination of multiple levels of parallelism. Figure 4a shows the sequential approach, in which the CABAC decoding of the next frame waits for the MBlock decoding of the current frame to finish. Figure 4b shows the diagram when pipelining parallelism is exploited. In this case, CABAC entropy decoding of the next frame can start just at the end of the CABAC decoding of the current frame. Finally, Figure 4c shows the combination of pipelining with data-level parallelism both in CABAC decoding and MBlock decoding. Multiple frames can be entropy decoded and MBlock decoded in parallel. Although is not necessary to enforce that MBlock decoding waits for the end of a whole frame of entropy decoding, we implemented in this way to reduce the amount of synchronization operations and to ensure that multiple threads can work on MBlock decoding without waiting for CABAC decoding.

A side benefit of this division is that it allows specialization. That means the use of different types of processors for different stages of the pipeline.

IV. Experimental Methodology

We use a fast trace-driven simulation methodology that allows to simulate systems with large numbers of cores.

A. H.264/AVC decoder and input videos

For these experiments we used the parallel H.264/AVC decoder that is available from HD-VideoBench [1]. This

[1]In our case uncompressed frames are not displayed on the screen and we have disabled the writing to an output file, then basically the display stage is only in charge managing the frame buffer

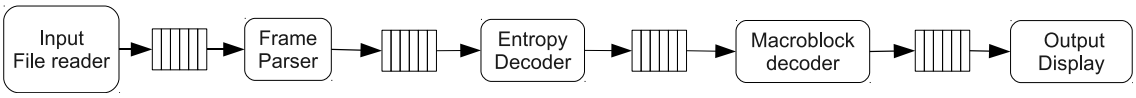
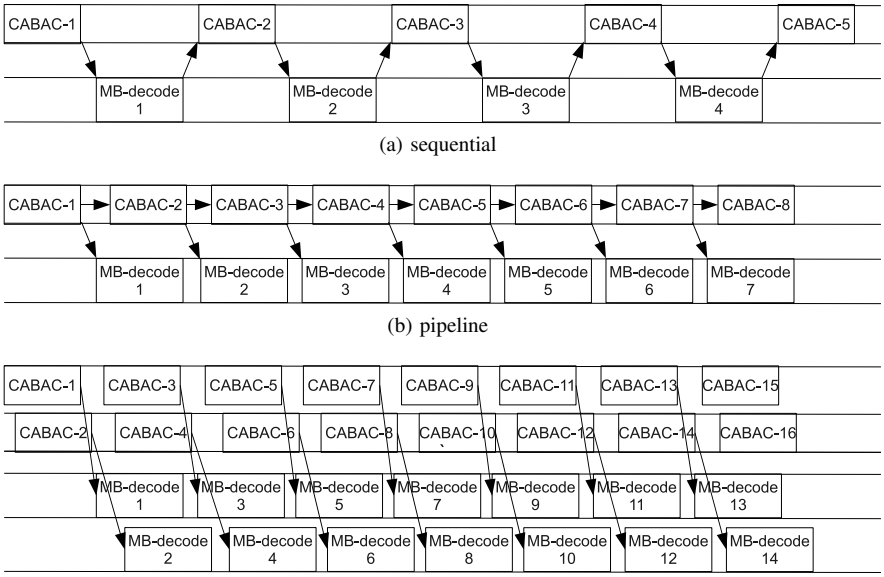


Fig. 3: H264 decoder with CABAC decoupling



(a) sequential

(b) pipeline

(c) pipeline+parallelism

Fig. 4: CABAC: multiple frames in flight and frame-level parallelism

code is based on the FFmpeg libavcodec library, and includes an implementation of the 2D-wave parallel implementation of the H.264/AVC decoder.

The benchmark includes four input sequences at different resolutions but due to space reasons only results for the *1088p25_pedestrian_area* test video are presented. Videos were encoded using the X264 encoder with the following options: 100 frames, P-frames, 1 reference frame, hexagonal motion estimation algorithm (hex) with maximum search range 24, one slice per frame. We restricted the encoder to produce only P-frames to allow parallel CABAC decoding without dependencies. One reference frame was enforced to simplify the tracking of dependencies for the 3D-wave algorithm. These two simplifications made a significant reduction in implementation complexity without losing the general applicability of the results.

B. Execution, instrumentation and trace generation

The parallel H.264/AVC decoder was executed on a SGI Altix which is a distributed shared memory (DSM) machine with a cc-NUMA architecture. The basic building block is the blade which has two dual-core Intel Itanium processors, 8GB of RAM and an interconnection module. The interconnection of blades is done using an interconnect fabric called NUMAlink-4 capable of 6.4 GB/s peak bandwidth through two 3.2 GB/s unidirectional links. The complete machine has 32 nodes with 2 dual-core processors per node for a total of 128 cores and a total of 1TB of RAM.

Each processor in the system is a Dual-Core Intel Itanium2 processor running at 1.6 GHz [8]. Main parameters of the processor are listed in Table I

The compiler used was gcc 4.1.0 and the operating system was Linux with kernel version 2.6.16.27. Traces were obtained in Paraver format using the Mintaka code instrumentation tool [20].

The code was executed using a 3 thread configuration. The first thread, called the master thread, is responsible of the main control of the application, bitstream file reading, frame parsing and all the slice initialization and finalization code. The second thread is responsible for CABAC entropy decoding. CABAC results are stored in a frame buffer for later use of by the MBlock decoding stage. Finally, the third thread is is responsible of MBlock decoding (more exactly MBlock reconstruction, including IDCT, IQ, prediction and deblocking filter).

The code was instrumented to obtain traces with CPU phases, synchronization operations and memory accesses. Execution of each thread was divided in different CPU phases and each phase was instrumented including a phase identifier and its execution time.

Configuration	SGI Altix
ISA	Itanium 64-bit
SIMD extensions	MMX, SSE, SSE2
Processor	Intel Itanium 2 9030
Technology	90nm
Clock frequency	1.6 GHz
Power	104 W
Level 1 I-cache	16 KB / core
Level 1 D-cache	16 KB / core
Level 2 I-cache	1 MB / core
Level 2 D-cache	256 KB / core
Level 3 cache	8 MB

TABLE I: Experimentation platform

Synchronization operations are special instrumentation marks that identify when a task either waits for some signal to be ready or posts a signal announcing the availability of some data. For example, CABAC decoding posts a signal every time it finishes the entropy decoding of a frame and the worker thread that process the first MBlock of a frame issues a wait operation for that signal.

Instrumented memory operations contain contains parameters such as address in main memory, size of the transfer in bytes and direction (read or write).

Paraver traces were processed with the *prv2ttf* tool to produce another trace with a task format. In the final trace there is a collection of separated tasks, each one with information of execution time of kernels, synchronization information and memory accesses. It is important to note that these are not instruction traces, but tasks traces. They contain information of CPU bursts that happen between synchronization or memory operations. The duration of these CPU burst are used for simulating systems with different number of cores as we will explain next. Our simulation methodology is similar, in concept, to the one developed by Seitner et al. [24].

C. Trace-driven Simulation of parallel architectures

For our simulations we have used TaskSim. TaskSim is a trace-driven simulator for accelerator-based multicore architectures. It targets the simulation of parallel applications coded in a master-worker task offload computational model. It uses task traces that contains information about the inter-task dependencies. That information allows TaskSim to reconstruct the dependencies at simulation time.

The computational CPU phases (bursts), such as task execution, are not simulated in detail. The burst duration is obtained from the trace and is simulated as a single event with the same runtime as the whole burst. Contrarily, the trace time for phases involving access to shared resources in the architecture, such as waiting for DMA transfers or synchronization operations, are discarded, and their timing is simulated in a cycle-accurate way by means of detailed simulation of DMA controllers, caches, interconnection, memory controllers, and DRAM modules.

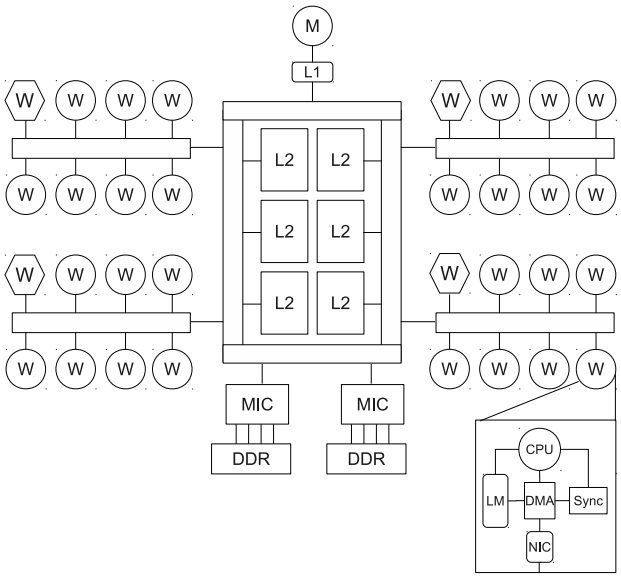


Fig. 5: Baseline Heterogeneous Multicore Architecture

On top of the task-level abstraction, TaskSim is built around an event-driven simulation framework that avoids unnecessary simulation of inactive hardware components and idle time. This is accomplished by skipping empty cycles (cycles with no activity) and selectively executing only the hardware components with scheduled activity or receiving external requests in a given cycle. This allows TaskSim to simulate hundreds of accelerators in less than an hour without loss of accuracy for architecture scalability studies[22].

D. The SARC architecture

As a baseline for simulation we have used the SARC architecture [21]. SARC is a heterogeneous multicore composed of a set of processors managed at runtime in a master-worker mode. The architecture includes different type of cores, an on-chip interconnection network, a multi-bank shared on-chip data cache, on-chip memory controllers and external memory as shown in Figure 5.

The Master processors (M) start the main() routine of the program, and run the core of the application generating tasks to be off-loaded to the specialized Worker (W) processors, as indicated by a software runtime scheduler. In the case of H.264/AVC decoding Master cores are also responsible for doing frame parsing. Worker cores execute tasks generated by the Masters ones. In this work we consider two types of workers: MBlock decoders and CABAC entropy decoders.

All processors have direct load/store access to the different scratchpads and the off-chip memory. Workers also have a DMA controller that can transfer data to/from

Memory controllers	4 x 2 DDR3 channels
Channel bandwidth	12.8 GB/s (DDR3-1600)
Memory latency	Real DDR3-1600
MIC policy	Closed-page, in-order processing
Shared L2 cache	128 MB (32 x 4 MB), 4-way assoc.
Sync. unit latency	256 cycles
L2 cache latency	40 cycles
Local Store	256 KB, 6 cycles
Interconnection links	8 bytes/cycle (25.6 GB/s)
Intra-cluster NoC	2-bus (51.2 GB/s)
Global NoC	16-bus (409.6 GB/s)

TABLE II: Baseline simulation parameters

Configuration	Low power	High Perf.
ISA	X86-64	X86-64
SIMD extensions	SSSE3	SSSE3
Processor	Atom 330	Xeon E7310
Cores	2	4
Technology	45 nm	65 nm
Clock frequency	1.6 GHz	1.6 GHz
Power	8 W	80W
Level 1 I-cache	32KB	32KB
Level 1 D-cache	24KB	32KB
Level 2 cache	1 MB	4 MB
Main Memory	1.5 GB	16 GB
Operating System	Linux 2.6.32-24	Linux 2.6.32
Compiler	gcc-4.4.3	4.4.1
Compiler Optimizations	-03	-03

TABLE III: Processor configuration for the heterogeneous system

the scratchpad memories, overlapping data transfer and computation.

Figure 5 shows a general diagram of the architecture and a detailed view of a worker node. The node is composed of a worker core (W), a local memory (LM), a DMA controller, a Network Interface Controller (NIC) that arbitrates the accesses to the bus and a synchronization module (Sync) used for on-chip synchronization operations.

The architecture has been extended to include a hardware synchronization facility. That includes a sub-unit in each worker core associated to its DMA controller for handling semaphore operations, and the addition of a global synchronization unit that keeps track of semaphore state.

As shown in Figure 5, workers are organized in clusters of N processors. In a 128-worker configuration, for example, the global NoC connects together 16 clusters of 8 processors.

Assuming 128 cores as a maximum we have defined a baseline configuration. The corresponding parameters of the simulator are shown in Table II.

V. Experimental results

In this section we present and discuss the experimental results for the proposed hardware and software optimizations.

A. Scalability of 3D-wave parallelization with multiple CABAC processors

In order to see the effect of having multiple CABAC processors we have conducted an experiment varying the number and acceleration of CABAC processors for a 3D-wave parallel decoder with a maximum of 8 frames in flight. The resulting performance can be seen in Figure 6 for 1, 2, 3 and 4 CABAC cores.

Figure 6a shows the results for 1 CABAC core. In this case, and with no CABAC acceleration (1X) a maximum performance of 80 fps is obtained with 8 MBlock decoding cores. In order to obtain more performance some acceleration on the CABAC core is needed. For example: 100 fps requires one CABAC core at 1.41X acceleration and 16 worker processors.

When the number of CABAC cores increases the scalability of the whole application improves. The decoder is able to reach 266 fps with 4 CABAC cores at 1X and 48 MBlock decoder cores. With 4 CABAC cores the CABAC front-end is not longer the bottleneck and the number of frames in flight starts to become the limiting factor. This reflects a clear tradeoff between throughput and latency (and memory usage).

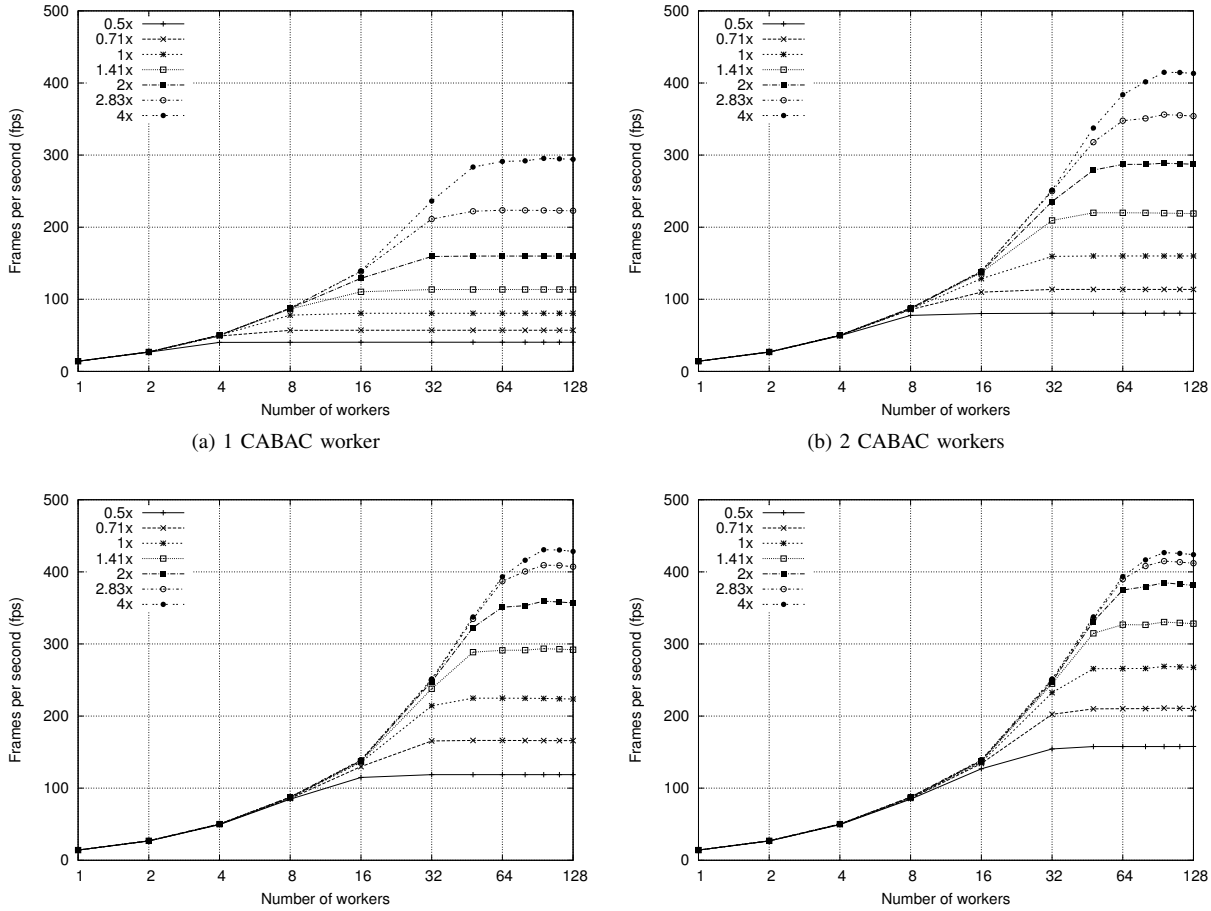
Having multiple CABAC processors with the 3D-wave algorithm allows to reach a fixed performance target even with de-accelerated CABAC cores. For example, FHDp100 can be reached with 16 MBlock decoding cores and 1 CABAC core at 1.41X, 2 CABAC cores at 0.71X, or 3 CABAC cores at 0.5X. Multiple de-accelerated cores use less area and power than one highly accelerated one.

It should be noted that the performance required to meet a real-time target depends on the input content (spatial and temporal characteristics of video). The performance and number of CABAC processors can be adjusted dynamically depending on the requirements on the specific execution of the application. This is an open research area.

B. Case study: low power heterogeneous manycore architecture

In this section we provide results for a different configuration of cores. Instead of using the Intel Itanium-2 cores we use low power processors for MBlock decoding combined with high performance cores for CABAC decoding. MBlock cores are SIMD processors based on the Intel Atom architecture which is a low power processor for mobile devices [13]). Entropy decoding cores are superscalar processors based on the Intel Xeon architecture. Both processors run at the same frequency but have a very different microarchitecture and memory hierarchy (asymmetric cores). Table III shows the main parameters for both processors.

Figure 7 shows the resulting performance in terms of



(a) 1 CABAC worker

(b) 2 CABAC workers

(c) 3 CABAC workers

(d) 4 CABAC workers

Fig. 6: CABAC acceleration and multiple CABAC processors for the 3D-wave H.264 decoder

frames per second for the homogeneous (Figure 7a) and the heterogeneous cases (Figure 7b).

First, it is important to note that it is not possible to decode FHD video in real-time (25 fps) on a single Atom core. Using a homogeneous system based on Atom processors requires at least 1 CABAC core and 8 MBlock cores to go beyond 25 fps. Decoding 50 fps requires 2 CABAC cores and 16 MBlock cores. The system is able to scale up to 48 MBlock cores with 4 CABAC cores reaching almost 140 fps. Adding more cores (CABAC or MBlock) results in diminishing benefits due to the limit of CABAC frames in flight.

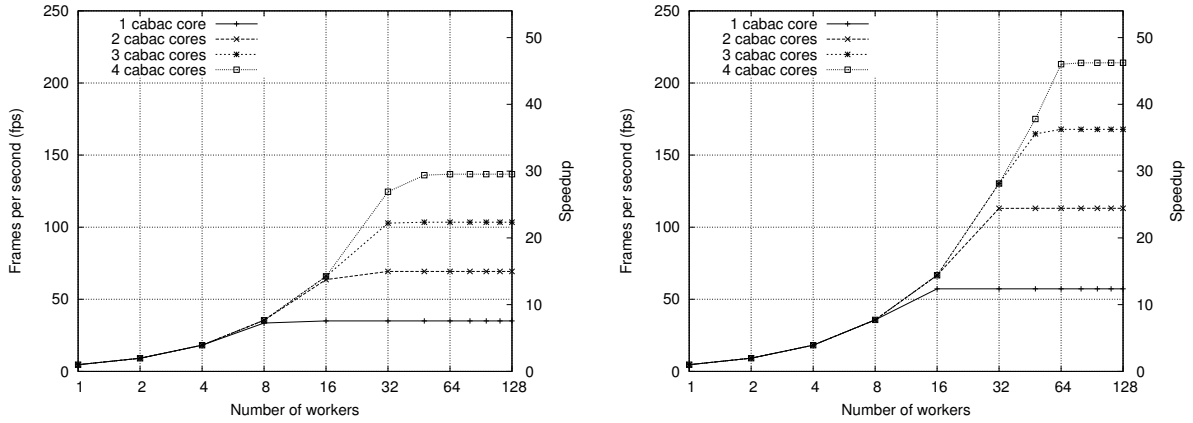
The heterogeneous configuration improves the performance of all the simulated configurations by 1.64X in average. Using asymmetric cores it is possible to decode 50 fps with 1 CABAC core and 16 MBlock cores. As a maximum, the system is able to decode 213 fps using 64 MBlock cores and 4 CABAC cores (plus one master core, for 69 cores in total).

C. Memory Requirements

Previous experiments have been conducted using a powerful configuration of main memory and cache hierarchy as shown in Table II. In this section we evaluate the actual memory requirements. As a baseline we configured a system with 4 CABAC cores and 64 MBlock decoding cores (and 1 master core) using the heterogeneous configuration presented in the previous section.

1) Impact of main memory bandwidth: In order to isolate the effects of main memory bandwidth we have disabled the L2 cache. Figure 8 shows the results. For reaching the maximum performance, 212 fps, a minimum of 102.4 GB/s are required. This can be provided with 4 MICs each one having two DDR-3-1600 modules.

2) Impact of L2 cache: In the baseline architecture each processor is equipped with two types of individual memories: a scratchpad and a L1 data cache. In the case of the 3D-wave decoder all the accesses to main memory have been implemented using explicit DMA commands.



(a) homogeneous (b) heterogeneous

Fig. 7: Parallel H.264 decoder with different type of cores

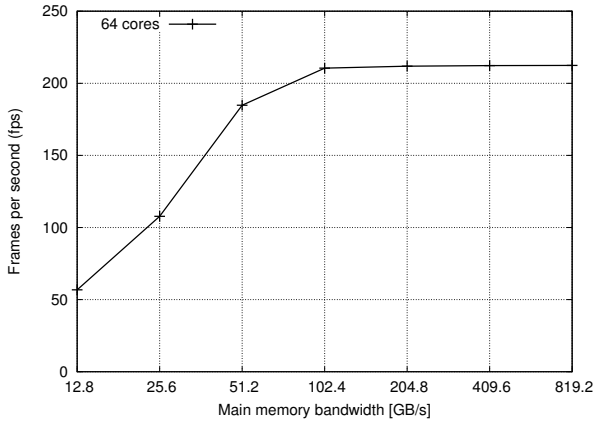


Fig. 8: Impact of memory bandwidth

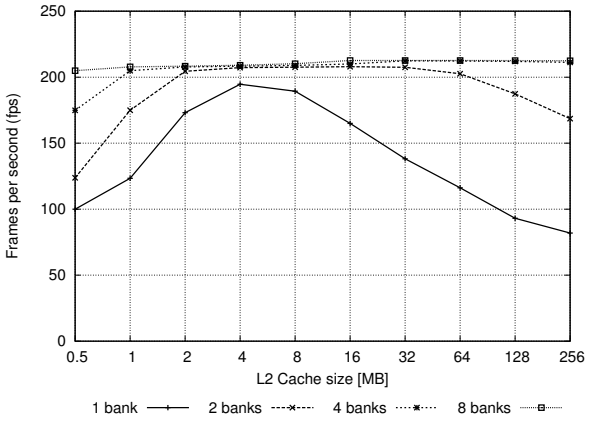


Fig. 9: Impact of L2 cache

As a result all the local accesses use the scratchpad memory but not the L1 data cache.

A shared multi-bank L2 cache is included in the architecture as it has been explained in section IV-D. The shared cache maintains a significant part of the memory accesses on-chip by capturing the references made by DMA controllers. With a multi-bank structure and a careful mapping of accesses to banks it is possible to provide high on-chip bandwidth and low latency access.

In order to determine the best cache configuration we change both the number of banks and the cache size. The latencies of cache banks with difference sizes have been estimated using CACTI 5.3 [27], for 45nm memory technology and the system is simulated with a main memory composed of 1 MIC with two DDR3 channels (12.8 GB/s). Figure 9 shows the effect of cache banks. The maximum performance of 212 fps can be reached either having a big cache (4MB) with small number of banks (2 banks), or having a smaller cache (1 MB) with a high number of banks (8 banks). In the first case a larger cache means a higher access latency, while in the second case a banked cache has lower latency but requires more bandwidth on the on-chip interconnection network.

VI. Related Work

der Tol et al.proposed the 2D-Wave algorithm for parallelization of the H.264/AVC decoder [12]. Chen et al. evaluated a similar approach: a combination of MB parallelism and frame-level parallelism for the H.264 encoder on Pentium machines with SMT and CMP capabilities [9]. Zhao and Liang presented a combination of frame-level parallelism and MBlock-level parallelism for H.264 encoding [29]. This scheme is a static variation of the 3D-Wave algorithm. Baik et al. developed a hybrid approach combining function-level parallelism and data-level parallelism for the H.264 decoder on the Cell B.E. architecture [5].

Alvarez et al. reported a scalability analysis of the 2D-wave algorithm on a cc-NUMA architecture [2]. Meenderinck et al. presented the original idea of the dynamic 3D-wave algorithm but without any real implementation [16]. Azevedo et al. reported an implementation of the 3D-wave algorithm for embedded media processors that scales up to 64 processors but not taking into account the CABAC stage [4].

Seitner et al. presented a comparison of different data-level parallelization approaches based on slice-level, and MBlock-level parallelism [23]. Nishihara et al. presented an implementation of the row-order MB-level parallelization of the decoder combined with function-level parallelism [17]. Sihn et al. applied a similar technique in which a combination of function-level and MB-level parallelism is optimized for better load balancing and a reduction of memory accesses [25].

Baker et al. implemented the row variant of intra-frame MB-level parallelism described by Seitner et al. on the Cell B.E. architecture. They mapped entropy decoding onto the PPE processor and MBlock decoding onto the SPE processors [6]. Chi et al. implemented a variant of the MB-level row approach and compared it with a centralized task-pool implementation [10]. Cho et al. implemented a similar approach on the cell processor but included a frame-level parallelization of the entropy decoder. In their implementation the master processor (PPE) runs a parser thread and two entropy decoding threads [11].

GPUs are also heterogeneous multicore architectures but some attempts to use them for video decoding exhibit limited performance gains because of the irregular behavior of H.264 decoding [19].

Our work make the following new contributions. First, we present a scalability analysis for architectures with manycores taking into account and solving the entropy decoding bottleneck. Previous works on highly parallel versions of the H.264 decoder did not consider CABAC or were clearly limited by its performance. Finally, we have analysed the suitability of heterogeneous architectures that use few high performane cores and many low power simple processors for providing the required performance for high quality video applications.

VII. Conclusions

We have presented an analysis of the scalability of parallel video decoding on heterogeneous manycore architectures. We have shown that it is possible to remove the entropy decoder bottleneck by exploiting multiple levels of parallelism such as pipeline parallelism, frame-level parallelism and macroblock-level parallelism. We have demonstrated that is possible to achieve the performance required by high bandwidth applications using processors with modest or no acceleration at all; even with multiple simpler cores operated at a reduced rate compared to the base processor. This allows to process, in real-time, emerging video applications with higher resolutions and frame rates.

We also presented the performance of the parallel H.264 decoder on a heterogeneous manycore architecture in which simpler low power processors are assigned to data parallel kernels and high performance cores for entropy decoding. Using this configuration it was possible to achieve high performance with a reduced power consumption.

In this paper we only consider a static combination of high performance and low power cores. One area of future work is to consider dynamic systems in which processor performance can be adjusted at run-time and tasks can be allocated dynamically to heterogeneous processors based on task complexity.

References

[1] 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*, Sept. 2007. URL http://people.ac.upc.edu/alvarez/hdvideobench.

[2] Mauricio Alvarez, Alex Ramírez, Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, and Mateo Valero. Scalability of Macroblock-level Parallelism for H.264 Decoding. In *The Fifteenth International Conference on Parallel and Distributed Systems (ICPADS'09)*, Dec 2009.

[3] M. Armstrong, D. Flynn, M. Hammond, S. Jolly, and R. Salmon. High frame-rate television. Technical report, BBC, 2009.

[4] Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, and Alex 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.

[5] Hyunki Baik, Kue-Hwan Sihn, Yun il Kim, Sehyun Bae, Najeong Han, and Hyo Jung Song. Analysis and parallelization of h.264 decoder on cell broadband engine architecture. In *2007 IEEE International Symposium on Signal Processing and Information Technology*, pages 791–795, 2007.

[6] Michael A. Baker, Pravin Dalale, Karam S. Chatha, and Sarma B.K. Vrudhula. A scalable parallel h.264 decoder on the cell broadband engine architecture. In *CODES+ISSS '09: Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 353–362. ACM, 2009.

[7] Shekhar Borkar. Thousand core chips: a technology perspective. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 746–749. ACM, 2007.

[8] McNairy Cameron and Soltis Don. Itanium 2 processor microarchitecture. *IEEE Micro*, 23:44–55, March 2003.

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

[10] Chi Ching Chi, Ben Juurlink, and Cor Meenderinck. Evaluation of parallel H.264 decoding strategies for the Cell Broadband Engine. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 105–114, 2010.

[11] Yongjin Cho, Seungkyun Kim, Jaejin Lee, and Heonshik Shin. Parallelizing the H.264 decoder on the cell BE architecture. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 49–58, 2010.

[12] 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.

[13] G. Gerosa, S. Curtis, M. D'Addeo, Bo Jiang, B. Kuttanna, F. Merchant, B. Patel, M. Taufique, and H. Samarchi. A Sub-1W to 2W Low-Power IA Processor for Mobile Internet Devices and Ultra-Mobile PCs in 45nm Hi-k; Metal Gate CMOS. In *IEEE International Solid-State Circuits Conference, 2008. ISSCC 2008*, pages 256–611, feb. 2008.

[14] h264. ISO/IEC 14496-10 and ITU-T Rec H.264, Advanced Video Coding, 2003.

[15] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July 2003.

[16] Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez. Parallel Scalability of Video Decoders. *Journal of Signal Processing Systems*, 57:173–194, November 2009.

[17] K. Nishihara, A. Hatabu, and T. Moriyoshi. Parallelization of H.264 video decoder for embedded multicore processor. In *2008 IEEE International Conference on Multimedia and Expo*, pages 329–332, 2008.

[18] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video Coding with H.264/AVC: Tools, Performance, and Complexity. *IEEE Circuits and Systems Magazine*, 4(1):7–28, Jan 2004.

[19] Bart Pieters, Dieter Van Rijsselbergen, Wesley De Neve, and Rik Van de Walle. Performance evaluation of H.264/AVC decoding and visualization using the GPU. In *Applications of Digital Image Processing XXX*, page 669606, 2007.

[20] V. Pillet, J. Labarta, T. Cortes, and S. Girona. Paraver: A tool to visualize and analyze parallel code. In Patrick Nixon, editor, *Proceedings of WoTUG-18: Transputer and occam Developments*, pages 17–31, mar 1995. ISBN 90 5199 222 X.

[21] A. Ramirez, F. Cabarcas, B. Juurlink, M. Alvarez Mesa, A. Azevedo, C. Meenderinck, G. Gaydadjiev, C. Ciobanu, S. Isaza, and F. Sanchez. The SARC Architecture. *IEEE Micro*, 30(5):16–29, Sept/Oct. 2010.

[22] Alejandro Rico, Felipe Cabarcas, Antonio Quesada, Milan Pavlovic, Augusto Javier Vega, Carlos Villavieja, Yoav Etsion, and Alex Ramírez. Scalable Simulation of Decoupled Accelerator Architectures. Technical Report UPC-DAC-RR-2010-14, Universitat Politècnica de Catalunya (UPC), 2010.

[23] Florian H. Seitner, Ralf M. Schreier, Michael Bleyer, and Margrit Gelautz. Evaluation of data-parallel splitting approaches for H.264 decoding. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 40–49, 2008.

[24] Florian H. Seitner, Michael Bleyer, Margrit Gelautz, and Ralf M. Beuschel. Development of a high-level simulation approach and its application to multicore video decoding. *IEEE Trans. Cir. and Sys. for Video Technol.*, 19:1667–1679, November 2009.

[25] Kue-Hwan Sihn, Hyunki Baik, Jong-Tae Kim, Sehyun Bae, and Hyo Jung Song. Novel approaches to parallel h.264 decoder on symmetric multicore systems. In *IEEE International Conference on Acoustics, Speech and Signal Processing, 2009. ICASSP 2009*, pages 2017–2020, 2009.

[26] Masayuki Sugawara. Super hi-vision — research on a future ultra-hdtv system. Technical report, European Broadcasting Union, 2008.

[27] Shyamkumar Thoziyoor, Naveen Muralimanohar, and Norman P. Jouppi. Cacti 5.0. Technical Report HPL-2007-167, Advanced Architecture Laboratory HP Laboratories, 2007.

[28] 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*, 13(7):560–576, July 2003.

[29] Zhuo Zhao and Ping Liang. Data partition for wavefront parallelization of H.264 video encoder. In *IEEE International Symposium on Circuits and Systems.*, 2006.