Design of the H264 application and Implementation on Heterogeneous Architectures

Chahrazed Adda
University of Tiaret Ibn khaldoun
LIM laboratory, Tiaret
Algeria

Abou Elhassen Benyamina
University of Oran1 Ahmed Ben Bella
LAPECI laboratory, Oran
Algeria

ABSTRACT
Multimedia applications are present in most mobile hand-held devices.H.264 is an emerging video coding standard, which aims at compressing high-quality video contents at low-bit rates. While the new encoding and decoding processes are similar to many previous standards, the new standard includes a number of new features and thus requires much more computation than most existing standards do. The complexity of H.264 standard poses a large amount of challenges to implementing the encoder/decoder in real-time requiring large amount of processing resources. This paper presents the design and analysis of the H.264 decoder implemented on a heterogeneous architecture (multi-CPUs/multi-GPUs). A model-driven approach is adopted by using the standard MARTE profile of UML. Our approach is based on hybrid partitioning that combines both functional and data partitioning which is applied to find the most suitable processors (CPU or GPU) regarding the execution time. We claim that our approach allows giving a better performance, which is crucial when implemented in modern complex systems.

Keywords
General-Purpose Graphics Processing Unit (GPGPU), Multimedia, H.264/AVC decoder, Parallel Processing, Functional Partitioning, Data Partitioning.

1. INTRODUCTION
In recent years, the performance improvement of Graphics Processing Unit (GPU) is remarkable and GPUs are becoming attractive as accelerators for heavy tasks. GPUs are now commonly used as co-processors in many embedded systems to accelerate general-purpose applications. They are particularly capable of executing data-parallel applications, due to their highly multithreaded architecture and high-bandwidth memory. Various embedded system domains can benefit high performance and better energy efficiency from utilizing GPUs. For example, GPUs can efficiently perform matrix operations such as factorization on large data sets and multidimensional FFTs and convolutions. Such operations are often seen in many embedded applications including signal processing, imaging and video processing. By leveraging new programming models, such as CUDA [1] and OpenCL [2], programmers can effectively develop highly data-parallel kernels to execute such applications on GPUs.

By integrating heterogeneous processing elements with different performance characteristics in the same system, heterogeneous CPU/GPU architectures are expected to provide more flexibility for better performance compared to homogeneous systems. Execution time and energy consumption are imperative performance metric that needs to be optimized in most embedded systems. In order to minimize execution time and energy consumption for running a set of workloads, the step that partitioning computations to processing elements is critical. In this paper, we consider the partitioning of workload problem in a heterogeneous system containing multiple CPUs and GPUs. Our goal is to minimize the execution time.

Embedded applications that we are used to process are generally complex such as MPEG, H263, and H264.....etc etc encoders. This type of application is characterized by parts of different types, a party is treated regularly and the other irregularly. The latter represents intensive computing. For this type of application two types of parallel processing are applied, regular processing (task Parallelism) and irregular processing (data Parallelism). Furthermore, video coding standards like H.264/AVC [3] and HEVC [4] are adopting complex algorithms like context-adaptive binary arithmetic coding (CABAC) and variable length coding (CAVLC) in order to achieve better compression and thus lower transmission bitrates for high resolution video sequences. The additional complexity of these algorithms has a major impact by increasing execution time and energy consumption.

In our research, we intend to solve the problem of high complexity of the H.264 decoder using parallelization on multicore embedded processors and on graphical processors. Video resolutions are increasing rapidly, which require more processing time and consequently more energy consumption. Many solutions based on parallel execution exist ranging from macroblocks (fine-grain) till groups of pictures (coarse-grain) parallel decoding. Macroblock parallel decoding is highly scalable since many macroblocks can be processed in parallel. However, dependencies and huge overheads are created as a result of communication and synchronization between macroblocks. Parallel decoding of groups of pictures require large memories for high definition video sequences. In addition, they have a lower scalability than macroblock decoding because of the limited number of groups of frames that can be decoded in parallel. Our solution is based on the H264 video decoder design taking into account the different parallelism and heterogeneous multiprocessor architecture CPUs/GPUs.

Our main contribution in this paper is to propose a new approach based on modeling analyze the computational requirements of H.264 decoder and implement H.264 decoder on heterogeneous architecture (multi-CPUs/multi-GPUs) in order to minimize execution time. Our approach utilizes parallel processing techniques such as workload partitioning.

The remainder of the paper is organized as follows. In Section 2, we present the related work concerning H.264 parallel optimizations. In Section 3, we describe our approach for parallel execution of macroblock rows of the H.264 decoder. In Section 4, we present the experimental results for execution time on CPUs and GPUs using a simulator for multicore processors. Final conclusion and future work are given in Section 5.

2. RELATED WORKS

Ever since the H.264/AVC standard [3] was published in 2003, researchers started to solve the high complexity issue of the new standard mainly using parallelism. Several modifications were suggested for the H.264 encoders and decoders in order to improve the performance in terms of execution time and memory usage. Parallel decoding techniques of H.264 exist from the highest level, which is the group of frames or pictures (GOP), the coarse-grain level, till the lowest level, which is the block inside a macroblock, the fine-grain level. Kannangara [5] reduced the complexity of the H.264 decoder (19-65%) by predicting the SKIP macroblocks using an estimation based on a Lagrangian rate-distortion cost function. Gurhanli [6] suggested a parallel approach by decoding independent groups of frames on different cores. The speedup is conditioned with the modification of the encoder in order to omit the start-code scanner process. Any modification to the encoder will require a long process for modifying the H.264 specification in order to be compliant with the standard. The exclusion of previously encoded video sequences is also an effect for modifying the H.264 encoder. Nishihara [7] proposed a load balancing mechanism among cores where partitions sizes are adjusted during runtime. He also reduced the memory access contention based on execution time prediction. Among frame-level and MB-level parallelization, the 3D-wave technique proposed by Azevedo [8] decodes independent MBs in parallel on different cores. A good scalability is proved for HD resolutions where macroblocks are scanned in zigzag mode and decode independent macroblocks in parallel. Chong [9] added a pre-parsing stage in order to resolve control dependencies for MB-level parallelization. Van Der Tol [10] mapped video sequences data over multiple processors providing better performance over functional parallelization. He groups macroblocks in a way that minimal dependency between cores is required. Horowitz [11] compared different H.264 implementations including FFmpeg [17] and the H.264 reference software JM [18]. He also analyzed the complexity of the H.264 decoder subsystems. Sihn [12] proposed a multicore pipeline for the deblocking filter based on the group of pictures data level partitioning. He also suggested software memory throttling and fair load balancing techniques in order to improve multicore processors performance when several cores are used. [13] Proposes to decode the lines of the macroblocks in parallel on a certain number of cores of the processors, the dependencies between macroblocks are ignored. [16] Implements H264 decoder on FPGA architecture. In our research we optimize the H.264 decoder knowing that our approach can be also applied to the H.264 encoder. We focus on improving the efficiency of the H.264 decoder using heterogeneous processors CPU/GPU. We model the H264 / AVC decoder with the recent standard MARTE profile [14] taking into account the different parallelism (component parallelism and data parallelism) and the data dependency between macroblocks. We map our implementation on CPU and GPU processors. Execution time implementation is calculated using simulated execution time. We further implement an OpenCL [2] version of our parallel H.264 implementation. Simulation experiments on heterogeneous processors are conducted using a CPU-GPU simulation Multi2Sim [21].

3. HETEROGENEOUS SYSTEM ARCHITECTURE (HAS)

The Generic graphics processors or GPGPU (General Purpose GPU) have evolved in such a way that they can now perform parallel calculations for a wide range of applications. However, programming these devices at the same time as the CPU present on the same chip constitutes a major difficulty. A new architectural concept, The HSA (Heterogeneous System Architecture), paves the way for greater fluidity in the development of heterogeneous code.

The GPUs (Graphics Processing Unit) have passed in recent years from the status of purely graphic accelerators to that of generic parallel processors, Supported by standard APIs and tools such as OpenCL and DirectCompute. Despite this promising start, there are still many obstacles to the existence of an environment using the GPU in a manner as transparent as the CPU (Central Processing Unit, General processor) for current tasks of programing. The CPU and the GPU, in particular, manage different memory spaces, and the hardware is not virtualized. The HSA (Heterogeneous System Architecture) architecture eliminates these obstacles, So that the programmer can exploit the parallel processor contained in the GPU as a coprocessor of the same level as the traditional multi-threaded CPU.

HSA is actually a software layer that provides a unified view of the fundamental processing elements, and allows the programmer to write applications that smoothly integrate CPUs and GPUs while benefiting from the best characteristics of each. The underlying strategy is to create a unified programming platform that serves as a foundation for the deployment of languages, Frameworks and applications that exploit parallelism. More specifically, HSA intends to break the programmability barrier CPU/GPU, Reduce communication latency CPU / GPU, Open the platform to a wider range of applications by accepting existing programming models, And prepare the reception of new processing units in addition to CPU and GPU[19].

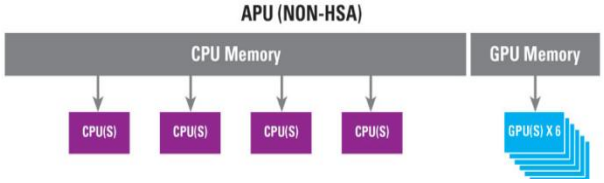

Fig 1: Architecture HAS

New class processors known as Accelerated Processing Unit or APU in a grate CPUs and GPUs in the same computer chip [20], two different processor units working together like a human brain is a enable by heterogeneous system architecture or HSA. “Figure 1”.

Two sides of AMD API (CPU and GPU) share the same system memory known as heterogeneous Uniform Memory Access or hUMA1 via cache coherent views. Advantages include an easier programming model and less copying of data between separate memory pools.

In our work we are interested by news type of architecture such as heterogeneous architecture GPGPU (multi-CPUs/multi-GPUs) based on Heterogeneous System

Architecture (HSA) in order to benefit from advantages of HSA.

4. AN OVERVIEW OF THE H.264 STANDARD

H.264/ AVC [22] video compression standard takes advantage from spatial and temporal redundancy in a video sequence. Therefore, it defines various prediction modes to predict each macroblock depending on its texture properties.

In encoder processing, residual macroblocks consists of difference between original macroblocks and the corresponding predicted one. Residual is the final data organized in bitstream.

Decoder is responsible to reconstruct a video sequence from the compressed data created by encoder. As shown in Figure 1, first step is entropy decoding. It receives the compressed bitstream to reconstruct video parameters and residual coefficients. Then, two primary paths are considered in decoder process. First one is the decoding of residual macroblocks by inverse quantization and inverse transform. Second path is the generation of the predicted macroblocks according to prediction mode fixed by encoder. The addition of the outputs of these two paths is the reconstruct macroblock. A deblocking filter is then applied to have a better video quality. For more details, following sub-sections describe every module of decoder.

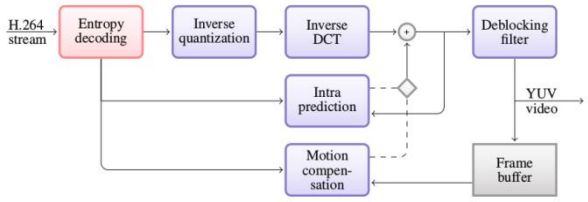

Fig 2: H.264/AVC decoder

4.1 Elements of a Video Sequence

H.264 is a block-based coder/decoder (codec), meaning that each frame is divided into small square blocks called macroblocks (MBs). The coding tools / kernels are applied to MBs rather than to whole frames, thereby reducing the computational complexity and improving the accuracy of motion prediction. Figure 3 depicts a generic view of the data elements in a video sequence. It starts with the sequence of frames that comprise the whole video. Several frames can form a Group of Pictures (GOP), which is an independent set of frames. Each frame can be composed of independent sections called slices, and slices ones, in turn, consist of MBs. Each MB can be further divided into sub-blocks, which in turn, consist of pixels.

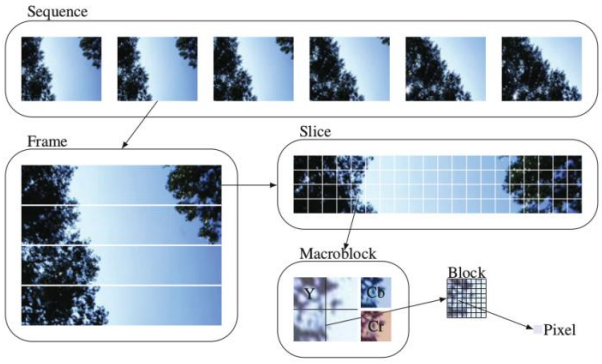

Fig 3: Elements of a video sequence

A MB consists of separated blocks for luma (denoted by Y) and chroma signals (denoted by Cb and Cr). A pre-processing step has to be applied to convert video from a different color component format (such as red-green-blue, RGB) to the Y Cb Cr color model. Chroma sub-sampling is applied to reduce the amount of color information, since the human eye is more sensitive to brightness (Y) than to color (Cb and Cr) [23]. The most common color structure is denoted by 4:2:0 in which the chroma signals (Cb and Cr) are sub-sampled by 2 in both dimensions. As a result, in H.264, as in most MPEG and ITU-T video codecs, each MB typically consists of one 16 × 16 luma block and two 8 × 8 chroma blocks.

4.2 Frame Types

H.264 defines three main types of frames: I-, P-, and B frames. An I-frame uses intra-prediction and is independent of other frames. In intra-prediction, each MB is predicted based on adjacent blocks from the same frame. A P-frame (Predicted frame) uses motion estimation as well as intra-prediction and depends on one or more previous frames, which can be either I-, P- or B-frames. Motion estimation is used to exploit temporal correlation between frames. Finally, B-frames (Bidirectionally predicted frames) use bidirectional motion estimation and can depend on previous frames as well as future frames [24].

Figure 4 illustrates a typical I-P-B-B (first an I-frame, then two B-frames between P-frames) sequence. The arrows indicate the dependencies between frames caused by motion estimation. In order to ensure that a reference frame is decoded before the frames that depend on it, and because B-frames can depend on future frames, the decoding order (the order in which frames are stored in the bitstream) differs from the display order. Thus a reordering step is necessary before the frames can be displayed, adding to the complexity of H.264 decoding.

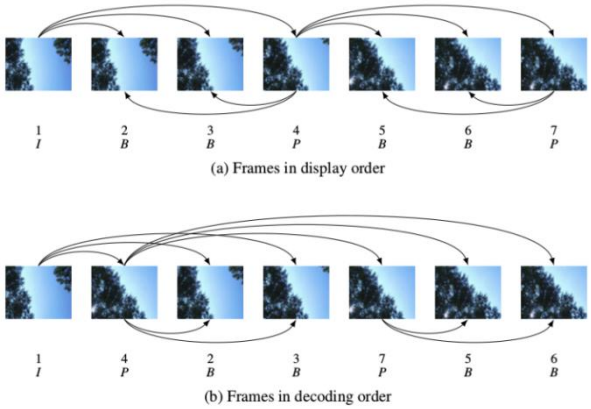

Fig 4: Types of frames and display order versus decoding order

4.3 H.264 Decoding Tools

The H.264 standard has many decoding tools each one with several options. Here we can only briefly mention the key features.

4.3.1 Entropy decoding

After decoding Network Abstraction Layer (NAL) parameters, the data elements are entropy decoded by two ways: Context-based Adaptive Variable Length Decoding (CAVLD) or a binary arithmetic coder (CABAC) which achieves higher compression and Exp-Golomb.

Exp-Golomb: is used for others syntaxes elements such as prediction mode and quantization parameter.

CAVLD is more time consuming than Exp-Golomb [27]. It is used to reconstruct and to reorder data on 4x4 block of 16 integers. By the mean of standard code tables, each 4x4 block is decoded into five syntax elements: Coefftoken, Sign, Level, TotalZeros, and Run [22][23].CAVLDis easier to implement than CABAC.

4.3.2 Inverse quantization

CAVLD output is a residual quantified macroblock. Following step is inverse quantization to produce a set of coefficients (W_{ij}). Since quantization is a losing information step, inverse quantization reconstructs data. It is multiplication operation as described in equation 1, where $Z_{ij}$is inverse quantization input, W_{ij} is its output and Q_{step} is a quantization factor given by standard according to $Q_p$value.

Q_p Is the quantization parameter fixed by encoder. It is decoded from the bitstream using Exp-Golomb codes.

$$W_{ij}=Z_{ij}\cdot Q_{step} \qquad (1)$$

In order to manipulate only integer value in transform step, H.264 standard have postponed real multiplication operation from transform to quantization [23].Details of this operation is given in inverse transform sub section. The final inverse quantization equation given by standard is described by equation 2, whereV_{ij} is the rescaling factor defined by the standard.

$$W_{ij}=Z_{ij}\cdot V_{ij}\cdot 2^{\mathrm{floor}(Q_p/6)} \qquad (2)$$

To implement this equation, a number of shifts equal to "$\mathrm{floor}(Q_p/6)$" was used instead of arithmetic multiplication. Shift operation is less time consuming than multiplication operation.

4.3.3 Inverse transform

In previous video coding standards, Inverse Discrete Cosine Transform (DCT) was used. Inverse transform step is applied for each 4x4 block. For 16x16 Intra prediction modes, a suppliant Hadamard transform is adding for DC Coefficients. Most of the energy is concentrated in the DC coefficients for a 16x16 intra coded macroblock. This extra transform helps to de-correlate the DC coefficients to take advantage of the correlation among coefficients. As shown in Figure 5, DC coefficients of each 4x4 block are assembling in a matrix to applied inverse Hadamard transform given by equation 4. An inverse DC quantization is also applied on DC matrix.

$$R=F^T(T\otimes E)F$$

$$\begin{bmatrix}1 & 1 & 1 & 1/2\\ 1 & 1/2 & -1 & -1\\ 1 & -1/2 & -1 & 1\\ 1 & -1 & 1 & -1/2\end{bmatrix}\left(\begin{bmatrix}T_{00} & T_{01} & T_{02} & T_{03}\\ T_{10} & T_{11} & T_{12} & T_{13}\\ T_{20} & T_{21} & T_{22} & T_{23}\\ T_{30} & T_{31} & T_{32} & T_{33}\end{bmatrix}\otimes\begin{bmatrix}a^2 & ab & a^2 & ab\\ ab & b^2 & ab & b^2\\ a^2 & ab & a^2 & ab\\ ab & b^2 & ab & b^2\end{bmatrix}\right)\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1/2 & -1/2 & -1\\ 1 & -1 & -1 & 1\\ 1/2 & -1 & 1 & -1/2\end{bmatrix} \qquad (3)$$

With a=$\frac{1}{2}$ and b=$\sqrt{\frac{2}{5}}$

$$R_{DCLuma}=\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & -1\\ 1 & -1 & 1 & -1\end{bmatrix}\begin{bmatrix}T_{DC0} & T_{DC1} & T_{DC2} & T_{DC3}\\ T_{DC4} & T_{DC5} & T_{DC6} & T_{DC7}\\ T_{DC8} & T_{DC9} & T_{DC10} & T_{DC11}\\ T_{DC12} & T_{DC13} & T_{DC14} & T_{DC15}\end{bmatrix}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & -1\\ 1 & -1 & 1 & -1\end{bmatrix} \qquad (4)$$

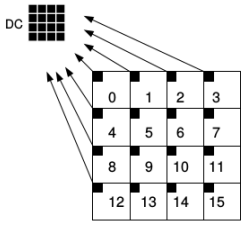

Fig 5:DC coefficients positions in a macroblock

4.3.4 Inverse prediction

Because of redundancy in video sequence, H.264/AVC standard is based on two principal prediction modes [22]. Temporal resemblance between frames is treated as inter prediction. Spatial resemblance in same frame is treated as intra prediction. In LETI decoder, first frame of a sequence is necessarily Intra (4x4 or 16x16) coded because it hasn't reference frame. For next P frames, each macroblock can be coded intra (4x4 or 16x16) or inter prediction.

The 4x4 intra prediction modes are suitable for significant details within a frame. Each 4x4 block is predicted independently from spatially neighboring coefficients. One of nine prediction modes illustrated by Figure 6 [23] is used. According to adjacent block availability, modes can be applied or not. Vertical prediction mode (called also Mode 0) cannot be applied only if top neighboring block at least is available, because this mode copies pixels above the 4x4 block as indicated in Figure 6. For horizontal prediction mode (Mode 1), the pixels to the left of the 4x4 block are copied horizontally if available. Adjacent pixels availability is not necessary to perform DC prediction mode (mode 2). The remaining 6 modes are diagonal prediction modes. They use defined equation to privilege specified direction. Directional modes are suited but they entail additional complexity in the decoding process [25].

The 16x16 intra predictions is characterized by four prediction modes: horizontal mode, vertical mode, DC mode and planer mode. Except of planer mode, all modes have respectively the same propriety of 4x4 modes but they are applied on a 16x16 macroblock. In planer mode, a curve fitting equation is used to form a prediction block having a brightness and slope in the horizontal and vertical directions that approximately matches the neighboring pixels. After statistic work [27], planer mode has been eliminated from LETI encoder because of its supplementary incising complexity relative to its video quality contribution.

In inter prediction case; motion vector is first extracted from bitstream. Then, motion compensation module is applied. It consists of adding motion vector coordinates to corresponding block in reference frame. Result is reconstructed block. Block size can change from one motion vector to other. Different block sizes are supported in H.264/AVC standard, as shown in Figure 7. In LETI decoder only one frame reference is applied and smaller block size for motion vector is 8x8[27].

The data obtained from the intra or inter prediction is added to the inverse transformed residual coefficients. This sum is

copied to the decoded buffer which is used as an input for deblocking filter step.

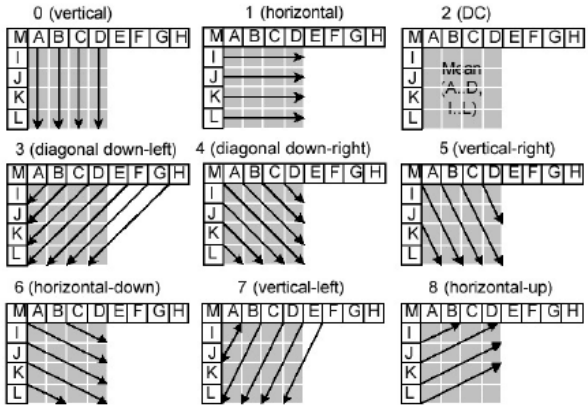

Fig 6:4x4 Intra prediction modes

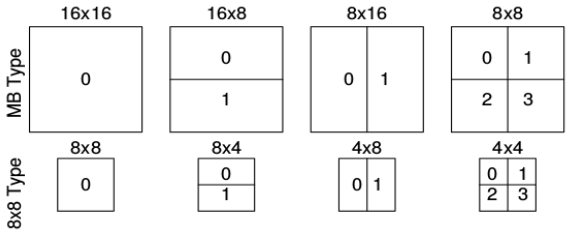

Fig 7:Inter prediction block size type

4.3.5 Deblocking filter

The deblocking filter performs in-loop filtering to reduce blocking artifacts created by image partitioning and quantization. After inverse quantization and inverse transform, the deblocking filter compares the edge values of each 4x4 block with its adjacent block to select the level of filtering. In LETI decoder, Strong or Standard filter is selected according to the block edge, macroblock position in frame and prediction mode. The design algorithm of deblocking filter used in this work is shown with details in [28].

5. H.264 IMPLEMENTATION

5.1 H264 Decoder Partitioning

The H264 decoder process is modeled with the UML / MARTE specification, as shown in the figure 8.Where it's divided into five main functional parts: Entropy Decoder (ED), Inverse Quantization (INVQ), Inverse Transform (IDCT), Inverse prediction (INV-Pred) and and Deblocking Filter (DF). After decoding Network Abstraction Layer (NAL) parameters, the data elements are entropy decoded, that is divided into 3 components: Exp-Golomb is used to extract syntax elements such as the prediction mode and the quantization parameter, CAVLD is used to reconstruct and to reorder data on 4x4 blocks of 16 integers. Incoming maclocks are analyzed with an inter-prediction and intra-prediction in order to increase the coding efficiency by finding redundant information. In the intra-prediction, it determine predicted pixels according to the prediction mode and the macroblock position within a frame (MBX, MBY). Neighboring pixels are generated by a suppliant component called "Neighboring pixels". In the intra-prediction, macroblocks are decoded by a vector, called motion vector, the component Intra/Inter selects the right prediction. The Sub-component consists in subtracting the redundant information from the initial frame. The component Inverse quantization input is 16 coefficients of a block seized 4x4 the CAVLC outputs, component Inverse Transform used Inverse Discrete Cosine Transform (DCT) Inverse transform step is applied for each 4x4 block. Deblocking filter is executed at the end of the decoding process in order to reduce the edging effect between macroblock borders 4X4 after adding the coefficients predicted and transformed by the component addition.

Whereas components, Inverse Quantization, Inverse Transform and Deblocking filter correspond contain intensive data-parallel computations. The Repetitive Structure Modeling (RSM) package of MARTE offers suitable concepts to describe such computations.

5.2 H264 Decoder Functional Partitioning

According the MARTE model in figure8, the video decoding application is composed of six tasks called Decoding Exp-Golomb (Exponential-Golom), MB-Header, Context-based Adaptive Variable Length Decoding (CAVLD), Inverse Quantization (INVQ), Inverse Transform (INVT), Inverse prediction (INV.Pred) and Deblocking filter (DBfilter). As shown in figure 9.

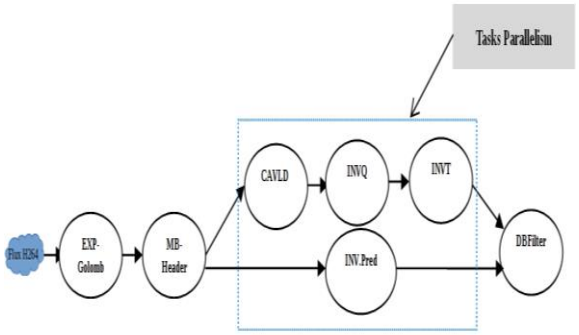

Fig 9:H.264 decoder task graph

Exp-Golomb and MB-Header are running in the first place and sequentially because of data dependency. CAVLD, Inverse quantization and inverse transform are running sequentially because of data dependency. Inverse prediction task can be done in parallel with CAVLD, inverse quantization and inverse transform (task Parallelism). Inverse prediction and inverse transform tasks should be completed before running DBfilter.

5.3 H264 Decoder Data Partitioning

5.3.1 Low-level parallelization: decode macroblocks in parallel

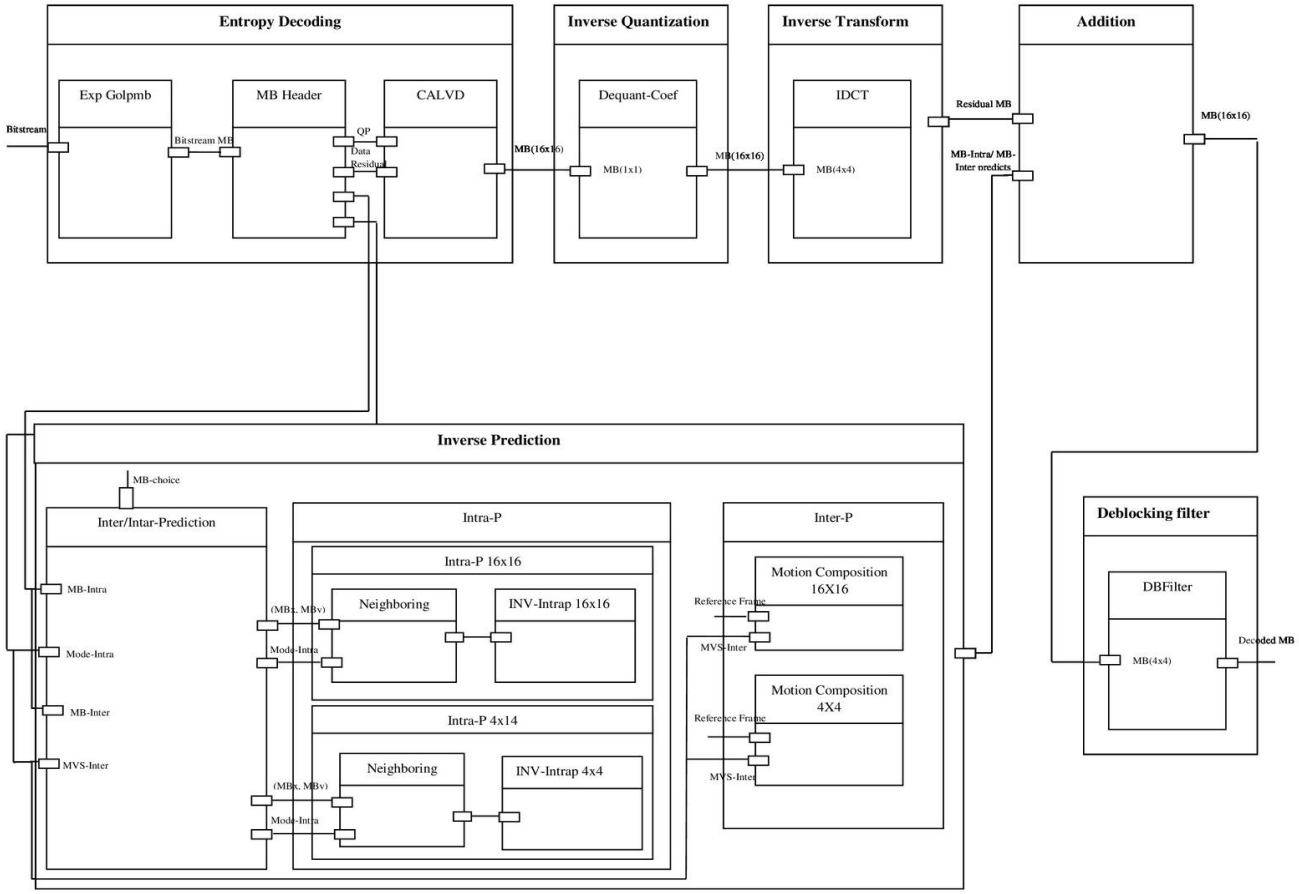

Fig 8: Video application modeling based on the H.264 decoder

In H.264, there are 3 types of macroblocks: I-, P-, and B frames. An I-MB uses intra-prediction., each MB is predicted based on adjacent blocks from the same slice of frame. A P-MB (Predicted macroblock) uses motion estimation as well as intra-prediction and depends on one or more macroblocks from previously decoded frames. Motion estimation is used to exploit temporal correlation between frames. Finally, B-MB (Bidirectionally predicted macroblocks) uses bidirectional motion estimation and can depend on previous frames as well as future frames.

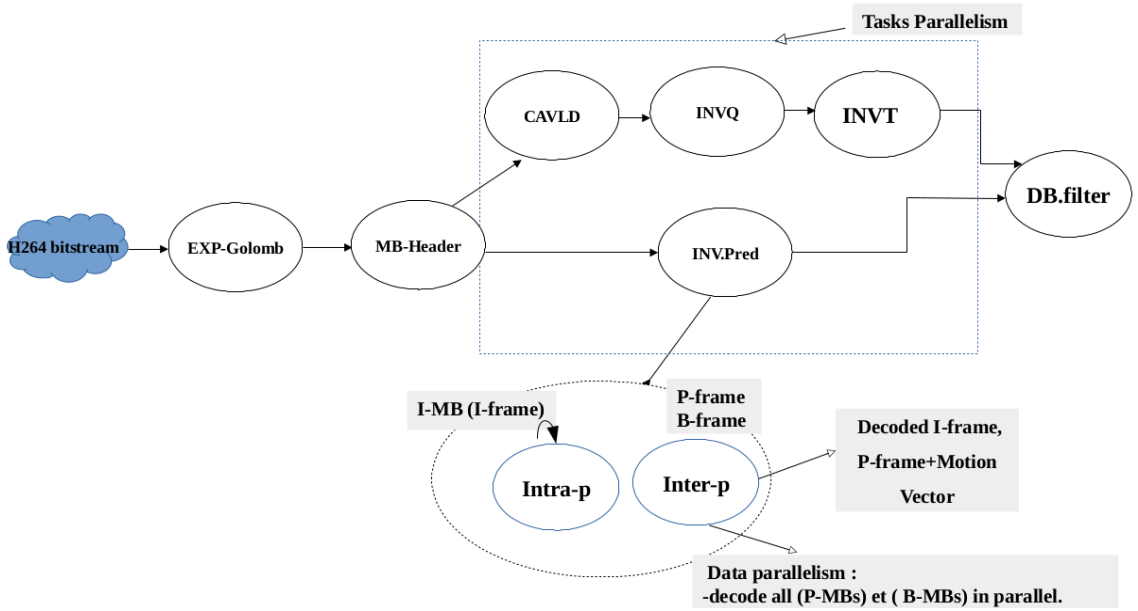

Fig 10:H.264 decoder task graph with Data parallelism

In our work we use the frame sequence I-, P-, and B frames, for more details see section 4.2. Each frame is subdivided into 16x16 or 4x4macroblocks. For each frame sequence we have a single frame of type Ï, we start with coded I-macroblocks

sequentially then P-macroblocks and B-macroblocks are coded in parallel. As shown in figure 10.

5.3.2 High-level parallelization: decode several slices in parallel

An H.264 video stream, an image can be decomposed into sub-images (slices). Idea behind is to treat slices together in parallel which reduces the execution time. Compressed stream processing tasks (Exp-Golomb and MB-Header) cannot be parallelized, the other processing tasks, they are duplicated for each slice. The task graph obtained is shown in Figure 11.

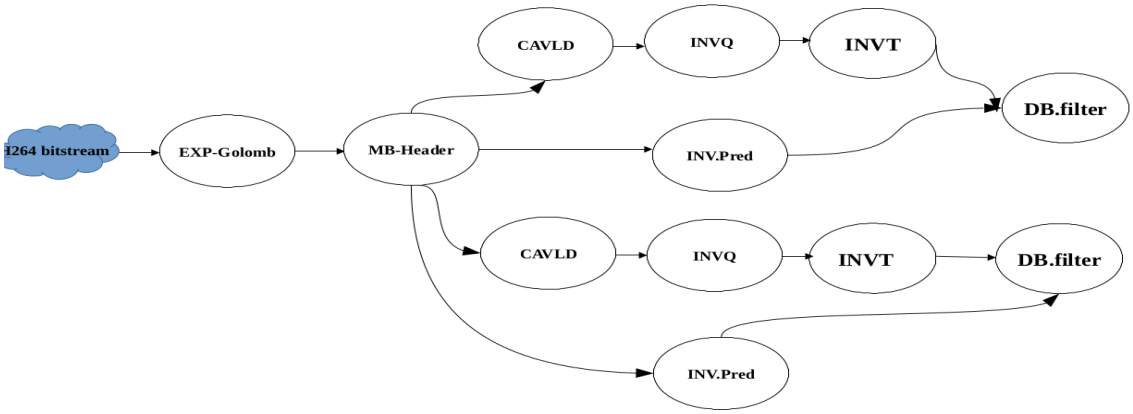

Fig 11: Task graph of the video decoder application with duplication of processing tasks

5.4 Implementation of H264 Decoder on CPUs / GPUs

In our work we are interested by news type of architecture such as heterogeneous architecture GPGPU (multi-CPUs/multi-GPUs) based on Heterogeneous System Architecture (HSA). In HSA, the CPU processor and GPU processor run together and in the same level, data access for the GPU is direct via shared heterogeneous memory.

Exp-Golomb and MB_Header tasks are executed in the first and sequentially because of data dependency on CPU. These two tasks are executed in the same to cancel the data transfer time.

CAVLD, Inverse quantization and inverse transform are running sequentially because of data dependency. These tasks are executed on GPU (first GPU) because this latter contain intensive data-parallel computations and The Repetitive Structure) in order to benefit from advantages of GPU (parallel computing).

Inverse prediction task can be done in parallel with C AVLD, inverse quantization and inverse transform (task Parallelism). This task is executed on GPU (second GPU). The I-macroblocks are coded sequentially on a GPU core, P and B macroblocks are coded in parallel on other GPU cores.

Inverse prediction and inverse transform tasks should be completed before running DBfilter. So DBfilter can be run on either GPU1 or GPU2.

6. EXPRIMENTAL AND RESULTS

6.1 Simulations

The H.264 reference software, JM [18], is an open source implementation used as a reference implementation for the H.264 standards. In our research, we modified the JM [18] source code of the H.264 decoder in order to decode macroblocks in parallel (P et B) and I sequentially, partition the source code into sub-code and run them on the different processor using the PThread library in C programming language. Our H.264 implementation is executed in real HSA architecture by emulator Multi2Sim [21], a cycle-accurate simulator for multicore x86 and graphics processors. Cache and memory configurations comply with common x86 processors that are available nowadays in many Intel [32] or AMD [29] processor chips. Each core has a private L1 cache of 512 KB and All other cores have a shared L2 cache of 2 MB and shared memory between the two processors CPU and GPU. We simulate the execution of our parallel H.264 decoder using 2 GPU multicore (AMD Evergreen) and CPU multicore (x86). We perform simulation experiments of the H.264 OpenCL version on the AMD Evergreen GPU family with the configurations of the AMD Radeon 5870 GPU [31].We gather statistics using 15 video sequences with HD resolution is performed for the H.264 decoding process of 60 frames for each video sequence.

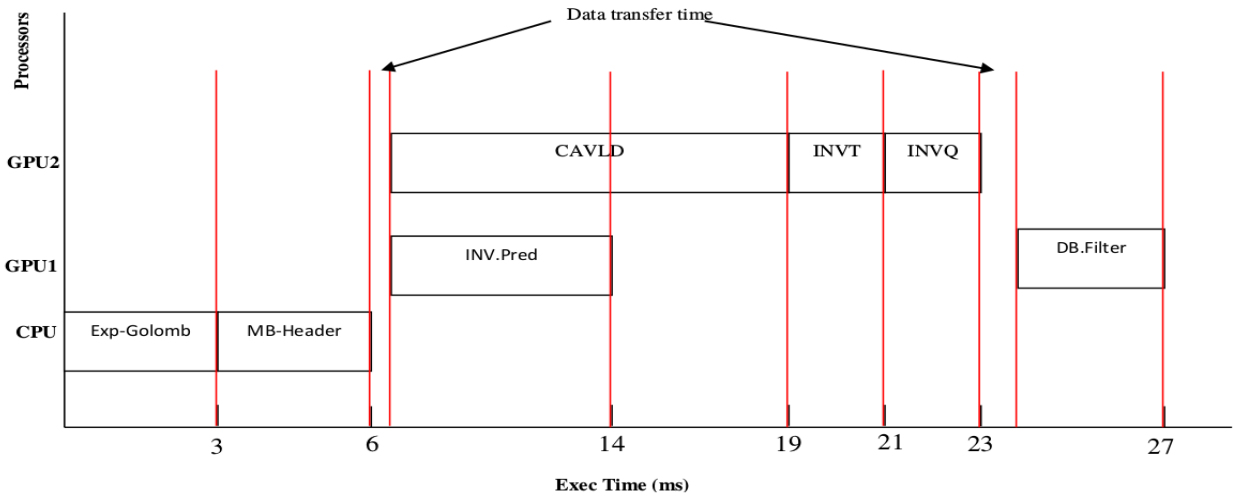

Fig 12: Execution time of the H264 decoder implementation on heterogeneous architecture HAS (multiCPU/multiGPU)

6.2 Results

Execution times with different processors CPU/GPU using HD resolutions are illustrated in figure 12.

The execution time for each task of the H264 video decoder on each type of processor taking into account the different parallelism (task parallelism, data parallelism) is shown in figure 12. This is the first work that deals with the implementation of H264 on heterogeneous architecture HAS (multiCPU/multiGPU) taking into account the different parallelism (task parallelism, data parallelism) and data dependence.

Comparing our result with previous work [13] and [16], our approach gives a better performance in terms of execution time for each H264 decoder task and for entire application.

7. CONCLUSION

This paper illustrated the design of the H.264 encoder on a heterogeneous architecture HAS (multiCPU/multiGPU). A high-level modeling approach based on the standard MARTE profile has been adopted. The obtained model has been analyzed by considering different parallelism, data dependence and characteristics processors (CPU/GPU). Our approach gives a better performance in terms of execution.

Finally, from a practical point of view, a complete implementation of our approach remains to be done (implementation of different slice in parallel).In the future, we are planning to integrate the idea of automatically mapping and scheduling multimedia applications such as H264 onto heterogeneous architectures taking into account different parallelism (task parallelism, data parallelism) and data dependence and the cost of data transfer.

8. REFERENCES

[1] C. Nvidia. Compute unified device architecture programming guide.

[2] K. Corporation. The OpenCL Language. www.khronos.org/opencl, 2011.

[3] AISO/IEC. International standard. Part 10: Advanced video coding,

[4] JCT-VC. High efficiency video coding (HEVC) text specification draft 8. 10th Meeting: Stockholm, SE, 1120 July2012.

[5] C. S. Kannangara and I. E. G. Richardson and M. Bystrom and J. Solera and Y. Zhao and A. Maclennan Complexity reduction of H.264 using Lagrange Optimization Methods. IEE VIE 2005, Glasgow, UK, 2005.

[6] A. Gurhanli and S. Hung. Coarse grain parallelization of h.264 video decoder and memory bottleneck in multi-core architectures. International Journal of Computer Theory and Engineering vol. 3, no. 3, pages 375–381, 2011.

[7] K. Nishihara, A. Hatabu, and T. Moriyoshi. Parallelization of h.264 video decoder for embedded multicore processor. ICME, pages 329–332, 2008.

[8] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hooger-brugge, M. Alvarez, and A. Ramirez. Parallel h.264 decoding on an embedded multicore processor. HiPEAC, pages 404–418, 2009.

[9] J. Chong, N. Satish, B. Catanzaro, K. Ravindran, and K. Keutzer. Efficient parallelization of h.264 decoding with macro block level scheduling. ICME, pages 1874–1877, 2007.

[10] E. Van Der Tol, E. Jaspers, and R. Gelderblom. Mapping of h.264 decoding on a multiprocessor architecture. Image and Video Communications and Processing, pages 707–718, 2003.

[11] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro. H.264/avc baseline profile decoder complexity analysis. IEEE Trans. Circuits Syst. Video Techn., 13(7):704–716, 2003.

[12] K. Sihn, H. Baik, J. Kim, S. Bae, and H. Song. Novel approaches to parallel h.264 decoder on symmetric multicore systems. Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 09, pages 2017–2020, Washington, DC, USA, 2009. IEEE Computer Society.

[13] Elias Baaklini and all, H.264 Parallel Optimization on Graphics Processors, MMEDIA 2013 : The Fifth International Conferences on Advances in Multimedia

[14] OMG, “MARTE Web Site,” 2009, www.omgmarte.org.

[15] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. Multi2Sim: A Simulation Framework for CPU-GPU Computing. Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques, Sep., 2012.

[16] R. Bonamy Modélisation, exploration et estimation de la consummation pour les architectures hetéerogénes reconfigurables dynamiquement HAL Id: tel-00931849 https://tel.archives-ouvertes.fr/tel-00931849v2Submitted on 17 May 2014.

[17] FFmpeg project. http://www.ffmpeg.org/.

[18] K. Suhring. H.264 reference software. http://bs.hhi.de/ suehring/tml/.

[19] TarunIyer (30 April 2013). "AMD Unveils its Heterogeneous Uniform Memory Access (hUMA) Technology". Tom's Hardware.

[20] George Kyriazis (30 August 2012). Heterogeneous System Architecture: A Technical Review (PDF) (Report). AMD.

[21] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. Multi2Sim: A Simulation Framework for CPU-GPU Computing. Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques, Sep., 2012.

[22] Wiegand (Ed.), T, “ Draft ITU-T Recommendation H.264/AVC and Draft ISO/IEC 14496-10 AVC”, Joint Video Team of ISO/IEC JTC1/SC29/WG11 &ITU-T SG16/Q.6 Doc. JVT-G050, Mar.

[23] Richardson, I.E.G.: Video Codec Design: Developing Image and Video Compression Systems. John Wiley and Sons (2002).

[24] Flierl, M., Girod, B.: Generalized B Pictures and the Draft H. 264/AVC Video-Compression Standard. IEEE Transactions on Circuits and Systems for Video Technology 13(7), 587–597(2003).

[25] Soon-kak Kwon, A. Tamhankar, K.R. Rao, “Overview of H.264/AVC / MPEG-4 Part 10”, Journal of Visual Communication and Image Representation, Vol. 17, No 2 , pp 186–216, Apr. 2006.

[26] Kessentini A., Kaaniche B., Werda I., Samet A., Masmoudi N., “Low complexity intra 16x16 prediction for H.264/AVC” International Conference on Embedded Systems & Critical Applications ICESCA,2008, Tunisia.

[27] Werda I., Chaouch H, Samet A, Ben Ayed M-A, Masmoudi N., “Optimal DSP Based Integer Motion Estimation Implementation for H.264/AVC Baseline Encoder”, The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010.

[28] Damak T., Werda I., Masmoudi N., Bilavarn S., “Fast prototyping H.264 deblocking filter using ESL Tools”, 2011 8th International Multi-Conference on Systems, Signals &Devices,SSD.

[29] AMD Opteron Processor Family. http://www.amd.com/.

[30] AMD Evergreen Family Instruction Set Arch. (vl.Od).

[31] http://developer.amd.com/sdks/amdappsdk/documentation/.

[32] Intel Core Processor Family. http://www.intel.com